SALZA : mesure d'information universelle entre chaînes pour la classificationet l'inférence de causalité

Marion Revolle

Communauté
UNIVERSITÉ Grenoble Alpes

THÈSE

Pour obtenir le grade de

DOCTEUR DE LA COMMUNAUTE UNIVERSITE GRENOBLE ALPES

Spécialité : **Signal Image Parole et Télécom**

Arrêté ministériel : 25 mai 2016

Présentée par

Marion REVOLLE

Thèse dirigée par **Nicolas LE BIHAN,** Directeur de recherche, CNRS,
et
codirigée par **François CAYRE**, Maître de conférences, Grenoble INP

préparée au sein du **Laboratoire Grenoble Image Signal Parole Automatique (GIPSA-lab)**
dans l'école doctorale **Électronique Électrotechnique Automatique Traitement du Signal (EEATS)**

SALZA : mesure d'information universelle entre chaînes pour la classification et l'inférence de causalité

SALZA: universal information measure between strings for classification and causality inference

Thèse soutenue publiquement le **25 octobre 2018**,
devant le jury composé de :

Steeve ZOZOR
Directeur de recherche, CNRS, Président
Thierry LECROQ
Professeur, Université de Rouen-Normandie, Rapporteur
Jean-Marc GIRAULT
Professeur, ESEO Group, Rapporteur
Pierre CHAINAIS
Professeur, École Centrale de Lille, Examinateur
Nicolas LE BIHAN
Directeur de recherche, CNRS, Directeur de thèse
François CAYRE
Maître de conférences, Grenoble-INP, Co-directeur de thèse

Remerciements

Je tiens tout d'abord à remercier les membres du jury pour avoir examiné mon travail et s'être déplacé jusqu'à Grenoble pour assister à ma soutenance. Vos retours constructifs et très intéressants donnaient envie de repartir pour trois nouvelles années. Je remercie tout particulièrement Steeve pour sa disponibilité et les échanges enrichissants que nous avons pu avoir au cours de ces trois années.

Je tiens également à remercier Nicolas et François. Vous avez su comprendre les moments où cela n'allait pas et essayer de m'épauler. C'est grâce à vous que le chemin n'a pas trop été semé d'embuche.

Ma thèse ne se résume pas aux quelques pages scientifiques qui vont suivre, elle est aussi composée d'une multitude de rencontres toutes plus formidables les unes que les autres. Je ne pourrais pas remercier tout le monde cependant vous êtes tellement géniaux que personne ne s'en offusquera mais voici quelques dédicaces. Déjà Pierre, merci d'avoir permis à Miya de rencontrer Loustic et de mettre tous les chiens renifleurs au chômage. Raph, merci pour cette accueille endiablé au Gipsa, les week-ends à la bergerie (même si je me couche trop tôt pour les étoiles) et encore mille fois mercis d'avoir relu ma thèse ! Enfin je remercie Jeanne, ma demi soeur de thèse, pour ta bonne humeur, ton engagement, ta force et aussi pour nous avoir marié avec Fabrice de la plus belle manière qui soit.

Enfin je tiens tout particulièrement à remercier trois personnes sans qui cette thèse n'aurait jamais abouti. Je remercie ma mère qui m'a toujours soutenue et poussée à faire ce que j'aimais. Pas grand monde ne lit une thèse de A à Z, alors lire toute la mienne alors que l'informatique n'est pas du tout ton domaine est une des plus belles preuves d'amour que je n'ai jamais eu. Je remercie aussi Marc sans qui la moitié du contenu scientifique n'aurait pas existé. Merci d'avoir lu, compris et participé à ce que je faisais. Nos échanges ont été la pierre angulaire de ce qui va être présenté et c'est grâce à eux que je n'ai jamais laissé tout tomber. Enfin merci à Fabrice, mon mari. En effet, pour avoir une thèse, il a fallu passer par une école d'ingénieurs, pour intégrer cette école il a fallu préparer les concours et pour rentrer en classe préparatoire il fallait le bac. Et déjà à cette époque tu étais là. C'est grâce à toi que j'ai réussi à franchir toutes ces étapes. C'est toi qui m'as relevée quand je suis tombée, c'est toi qui m'as arrêtés quand ça devenais trop, ... Bref c'est toi qui t'occupes si bien de moi depuis si longtemps, ce manuscrit est donc pour toi.

Table des matières

Table des figures

Introduction

Problématique

Depuis de nombreuses années, grâce au monde numérique, de plus en plus de données sous forme de chaînes de caractères sont à notre disposition. Ces données sont extrêmement variées, allant du signal électroencéphalographique quantifié à des textes littéraires numérisés en passant par de la musique en format MIDI. Très vite, le besoin d'une mesure de description universelle de ces chaînes s'est fait ressentir. C'est ainsi que dans les années 1950, Shannon introduit l'entropie qui mesure la quantité moyenne d'information de chaque caractère. Plus la chaîne est aléatoire plus l'entropie de Shannon est élevée et inversement plus la chaîne est ordonnée plus l'entropie de Shannon est proche de zéro. Cependant cette mesure repose sur la théorie des probabilités et il n'est pas toujours possible ou pertinent d'estimer une densité de probabilités discrète sur tous les types de données. C'est ainsi que dans les années 1960, une description universelle des chaînes de caractères indépendante des probabilités a vu le jour. Cette définition est aujourd'hui connue sous le nom de complexité de Kolmogorov et elle repose sur une idée très simple : une chaîne est complexe quand il n'en existe pas une description courte.

C'est au cours du 20ème siècle que les prémices de l'informatique ont été définis puis développées : formalisation de la machine de Turing, puis essor des ordinateurs ou encore théorie des fonctions récursives... Cette période de grande émulation a été un moment propice pour la recherche, à tel point que trois chercheurs ont formalisé à peu près en même temps et dans des lieux différents ce qu'on appelle aujourd'hui la complexité de Kolmogorov, et qui aurait pu très bien s'appeler la complexité de Solomonoff, Kolmogorov et Chaitin.

En novembre 1960 R.J. Solomonoff publie un rapport technique de la société Zator [Sol60] qui présente les fondamentaux de la théorie algorithmique de l'information. Dans ce rapport, il montre comment utiliser l'algorithmique comme moyen de surmonter les problèmes sérieux associés à l'application de la règle de Bayes dans les statistiques et il prouve le théorème d'invariance que nous aborderons dans le chapitre I (théorème I.1.1). Le travail de Solomonoff a ainsi conduit à une nouvelle approche des statistiques : les probabilités algorithmiques ou dites de Solomonoff.

Qulques années plus tard, Kolmogorov travaille lui aussi sur l'algorithmique. Il introduit, en 1965 [Kol65], une mesure de complexité qui représente le contenu de l'information individuelle d'une chaîne de caractères. Il démontre à son tour le théorème d'invariance I.1.1. En plus de résoudre la question du caractère aléatoire des objets, Kolmogorov visait à refonder la théorie probabiliste de l'information par une approche algorithmique du contenu informationnel des séquences individuelles. Il ne découvre les travaux de Solomonff que deux ans plus tard en 1963, et écrit alors dans [Kol68] : "Je suis arrivé à des conclusions similaires, avant de prendre conscience de l'oeuvre de Solomonoff, en 1963-1964".

Exactement à la même période et en parallèle de Kolmogorov, Chaitin définit aussi une

complexité algorithmique [Cha66] mais ne démontre le théorème d'invariance I.1.1 qu'en 1969. Il écrit alors en comparant sa découverte à celle de Kolmogorov : "cette définition [de la complexité de Kolmogorov] a été indépendamment proposée vers 1965 par A.N. Kolmogorov et moi ...".

La complexité de Kolmogorov est définie aujourd'hui par la taille du plus petit programme capable de générer la chaîne de caractères. Cette mesure est fondatrice de la théorie algorithmique de l'information tout comme la théorie probabiliste de l'information repose sur l'entropie de Shannon. La théorie algorithmique de l'information vise ainsi à quantifier et à qualifier le contenu en information d'une chaîne ou d'un ensemble de chaînes. Nous chercherons donc la description d'une chaîne la plus courte possible, mais aussi la description d'une chaîne la plus courte possible connaissant une autre chaîne ou encore la description la plus courte possible de deux chaînes simultanément. Nous définirons ainsi l'information mutuelle algorithmique qui mesure l'indépendance algorithmique entre deux chaînes, théorie qui a été formalisée et synthétisée dans l'ouvrage [LV08].

L'inconvénient majeur de la complexité de Kolmogorov est qu'elle n'est pas calculable en un temps fini. Dès 1976, Lempel et Ziv proposent une nouvelle complexité pour pallier cet inconvénient, en réduisant le nombre d'opérations utilisées dans le programme de la complexité de Kolmogorov. Cette nouvelle complexité est appelée complexité de Lempel-Ziv. Elle est à la base de nombreux programmes de compression sans perte que nous utilisons dans notre quotidien comme `GZIP`. En effet, le but de la compression est de trouver la plus petite description possible de la chaîne de caractères à compresser, ce qui n'est donc pas sans rappeler la définition de la complexité.

Nous pouvons en déduire deux façons d'estimer la complexité de Kolmogorov : par la complexité de Lempel-Ziv ou l'utilisation de compresseurs. La complexité de Lempel-Ziv a par exemple été utilisée en compression, en classification ([ZM93]), en analyse de signaux biomédicaux ([Abo+06],[Li+08],[IM+15]) ou encore en mesure de causalité ([SJS10]). Les compresseurs ont beaucoup été utilisés pour la classification ([CV05]) dans des domaines très variés comme l'imagerie hyperspectrale ([VDG12]), l'ADN ([CV05]), les langues ([Li+04]) ou même encore la musique ([FDK15]). Cependant ces deux estimateurs de complexité sont mal définis pour estimer la complexité conditionnelle et jointe. Il est donc difficile d'étendre ou d'appliquer la complexité de Lempel-Ziv ou les compresseurs à la théorie algorithmique de l'information.

Contribution

Partant de ce constat, nous proposons un nouvel estimateur de complexité que nous avons appelé `SALZA` et qui est basé sur la complexité de Lempel-Ziv. Tout comme Kolmogorov qui a d'abord introduit la complexité conditionnelle, nous décidons d'élaborer notre estimateur de façon conditionnelle. Étant une mesure d'information, nous avons donc pu étendre `SALZA` à la théorie algorithmique de l'information. En plus de l'information mutuelle algorithmique, nous proposons un nouvel encodage pour l'estimation de la complexité jointe grâce à `SALZA` ce qui nous permet d'introduire pour la première fois l'information dirigée algorithmique. L'implémentation et la bonne définition de notre mesure permettent

alors un calcul efficace des grandeurs de la théorie algorithmique de l'information.

Benett et al [Ben+98] proposent en 1998 une distance basée sur la complexité de Kolmogorov appelée distance d'information normalisée (*NID*). Les compresseurs sans perte usuels ont alors été utilisés par Cilibrasi et Vitányi [CV05]) pour former une métrique calculable universelle très populaire : la distance de compression normalisée (*NCD*). Dans le cadre de cette application, nous proposons notre propre métrique, la *NSD*, et montrons qu'il s'agit d'une semi-distance universelle sur les chaînes de symboles. Les différentes classifications obtenues grâce à ces deux métriques montrent que la *NSD* surclasse la *NCD* en s'adaptant naturellement à davantage de diversité des données et en utilisant le bon conditionnement `SALZA`.

De part les qualités de prédiction universelle de la complexité de Lempel-Ziv, nous explorons ensuite les questions d'inférence de causalité algorithmique. Dans un premier temps, les conditions algorithmiques de Markov sont rendues calculables grâce à SALZA. Puis en définissant, pour la première fois, l'information dirigée algorithmique, nous définissons la causalité de Granger algorithmique. Nous montrons, sur des données synthétiques et réelles, la pertinence de notre approche.

Plan

La théorie algorithmique de l'information sera présentée dans le chapitre I. Tout d'abord, nous reprendrons succinctement les bases de la théorie probabiliste de l'information pour bien comprendre la démarche et le fondement de la théorie algorithmique de l'information. Nous présenterons ensuite la complexité de Kolmogorov d'une chaîne de caractères qui est la taille du plus petit programme capable de générer cette chaîne. Nous verrons ensuite comment la théorie algorithmique de l'information est bâtie sur la complexité de Kolmogorov. N'étant pas calculable en un temps fini, il n'est pas possible de mesurer les grandeurs algorithmiques. Une première méthode d'estimation de la complexité de Kolmogorov est la complexité de Lempel-Ziv qui restreint les opérations du programme à copier et insérer. Après l'avoir présentée, nous introduirons les différentes versions de la théorie algorithmique de l'information basée sur la complexité de Lempel-Ziv. La deuxième méthode d'estimation consiste à utiliser un compresseur. La complexité de Kolmogorov est alors la taille de la chaîne compressée par le compresseur. Nous verrons alors sous quelles conditions une théorie algorithmique de l'information basée sur la compression peut être définie.

Dans le chapitre II, nous présenterons un nouvel estimateur conditionnel de complexité que nous avons appelé `SALZA` (Signal Analisys with Lempel-Ziv Algorithm) et qui est basé sur la complexité de Lempel-Ziv. Tout d'abord, différentes possibilités pour réaliser une décomposition de Lempel-Ziv conditionnelle sont proposées. Puis nous montrerons qu'il est important de prendre en compte la longueur des opérations `copier` d'une décomposition d'une chaîne et pas uniquement leur nombre comme dans la complexité de Lempel-Ziv. À partir de ces deux constats, nous prouverons que l'estimateur SALZA est bien une mesure d'information ce qui permet de développer une théorie de l'information basée sur `SALZA`. Cette théorie algorithmique de l'information basée sur `SALZA` repose sur trois encodages

différents pour estimer la complexité jointe. Cela nous permettra de définir l'information mutuelle algorithmique et d'introduire pour la première fois l'information dirigée algorithmique. SALZA sera ensuite comparé aux deux autres estimateurs de la complexité de Kolmogorov qui sont la complexité de Lempel-Ziv et les compresseurs. Cela nous permettra de conclure sur l'importance de l'utilisation des longueurs des opérations `copier`.

Dans le chapitre III, la complexité sera utilisée pour réaliser une classification non-supervisée sur des jeux de données variés. Pour cela nous présenterons deux métriques basées respectivement sur la complexité de Lempel-Ziv et les compresseurs. Nous introduirons une semi-distance avec `SALZA` sur la base d'une distance reposant sur la complexité de Kolmogorov (*NID*, Normalized Information Distance). Les matrices de dissimilarité permettent de réaliser une classification ascendante hiérarchique. Les jeux de données très variés (ADN, texte en différentes langues, chaînes de Markov et texte numérisé de la littérature française), permettront de mettre en avant l'efficacité de `SALZA` grâce au choix judicieux de son conditionnement et à l'utilisation des longueurs des opérations `copier`.

Dans le chapitre IV, la complexité sera utilisée pour mesurer les liens entre différentes chaînes de caractères. Le but est d'inférer un modèle graphique sous-jacent capable de rendre compte des relations causales entre les différents processus. Nous aborderons deux méthodes. La première est la condition de Markov qui dit qu'une chaîne de caractère doit être indépendante de ses non-descendantes connaissant ses parents. Nous montrerons sur des données réelles la pertinence de l'approche algorithmique des conditions de Markov. La seconde méthode d'inférence de causalité, que nous présenterons, sera la causalité de Granger. L'estimation la causalité de Granger algorithmique se fera alors grâce à l'information dirigée algorithmique basée sur `SALZA`. Un jeu de données synthétique permettra de valider notre approche.

Notations

Nous utiliserons le système de notation suivant dans ce manuscrit.

Nous travaillons sur un alphabet fini $\mathcal{A}$ de taille α. Les caractères de $\mathcal{A}$ sont notés a_i $(1 \leq i \leq \alpha)$. Une chaîne de caractères X finie de taille n définie sur $\mathcal{A}$ est la succession de n caractères de $\mathcal{A}$. Elle est entièrement spécifiée en écrivant $X = x_1x_2...x_n$, où $\forall i = 1...n\ x_i \in \mathcal{A}$. La longueur d'une chaîne X est le nombre de caractères de X. Elle est notée $l(X)$. La chaîne vide est notée Λ et par définition $l(\Lambda) = 0$

Dans ce travail, nous condidérons des séquences définies sur divers alphabets. Par exemple, nous utiliserons des séquences d'ADN pour lesquelles l'alphabet $\mathcal{A} = \{C, A, T, G\}$ est constitué des quatre bases nucléiques : cytosine, adénine, thymine et guanine. Nous travaillerons aussi sur des textes numérisés pour lesquels l'alphabet est composé des 26 lettres de l'alphabet latin et des signes de ponctuation usuels. Les chaînes utilisées sont détaillées dans l'annexe D.

$\mathcal{A}^*$ est l'ensemble de toutes les chaînes de longueur finie sur $\mathcal{A}$. L'ensemble de toutes les chaînes finies de taille n sur $\mathcal{A}^*$ est noté $\mathcal{A}^n = \{X \in \mathcal{A}^* | l(X) = n\}$, $\forall n \in \mathbb{N}$. Pour désigner une sous-chaîne de X qui commence à la position i et se termine à la position

j, nous utilisons la notation standard $X(i,j)$. Si $i < j$ alors $X(i,j) = x_i x_{i+1} ... x_j$, sinon $X(i,j) = \Lambda$. La concaténation d'une chaîne $X \in \mathcal{A}^n$ et d'une chaîne $Y \in \mathcal{A}^m$ est la chaîne $XY \in \mathcal{A}^{m+n}$.

Chapitre I

Théorie algorithmique de l'information : complexité, mesure d'information et compression

Sommaire

Lorsque Kolmogorov définit la complexité algorithmique en 1965 [Kol65], son but est double. Dans un premier temps, il veut pouvoir mesurer à quel point une chaîne est aléatoire. Puis dans un deuxième temps, il veut refonder la théorie probabiliste de l'information par une approche algorithmique du contenu informationnel des séquences individuelles. Ainsi, pour bien saisir le sens de sa démarche, il faut tout d'abord comprendre comment est définie l'entropie de Shannon qui est à la base de la théorie probabiliste de l'information (section I.1.1). Une fois ces notions définies, nous reprenons le travail de Kolmogorov en détail pour définir la complexité algorithmique et pour expliquer comment se construit la théorie algorithmique de l'information (sections I.1.2 et I.1.3). Le défaut majeur de la complexité de Kolmogorov est qu'elle n'est pas calculable en un temps fini sur une machine de Turing. De ce fait, plusieurs approximations ont été proposées afin de conserver les principaux avantages tout en la rendant opérationnelle.

Une première approximation de la complexité de Kolmogorov est la complexité de Lempel-Ziv. Cette complexité, introduite en 1976 par Lempel et Ziv dans [LZ76], repose sur deux opérations (`copier` et `insérer`) et est calculable sur une machine de Turing. Nous expliquerons donc en détail son processus de production (section I.2). Comme la complexité de Lempel-Ziv est une mesure d'information [SJS10], elle peut être la base d'une théorie algorithmique de l'information. Nous verrons alors qu'il existe différentes versions de cette

théorie algorithmique de l'information basée sur la complexité de Lempel-Ziv selon les besoins des auteurs.

Une deuxième façon d'approcher la complexité de Kolmogorov est la taille d'une chaîne de caractères compressée par un compresseur. Si le compresseur vérifie certaines conditions, présentées dans la section I.3 (voir [CV05]), il est alors possible d'assimiler la taille du fichier compressé à la complexité de Kolmogorov puis de définir une théorie algorithmique de l'information basée sur la compression.

I.1 Théorie algorithmique de l'information

L'entropie de Shannon, basée sur les probabilités est la mesure la plus connue qui permette de quantifier l'information contenue dans une chaîne de caractères X. Il existe cependant une autre entropie basée sur la combinatoire et indépendante de toute hypothèse probabiliste. C'est en partant des limites respectives des deux entropies que Kolmogorov a présenté sa complexité en 1965 [Kol65]. Cependant, complexité et entropie sont équivalentes lorsque la taille de la chaîne est très grande. Ainsi il est possible de construire la théorie algorithmique de l'information fondée sur la complexité de Kolmogorov comme la théorie probabiliste de l'information fondée sur l'entropie de Shannon. Nous proposons ici de mettre en perspective ces différentes versions de la théorie de l'information.

Tout comme en théorie probabiliste de l'information, la restriction des programmes et des séquences à un alphabet binaire n'implique aucune perte de généralité ([LV08]), c'est pourquoi $\mathcal{A} = \{0, 1\}$ dans toute cette section. Utiliser un alphabet plus grand laisse tout ce qui suit vrai à un facteur logarithmique près, lié justement à la taille de l'alphabet.

I.1.1 Entropie de Shannon et entropie combinatoire

En 1965, Kolmogorov s'interroge sur la mesure de l'information contenue dans une chaîne X à propos d'une autre chaîne Y. Il existe déjà à cette époque deux approches : combinatoire et probabiliste. Ces quantités qui mesurent l'information sont appelées des entropies. La compréhension de leur fonctionnement ainsi que leurs limites est centrale, car c'est en partant de ces dernières que Kolmogorov définit une nouvelle approche algorithmique.

I.1.1.1 Entropie combinatoire

La première mesure d'information qu'étudie Kolmogorov n'est pas la définition probabiliste de l'entropie communément retenue aujourd'hui, mais l'entropie combinatoire. Soit a un caractère de l'alphabet $\mathcal{A}$, alors l'entropie combinatoire de a est notée $H_c(a)$ et est égale à $log_2\alpha$, où α est la taille de l'alphabet $\mathcal{A}$. Si a est fixé, alors une information est transmise. Cette information est donnée par $I_c = log_2\alpha$.

D'une manière plus générale, prenons un alphabet $\mathcal{A}$ et un certain nombre de règles

grammaticales sur cet alphabet : par exemple le caractère a_1 ne peut pas être suivi du caractère a_3. Alors l'entropie combinatoire d'une chaîne de n caractères est égale au logarithme en base deux du nombre de chaînes de taille n sur $\mathcal{A}$ qui respectent les règles grammaticales. Par exemple, considérons un alphabet ternaire (c'est-à-dire composée de trois caractères : '0','1','2'). La règle grammaticale est qu'il ne peut y avoir qu'un seul caractère '2' dans la chaîne. Le nombre de chaînes différentes composées de k '0' et '1' et d'un '2' est $2^k(k+1)$. En prenant une chaîne dans cet ensemble, nous transmettons l'information $I_c = k + log_2(k+1)$. Cette information est le nombre de bits nécessaire pour coder ce message en base ternaire dans une base purement binaire (c'est-à-dire avec deux caractères : '0','1').

Pour illustrer son propos, Kolmogorov donne l'exemple de la mesure de l'entropie du langage. En effet, une langue peut être décrite par son dictionnaire, dans son cas le dictionnaire russe d'Ozhegov, et des règles grammaticales. Ainsi deux chercheurs russes, M. Ratner et N. Svetlova [Kol65] [1] ont voulu estimer l'entropie moyenne par caractère de textes russes. Pour cela ils ont pris N textes grammaticalement corrects de longueur n caractères (espace compris). Ils ont alors estimé qu'il faut asymptotiquement environ $1,9$ bits par caractère. Cependant, une méthode expérimentale permet de mesurer cette entropie. Elle consiste à faire deviner à une personne le dernier mot d'un texte. La valeur mesurée de cette manière est bien plus faible que $1,9$ bits par caractère. Cet écart est néanmoins tout à fait naturel : les textes littéraires répondent à beaucoup d'autres exigences que les seules règles grammaticales. Cela met en avant le point faible de cette méthode : il est difficile de modéliser des règles très complexes et cette entropie est en pratique très difficilement calculable.

I.1.1.2 Entropie de Shannon : approche probabiliste

La deuxième mesure d'information étudiée par Kolmogorov est l'entropie de Shannon définie en 1948 [Sha48]. Cette approche probabiliste a été développée par Shannon dans le cadre de la théorie de la transmission de l'information dans le cadre des canaux de communication. Ces canaux transportent des informations constituées d'un grand nombre de messages non liés ou faiblement liés, obéissant à des lois probabilistes définies.

Supposons $\mathbf{X}$ une variable aléatoire, le tirage de n valeurs de cette variable aléatoire forme une chaîne notée X. Le i^{eme} caractère x_i a une probabilité $P(\mathbf{X} = x_i) = P(x_i)$ d'apparaître. L'entropie de Shannon de X est donnée par :

$$H(X) = -\sum_{k=1}^{n} P(x_i) log_2 P(x_i). \tag{I.1}$$

Elle représente la quantité moyenne d'information que la source délivre pour chaque caractère et mesure l'incertitude moyenne au sens des probabilités de l'expérience aléatoire $\mathbf{X}$. Si par exemple, $\mathbf{X}$ est en réalité une constante et donne toujours le même caractère x alors l'entropie de Shannon est nulle. En effet nous avons $P(\mathbf{X} = x) = 1$ donc $\forall i \; log_2(P(x_i)) = 0$. De manière générale, plus la source est redondante, plus l'entropie de Shannon est faible.

1. Kolmogorov fait référence à ces travaux dans son article mais ne cite aucun article.

À l'inverse, l'entropie est maximale lorsque tous les caractères sont équiprobables dans le tirage de **X**.

L'entropie de Shannon est la base de la théorie probabiliste de l'information. En effet, il est possible de définir l'entropie jointe de X et Y :

$$H(X,Y) = -\sum_{k=0}^{n} P(x_i, y_i) log_2 P(x_i, y_i). \tag{I.2}$$

À partir de l'entropie jointe, on peut alors définir l'information mutuelle de Shannon comme suit :

$$I_s = H(X) + H(Y) - H(X,Y). \tag{I.3}$$

La quantité I_s mesure la dépendance statistique des variables X et Y.

Dans [LV08], Li et Vitányi soulignent deux lacunes de la théorie probabiliste de l'information.

i La première concerne la théorie classique des probabilités. Prenons l'exemple d'un adversaire qui a une pièce de monnaie non truquée. Après l'avoir lancée une centaine de fois, la pièce est toujours retombée sur le côté face. En voyant cela, nous prétendons que la pièce est truquée. L'adversaire, cependant, fait appel à la théorie des probabilités, qui dit que chaque chaîne de résultats d'une centaine de lancés a exactement la même probabilité d'apparition que toutes les autres chaînes. Ainsi la chaîne de 100 faces a autant de chance de se réaliser que 50 successions de pile et de face. La théorie des probabilités ne nous donne aucune raison de remettre en question un résultat après qu'il se soit produit. Nous ne pouvions que faire le pari qu'il soit très peu probable de voir 100 faces consécutives apparaître. En fait, la théorie des probabilités classique ne peut pas exprimer le caractère aléatoire d'une réalisation d'un processus aléatoire. Elle ne peut qu'exprimer les propriétés attendues d'un résultat d'un processus aléatoire. La réalisation des 100 faces consécutives ne nous permet donc pas de conclure sur le fait que la pièce est truquée ou non.

ii Leur deuxième réserve porte sur le fait que la théorie de l'information de Shannon attribue une quantité d'information à un ensemble de messages possibles. Si tous les k messages de l'ensemble $\mathcal{M}$ sont équiprobables, l'entropie de Shannon est le nombre de bits nécessaires pour compter toutes les possibilités, c'est-à-dire $\forall m \in \mathcal{M},\ H(m) = log_2(k)$. Chaque message m de l'ensemble $\mathcal{M}$ peut donc être communiqué en utilisant $log_2(k)$ bits. Cependant, il ne dit rien sur le nombre de bits nécessaire pour transmettre un message individuel sans prendre en compte l'ensemble $\mathcal{M}$. Par exemple, considérons l'ensemble composé de toutes les chaînes binaires de longueur 1000. Prenons deux chaînes de cet ensemble : c_1 qui représente le chiffre 2^{999} et c_2 composée de 1000 bits aléatoires. La chaîne c_1 peut être encodée en écrivant un '1' puis 999 fois '0'. La description de cette chaîne peut être considérablement compressée grâce à sa grande régularité. La chaîne c_2 de 1000 bits aléatoires a de fortes chances d'avoir besoin d'environ 1000 bits pour être encodée. Cependant avec la mesure d'entropie de Shannon, nous avons besoin de 1000 bits en moyenne pour encoder une chaîne dans l'ensemble $\mathcal{M}$ composé de toutes les chaînes binaires de longueur 1000. Ainsi pour Shannon c_1 et c_2 ont la même entropie : l'entropie de Shannon mesure l'information des chaînes selon l'ensemble $\mathcal{M}$. La différence d'information entre c_1 et c_2 en tant que chaîne individuelle n'est pas prise en compte.

Kolmogorov émet quant à lui des réserves plus philosophiques. La première est que souvent la probabilité d'un caractère est associée à la fréquence empirique de ce dernier dans une chaîne temporelle assez longue. Ceci est vrai dès lors que la chaîne est assez longue, mais l'imprécision, faite en amalgamant la théorie mathématique des probabilités et l'événement réel aléatoire, le laisse dubitatif. La deuxième réserve qu'il émet porte sur le sens du modèle probabiliste. Est-ce qu'il y a un réel sens à chercher un modèle probabiliste pour un roman ? Il faudrait alors soit inclure ce roman dans l'ensemble des romans possibles soit postuler une distribution de probabilité sur cet ensemble. Tout comme nous l'avons fait remarquer avec l'entropie combinatoire, il est assez difficile de trouver un ensemble de tous les romans possibles. Un auteur ne doit pas seulement respecter des règles orthographiques, mais aussi un style ou tout simplement un sujet précis. L'autre solution, qui est d'émettre un modèle probabiliste sur un roman, n'est pas non plus sans danger. Par exemple, l'auteur Georges Perec a réussi à écrire tout son roman, *La Disparition* ([Per69]), sans utiliser la lettre 'e' qui est pourtant la lettre la plus probable en français.

En résumé

Kolmogorov [Kol65] présente deux mesures de la quantité d'information contenue dans une chaîne X à propos d'une autre chaîne Y.

Entropie combinatoire :

- *Principe* : l'entropie combinatoire est le nombre de combinaisons possibles selon un ensemble de règles préalablement définies.
- *Avantage* : elle est indépendante de toute hypothèse probabiliste.
- *Inconvénient* : il est très difficile de construire un ensemble de règles exhaustif, par exemple un texte n'obéit pas aux seules règles grammaticales mais, dépend aussi du genre littéraire, du sujet, ...

Entropie de Shannon :

- *Principe* : l'entropie de Shannon est la quantité moyenne d'information pour chaque caractère.
- *Avantage* : elle est facilement estimée en remplaçant les probabilités de chaque caractère par les fréquences empiriques de ces derniers.
- *Inconvénient* : l'information délivrée dépend de l'ensemble des messages possibles (c'est-à-dire que la mesure d'information n'est pas absolue) et le modèle probabiliste n'a pas toujours un sens pour la chaîne étudiée.

I.1.2 Une nouvelle mesure d'information : la complexité de Kolmogorov

Mesurer l'information à partir de l'entropie ne semble pas toujours adapté. En effet, l'entropie combinatoire n'est pas calculable. L'entropie de Shannon, quant à elle, mesure

l'information de la chaîne par rapport à un ensemble, cette mesure n'est pas absolue. De plus, la théorie probabiliste ne s'applique pas à tous les types de chaînes.

C'est pourquoi Kolmogorov [Kol65] a proposé une nouvelle mesure de l'information contenue dans une chaîne X appelée complexité. Cette mesure ne repose plus sur les probabilités, mais sur l'algorithmique. La complexité a pour but d'être une mesure d'information absolue, c'est-à-dire qu'elle ne dépend pas de l'ensemble auquel appartient la chaîne. L'indépendance de la complexité à tout modèle sur les données lui permet de s'appliquer à tout type de chaîne. Cette quantité est la base de la théorie algorithmique de l'information tout comme l'entropie de Shannon est la base de la théorie probabiliste de l'information.

Les preuves des théorèmes présentés dans la suite de la section I.1 peuvent être trouvées dans [LV08]. Pour un souci de clarté, elles ne seront pas reprises ici.

I.1.2.1 La complexité de Kolmogorov inconditionnelle

Comme nous l'avons vu dans la section I.1.1.2, l'information donnée par l'entropie de Shannon sur une chaîne dépend de l'ensemble des possibilités de cette chaîne. Il semble plus pertinent que l'information sur la chaîne soit absolue et ne dépende pas de l'ensemble des chaînes possibles. Ainsi contrairement à Shannon, le but de Kolmogorov est de définir une information absolue de la chaîne indépendamment de l'ensemble des chaînes possibles. L'intuition semble nous suggérer que certaines chaînes sont compliquées et d'autres sont simples. Reprenons par exemple le nombre 2^{999} et la chaîne binaire associée appelée c_1. Il est assez évident que cette chaîne est très simple. Il est en effet possible de l'écrire de manière simple et efficace : écrire un '1' puis 999 '0'. Cependant, si nous prenons la chaîne c_2 de 1000 bits au hasard, il sera assez difficile de l'écrire autrement qu'avec les 1000 bits les uns à la suite des autres. La description de c_2 prend alors environ 1000 caractères ce qui est bien plus que la description de c_1. Il est donc possible de définir la quantité d'information d'une chaîne par le nombre de bits nécessaires pour la décrire de la manière la plus courte possible.

Notons tout de suite que cette description n'est utile que si on peut reconstruire la chaîne en entier à partir de cette description. En effet cette description n'a un sens que s'il est possible de retrouver la chaîne d'origine à partir de cette dernière. Sinon, une description possible de c_2 serait *chaîne très longue qui ne semble pas vraiment ordonnée*, mais cette description ne permet pas de reconstruire c_2 (même si, en un sens, elle la décrit) et n'apporte donc pas une quantité d'information sur c_2 en particulier.

La complexité de Kolmogorov, fondée sur l'algorithmique, nécessite donc en premier lieu de spécifier les conditions opérationnelles de sa définition : un langage et une machine capable d'accepter ce langage.

Définition I.1.1 (Fonction récursive ou méthode de programmation)**.**
Une fonction récursive ou méthode de programmation ϕ est

- *une fonction des entiers naturels vers les entiers naturels, ($\phi : \mathbb{N} \to \mathbb{N}$)*
- *une fonction partielle, c'est-à-dire qu'elle peut ne pas être définie sur certains entiers,*
- *une fonction qui peut être exécutée par une machine de Turing.*

L'ensemble des fonctions partielles récursives est noté φ.

La méthode de programmation ou fonction récursive peut être vue comme le langage dans lequel nous écrivons notre programme. Par exemple, le langage `C` ou le langage `FORTRAN` sont tous les deux des méthodes de programmation. Ces méthodes de programmation permettent d'exécuter des programmes qui génèrent des chaînes. Par exemple, prenons un programme qui affiche la phrase `Hello world!` . Le programme I.1 est le programme capable de générer la chaîne `Hello world!` en langage `C`. En `FORTRAN`, pour générer la chaîne `Hello world!`, nous pourrions utiliser le programme I.2. Ainsi, ces deux programmes sont les plus petits programmes qui permettent de générer la chaîne `Hello world!` respectivement en `C` et en `FORTRAN`.

```c
#include <stdlib.h>
#include <stdio.h>
int main () {
  printf( "Hello world!" );
  return  EXIT_SUCCESS ;
}
```

Programme I.1 – `Hello world` en `C`

```
Program Hello
Print *, "Hello world!"
End Program Hello
```

Programme I.2 – `Hello world` en `FORTRAN`

Si la méthode de programmation $\phi \in \varphi$ et s'il existe un programme p qui permet de générer la chaîne X par la méthode de programmation ϕ, alors on note $X = \phi(p)$. Notons que le programme p est également une chaîne de $\mathcal{A}^*$. Par exemple si $X =$ `Hello world!`, ϕ_C représente la méthode de programamtion par le langage `C` et p est le programme I.1 alors nous pouvons noter $X = \phi_C(p)$.

Soit l'ensemble des chaînes étudiées $\mathcal{X} \subset \mathcal{A}^*$ et supposons une énumération standard des chaînes X par des entiers naturels $n(X)$ (c'est-à-dire que la première chaîne de $\mathcal{X}$ est la chaîne numéro 0, la deuxième la numéro 1, et ainsi de suite). Il peut exister une méthode plus économique pour spécifier la chaîne X que $n(X)$: c'est cette méthode que nous voulons trouver pour définir la complexité. Pour rappel, on note $l(X)$ la longueur de la chaîne $X \in \mathcal{A}^*$.

Définition I.1.2 (Complexité)**.**
Soit ϕ une fonction récursive partielle, alors la complexité de la chaîne X selon la méthode de programmation ϕ est :

$$K_\phi(X) = \min_{p \in \mathcal{A}^*} \{l(p) | \phi(p) = X\}. \tag{I.4}$$

et $K_\phi(X) = \infty$ s'il n'existe pas de programme p tel que $X = \phi(p)$.

La complexité de X est donc la longueur du plus petit programme p capable de générer X selon la méthode de programmation ϕ.

Reprenons l'exemple des deux chaînes, c_1 qui représente le nombre 2^{999} en binaire et c_2 composée de 1000 bits indépendants pris au hasard. Le pseudo-code pour générer c_1 est donné par l'algorithme 1.

Algorithme 1 Générer $c_1 = 2^{1000}$

```
écrire 1
écrire 999 fois 0
```

Ce programme est très simple et sa longueur assez faible, qu'il soit écrit en `FORTRAN`, en `LISP`, en `C` ou avec n'importe quelle méthode de programmation ϕ. Mais le plus petit programme pour générer c_2 consiste en la réécriture des 1000 bits qui le composent. Ce programme est bien plus long que le programme pour générer c_1. On peut donc dire que c_1 est moins complexe que c_2 selon la définition I.1.2.

À ce stade, le problème est que, selon la méthode de programmation utilisée, la mesure de complexité n'est pas la même. Par exemple, les deux programmes I.2 et I.1 ne font pas la même taille, mais ils sont les plus petits programmes capables de générer `Hello world!` respectivement en `FORTRAN` et en `C`. Nous allons maintenant chercher à nous abstraire de la méthode de programmation utilisée. Pour cela il convient de définir la notion de méthode de programmation additivement optimale.

Définition I.1.3 (Fonction additivement optimale, *voir Def 2.0.1 dans [LV08]*)**.**
Soit $f \in \varphi$, f est additivement optimale si $\forall g \in \varphi$, il y a une constante $c_{f,g}$ indépendante de X telle que $\forall X \in \mathcal{A}^* \;\; K_f(X) \leq K_g(X) + c_{f,g}$

Le théorème d'invariance établit l'existence d'une telle fonction.

Théorème I.1.1 (Théorème d'invariance, *voir Th 2.1.1 dans [LV08]*)**.**
Il existe toujours une fonction récursive partielle additivement optimale dans le sens de la définition I.1.3 notée ϕ_0.

Avec les notions que nous venons d'introduire, il est possible de définir la complexité de Kolmogorov.

Définition I.1.4 (Complexité de Kolmogorov)**.**
Soit ϕ_0 une fonction récursive partielle additivement optimale (qui existe au vu du théorème I.1.1), alors la complexité de Kolmogorov d'une chaîne X est définie par :

$$K(X) = K_{\phi_0}(X). \tag{I.5}$$

La complexité de Kolmogorov de X est la longueur du plus petit programme p capable de générer X avec la méthode de programmation ϕ_0 additivement optimale. La signification de la complexité de Kolmogorov reste la même que celle de la complexité : c_1 est moins complexe que c_2. La différence réside dans le fait que dans le cas de la complexité de Kolmogorov la méthode de programmation utilisée est optimale.

Théorème I.1.2 (Borne supérieure, *voir Th 2.1.2 dans [LV08]*)**.**
Il existe une constante c indépendante de X telle que pour tout X,

$$\forall X \in \mathcal{A}^* \;\; K(X) \leq l(X) + c.$$

En effet, pour expliquer X dans le pire des cas il suffit de réécrire ce dernier. La constante c est due au fait que le programme peut être vu comme étant composé de deux parties. La première est la description de la méthode de programmation ϕ_0 et la deuxième est le programme en lui-même qui grâce à ϕ_0 génère X. La constante c vient alors de la première partie du programme.

I.1.2.2 La complexité de Kolmogorov conditionnelle

Nous observons rarement une chaîne seule et nous souhaitons souvent la mettre en relation avec une autre, typiquement pour déterminer la quantité d'information apportée par l'une à l'autre. Nous avons donc besoin de définir un outil permettant une telle mise en relation de deux chaînes : la complexité conditionnelle.

Considérons la fonction $< \cdot, \cdot >: \mathcal{A}^* \times \mathcal{A}^* \to \mathcal{A}^*$ qui est une fonction bijective et récursive qui associe la paire (X, Y) au singleton $< X, Y >$. Il est alors possible de définir la complexité de Kolmogorov conditionnelle de la façon suivante.

Définition I.1.5 (Complexité conditionnelle)**.**
Soit ϕ une fonction partielle récursive, alors la complexité conditionnelle de X sachant Y selon la méthode de programmation ϕ est définie par :

$$K_\phi(X|Y) = min\{l(p) : \phi(< Y, p >) = X\}. \tag{I.6}$$

La complexité de Kolmogorov de X sachant Y est la longueur du plus petit programme p capable de générer X selon la méthode de programmation ϕ avec en entrée Y.

Pour comprendre la signification de la complexité conditionnelle, reprenons nos deux chaînes c_1 et c_2 et regardons la complexité de c_1 et c_2 connaissant c_2. L'algorithme pour générer c_1 avec la connaissance de c_2 ne va pas changer, ça sera toujours l'algorithme 1. En effet, nous pouvons essayer de faire des références à c_2 pour expliquer c_1 comme des opérations d'addition, des copier-collers, etc. Cependant, c_2 n'apporte aucune information

à c_1 : la description de ces opérations rend le programme résultant nécessairement plus long que l'algorithme 1. Par contre pour générer c_2 avec la connaissance de c_2, nous utilisons trivialement l'algorithme 2.

Algorithme 2 Générer c_2, une chaîne aléatoire de 1000 bits, connaissant c_2

copier c_2

On peut donc dire que c_1 est plus complexe en connaissant c_2 que c_2 ne l'est en connaissant c_2. Pour définir la complexité de Kolmogorov conditionnelle, nous utilisons le théorème I.1.1.

Définition I.1.6 (Complexité de Kolmogorov conditionnelle)**.**
Soit ϕ_0 une fonction récursive partielle additivement optimale, alors la complexité de Kolmogorov conditionnelle de $X \in \mathcal{A}^*$ sachant $Y \in \mathcal{A}^*$ est définie par :

$$K(X|Y) = K_{\phi_0}(X|Y)$$

La complexité de Kolmogorov conditionnelle est la longueur du plus petit programme p capable de générer X selon la méthode de programmation ϕ_0 additivement optimale avec en entrée Y.

Il est possible de faire un lien entre la complexité conditionnelle et la complexité non conditionnelle comme suit.

Théorème I.1.3 (Conditionnelle et inconditionnelle, *voir Def 2.1.2 de [LV08]*)**.**
Soit Λ la chaîne vide et $X \in \mathcal{A}^*$,

$$K(X) = K(X|\Lambda).$$

En effet, la différence entre les définitions I.1.4 et I.1.6 est que, dans la seconde, le programme prend en entrée Y. Si, dans cette définition I.1.6, l'entrée Y est vide, c'est-à-dire $Y = \Lambda$ alors nous n'apportons aucune information a priori de X et cela revient à encoder X en ne connaissant que lui-même comme dans la définition I.1.4.

Théorème I.1.4 (Borne supérieure, *voir Th 2.1.2 dans [LV08]*)**.**
Il existe une constante c indépendante de X et Y telle que pour tout $X \in \mathcal{A}^*$ et pour tout $Y \in \mathcal{A}^*$,

$$K(X|Y) \leq K(X) + c$$

Pour expliquer X connaissant Y, nous ajoutons comme a priori Y. Dans le pire des cas Y n'apporte aucune information à X et cela revient à expliquer X connaissant Λ. Or nous savons grâce au lemme I.1.3, que $K(X|\Lambda) = K(X)$. Donc on a bien $K(X|Y) \leq K(X) + c$.

En résumé

Kolmogorov [Kol65] définit une nouvelle mesure de la quantité d'information contenue dans une chaîne X : la complexité de Kolmogorov.

- *Principe* : la complexité de Kolmogorov est la taille du plus petit programme capable de générer la chaîne X.
- *Avantage* : elle est indépendante de toute hypothèse probabiliste et est une mesure absolue de l'information (c'est-à-dire qu'elle ne dépend pas de l'ensemble auquel la chaîne appartient).
- *Inconvénient* : l'évaluation de la complexité de Kolmogorov en un temps fini n'est pas garantie [CT12].

I.1.3 La théorie algorithmique de l'information

Comme nous l'avons vu précédemment, la complexité de Kolmogorov est une mesure absolue de l'information contenue dans X, c'est-à-dire qu'elle ne dépend pas de l'ensemble auquel appartient la chaîne X. Bien qu'il soit possible d'écrire la théorie algorithmique de l'information basée sur la complexité de Kolmogorov, tout comme la théorie probabiliste de l'information basée sur l'entropie de Shannon, nous montrons que la complexité de Kolmogorov et l'entropie de Shannon sont équivalentes pour des chaînes très grandes.

I.1.3.1 Lien entre la complexité de Kolmogorov et l'entropie de Shannon

L'entropie de Shannon, $H(X)$ vue dans la section I.1.1.2, est le nombre moyen de bits nécessaire pour décrire X. La complexité de Kolmogorov, $K(X)$ vue dans la section I.1.2.1, est la longueur minimale d'un programme binaire pour générer X. Les deux définitions sont donc relativement proches dans le but qu'elles se proposent et il est possible de les mettre en relation.

Soit une variable aléatoire $\mathbf{X}$ qui prend ses valeurs dans $\mathcal{A}^n$, la probabilité que $\mathbf{X}$ prenne la valeur X est notée $P(\mathbf{X} = X)$. Alors l'entropie des chaînes de longueur n est :

$$H = -\sum_{X \in \mathcal{A}^n} P(\mathbf{X} = X) log(P(\mathbf{X} = X)).$$

Pour pouvoir relier l'entropie de Shannon et la complexité de Kolmogorov, il faut introduire des probabilités dans la complexité de Kolmogorov. Pour cela nous définissons la complexité en espérance comme étant :

$$E = \sum_{X \in \mathcal{A}^n} P(\mathbf{X} = X) K(X).$$

Alors quand $n \to \infty$, l'entropie et la complexité en espérance sont équivalentes.

Théorème I.1.5 (Lien entre l'entropie et la complexité, *voir Th 2.1.2 dans [LV08]*).
Soient P une distribution de probabilité et $\mathbf{X}$ une variable aléatoire qui prend ses valeurs dans $\mathcal{A}^n$, alors

$$\sum_{X\in\mathcal{A}^n} P(\mathbf{X}=X)K(X) \underset{n\to\infty}{\sim} \sum_{X\in\mathcal{A}^n} P(\mathbf{X}=X)log\frac{1}{P(\mathbf{X}=X)} \iff \lim_{n\to\infty}\frac{H}{E}=1.$$

L'interprétation de la complexité comme la mesure de l'information contenue dans une chaîne individuelle X est étayée par la relation quantitative étroite avec la notion probabiliste de Shannon. Il convient maintenant de déterminer dans quelles mesures des relations entre quantités d'information, similaires à la théorie probabiliste de l'information, existent.

I.1.3.2 La théorie algorithmique de l'information

$K(X)$ est la quantité d'information nécessaire pour générer la chaîne X à partir de rien. De même, $K(X|Y)$ est la quantité d'information nécessaire pour générer la chaîne X à partir de Y. Ainsi, si $K(X|Y)$ est beaucoup plus petite que $K(X)$ alors cela veut dire que Y contient beaucoup d'information sur X. Il est donc possible de définir l'information sur X contenue dans Y.

Définition I.1.7 (Information mutuelle algorithmique).
L'information mutuelle algorithmique sur X contenue dans Y est définie comme

$$I_K(Y:X) = K(X) - K(X|Y). \tag{I.7}$$

L'information mutuelle algorithmique sur X contenue dans Y mesure la simplification qu'apporte Y pour générer X. Elle permet de mesurer l'indépendance algorithmique entre deux chaînes de caractères X et Y.

Cette information mutuelle algorithmique vérifie deux des propriétés de l'information mutuelle de Shannon. Grâce au théorème I.1.4, nous savons que $K(X|Y) \leq K(X)$ à une constante près. Donc, on a bien $I_K(y:x) \geq 0$ à une constante près. De plus $K(X|X) = 0$, donc $I_K(X:X) = K(X)$ à une constante près. Cependant, elle ne vérifie pas la symétrie. Pour quantifier l'asymétrie de l'information algorithmique, il faut définir la complexité jointe.

Définition I.1.8 (Complexité de Kolmogorov jointe).
La complexité de Kolmogorov jointe de $X \in \mathcal{A}^*$ et $Y \in \mathcal{A}^*$

$$K(X,Y) = K(< X,Y >) \tag{I.8}$$

La complexité de Kolmogorov jointe est la longueur du plus petit programme qui est capable de générer X et Y et de les séparer En effet, le programme p va générer en sortie une chaîne Z dans $\mathcal{A}^*$. Cependant, comme nous l'avons vu dans la section I.1.2.1, il est indispensable de pouvoir reconstruire X et Y séparément. Il faut donc être en mesure de savoir quels caractères de Z viennent de X et lesquels viennent de Y.

L'entropie de Shannon jointe $H(X,Y)$ est égale à $H(X) + H(Y|X)$ mais pour la complexité de Kolmogorov, cette propriété reste valide à une constante près.

Théorème I.1.6 (Asymétrie de la complexité jointe, *voir Th 2.8.2 dans [LV08]*)**.**
Pour tout X et Y dans $\mathcal{A}^*$

$$K(X,Y) = K(X) + K(Y|X) + \mathcal{O}(logK(X,Y)) \tag{I.9}$$

Ce théorème signifie qu'il est presque équivalent d'encoder simultanément X et Y ou d'encoder X puis Y connaissant X.

Il est donc possible d'écrire :

$$I_K(Y:X) = K(X) + K(Y) - K(Y,X) + \mathcal{O}(logK(X,Y)).$$

L'asymétrie est donc

$$|I_K(X:Y) - I_K(Y:X)| = \mathcal{O}(logK(X,Y)).$$

Ainsi avec la complexité de Kolmogorov l'information mutuelle algorithmique de X contenue dans Y n'est pas strictement égale à celle de Y contenue dans X. L'ordre dans lequel X et Y sont donnés dans l'information mutuelle est donc très important.

En résumé

L'entropie de Shannon et la complexité de Kolmogorov sont équivalentes lorsque la taille de la chaîne est très grande. Il est ainsi possible de définir la théorie algorithmique de l'information basée sur la complexité de Kolmogorov tout comme la théorie probabiliste de l'information l'est avec l'entropie de Shannon.

Complexité de Kolmogorov conditionnelle : la taille du plus petit programme capable de générer la chaîne X en connaissant Y.

Complexité de Kolmogorov jointe :

- *Principe* : c'est la taille du plus petit programme capable de générer les chaînes X et Y et de les séparer.
- *Inconvénient* : l'égalité $K(X,Y) = K(X) + K(Y|X) + \mathcal{O}(logK(X,Y))$ n'est valable qu'à une constante près.

Information mutuelle algorithmique :

- *Principe* : c'est l'indépendance algorithmique des deux chaînes de caractères Y et X.
- *Inconvénient* : la symétrie n'est valable qu'à une constante près $\mathcal{O}(logK(X,Y))$.

I.2 Complexité de Lempel-Ziv

Pour Lempel et Ziv, la mesure d'une complexité absolue telle que définie par Kolmogorov n'existe pas, puisqu'il est impossible de garantir de pouvoir trouver le programme le plus court en un temps fini. Ainsi, ils proposent une nouvelle méthode pour mesurer la complexité d'une séquence finie [LZ76]. Cette méthode est toujours basée sur l'algorithmique et ne fait pas appel aux probabilités. De plus la complexité de Lempel-Ziv est une mesure d'information absolue, car elle ne dépend pas de l'ensemble auquel appartient la chaîne. Nous donnerons aussi une définition mathématique de ce qu'est une mesure d'information dans un sens tout à fait général. Nous verrons alors que la complexité de Lempel-Ziv peut être la base d'une théorie algorithmique de l'information, mais cette fois pleinement opérationnelle, car calculable en un temps fini.

I.2.1 Définition de la complexité de Lempel-Ziv

Lempel et Ziv proposent une nouvelle complexité dans [LZ76]. Cette complexité n'est plus basée sur un programme comme la complexité de Kolmogorov, mais sur un processus de production de la chaîne. Ce processus de production permet de créer la chaîne à partir de son passé grâce à des opérations élémentaires comme `copier` et `insérer`. L'avantage d'une telle complexité est qu'elle est calculable en temps fini, du fait de la restriction du programme à ces deux opérations élémentaires.

I.2.1.1 Notations

Dans un premier temps, il est nécessaire de poser quelques définitions pour bien comprendre comment est calculée la complexité de Lempel-Ziv. On rappelle que pour une chaîne $X = x_1x_2...x_n \in \mathcal{A}^*$, $X(i,j) = x_i...x_j$ est la sous chaîne de X composée de $j-i+1$ caractères compris entre l'indice i et j.

Définition I.2.1 (Préfixe)**.**
Q est appelé un préfixe de X dans $\mathcal{A}^*$ et X est appelée une extension de Q s'il existe un entier i tel que $Q = X(1,i)$. Un préfixe Q et son extension X sont dits propres si $l(Q) < l(X)$.

Lorsque la longueur d'une chaîne X n'est pas entièrement spécifiée, il est commode d'identifier les préfixes de X au moyen d'un opérateur spécial π tel que $X\pi^i = X(1, l(X) - i)$. On a alors $X\pi^0 = X$, $X\pi^1 = X\pi = X(1, l(X)-1)$ (c'est-à-dire la supression du dernier caractère) et si $i \geq l(X)$, $X\pi^i = \Lambda$ (la chaîne vide).

Définition I.2.2 (Vocabulaire)**.**
Le vocabulaire d'une chaîne X, noté $v(X)$, est un sous-ensemble de $\mathcal{A}^*$ formé par toutes les sous-chaînes de X

Par exemple, le vocabulaire de la chaîne 0010 est donné par :

$$v(0010) = \{\Lambda, 0, 1, 00, 01, 10, 001, 010, 0010\}.$$

Définition I.2.3 (Reproduction)**.**
Une chaîne non nulle X est reproductible à partir de son préfixe $X(1,j)$ si :

i $X(j+1, l(X)) \in v(X\pi)$,

ii $j < l(X)$.

On note alors $X(1,j) \to X$.
Le contraire de la reproduction, c'est-à-dire que X n'est pas reproductible à partir de $X(1,j)$, est noté $X(1,j) \not\to X$.

Si X est reproductible à partir de son préfixe $X(1,j)$, cela implique qu'il est possible de créer X à partir de $X(1,j)$. En effet $X(j+1, l(X))$ est sous-mot de $X\pi$, il est donc possible de réaliser une opération `copier` qui commence dans $X(1,j)$ pour créer $X(j+1, l(X))$. Donc il existe une position $t \leq j$ dans X telle que $X(j+1, l(X)) = X(t, l(X) - j + t - 1)$. Cette position est appelée pointeur pour la reproduction $X(1,j) \to X$.

Prenons par exemple la chaîne $X = 00101010$. Le vocabulaire de $X\pi$ est

$$v(X\pi) = \{\Lambda, 0, 1, 00, 01, 10, 001, 010, 101, 0010, 0101, 1010,\\ 00101, 01010, 10101, 001010, 010101, 0010101\}.$$

On peut donc prendre $j = 3$ et dire que $X(1,j) = X(1,3) = 001 \to X = 00101010$ car $X(j+1, l(X)) = X(4,8) = 01010 \in v(X\pi)$. Le pointeur de reproduction est $t = 2$ car $X(4,8) = 01010 = X(2,6)$.

Définition I.2.4 (Production)**.**
Une chaîne non nulle $X \in \mathcal{A}^*$ est dite productible à partir de son préfixe $X(1,j)$ si

i $X(1,j) \to X\pi$,

ii $j < l(X)$.

On note alors $X(1,j) \Rightarrow X$ et on dit que $X(1,j)$ est une base de X.

Notons que X, une chaîne non nulle, a toujours une base (par exemple $X\pi$) et que Λ est une base pour tous les caractères $a \in \mathcal{A}$ mais pour aucune autre chaîne dans $\mathcal{A}^*$. De plus si X est productible à partir de son préfixe $X(1,j)$ alors elle est reproductible à partir de ce même préfixe.

La différence entre la reproduction et la production est que dans la production le dernier caractère peut être nouveau. En effet, la condition de productibilité $X(1,j) \to X\pi$ équivaut à $X(j+1, l(X)-1) \in v(X\pi^2)$. Pour bien comprendre cette différence, reprenons l'exemple si dessus et changeons le dernier caractère : $X = 00101011$. Nous avons alors pour $j = 3$, $X(1,j) = X(1,3) = 001 \not\to X = 00101010$. Par contre, $X(1,j) = X(1,3) = 001 \Rightarrow X = 00101010$. En effet, le vocabulaire de $X\pi^2$ est

$$v(X\pi^2) = \{\Lambda, 0, 1, 00, 01, 10, 001, 010, 101, 0010, 0101, 1010, 00101, 01010, 001010\}.$$

Nous avons bien $X(j+1, l(X)-1) = X(4,7) = 0101 \in v(X\pi^2)$. Il est possible de produire X grâce à $X(1,3)$ en réalisant une opération `copier` puis une opération `insérer` pour le dernier caractère.

I.2.1.2 Définitions

Il est possible d'interpréter l'opération *production* comme un mécanisme de génération de X basé sur les deux opérations `copier` et `insérer` à partir de l'un de ces préfixes. Or nous savons que toute chaîne non nulle X peut être considérée comme une production à partir d'un préfixe propre.

Définition I.2.5 (Processus de production)**.**
Un processus de production est un processus étape par étape qui génère X grâce à des opérations de production à partir de ses préfixes.

Ce processus existe dans tous les cas, et découpe lors de ses m étapes la chaîne en m sous-chaînes. La coupure entre la sous-chaîne numéro i et la sous-chaîne $i+1$ est réalisée au caractère numéro d_i de X.

Reprenons l'exemple de notre chaîne $X = 00101010$. La première étape est dans tous les cas d'exécuter l'opération de production $\Lambda = X(1,0) \Rightarrow X(1,1) = 0$, on a donc $d_1 = 1$. La deuxième étape peut être $X(1,1) = 0 \Rightarrow X(1,2) = 00$, on a donc $d_2 = 2$. La troisième étape peut être $X(1,2) = 00 \Rightarrow X(1,3) = 001$, on a donc $d_3 = 3$. Puis la quatrième et dernière étape peut être $X(1,3) = 001 \Rightarrow X(1,8) = 00101010$, on a donc $d_4 = l(X) = 8$. Ce processus de production a donc généré X en 4 étapes.

Cependant ce processus de production n'est pas unique, par exemple lors de la deuxième étape $X(1,1) = 0$ est bien une base de $X(1,2) = 00$ mais aussi de $X(1,3) = 001$, il est donc possible de poser $d_2 = 3$. Ce deuxième processus de production a donc généré X en 3 étapes. On montre facilement que le processus de production contient au plus $l(X)$ étapes et que dans tous les cas $d_m = l(X)$.

Définition I.2.6 (Historique de production)**.**
L'historique de production de X est $D(X) = X(1,d_1)X(d_1+1,d_2)...X(d_{m-1}+1,d_m)$
Les m mots $D_i(X) = X(d_{i-1}+1,d_i)$ pour $i = 0,1,...,m$ sont appelés les composants de $D(X)$. On note $c_D(X) = m$, le nombre de composants de D et $\mathcal{D}(X)$ l'ensemble des historiques de production qui génère X.

Lempel et Ziv considèrent que le nombre d'étapes nécessaires dans le processus de production d'une chaîne X est lié à sa complexité. De la même manière que Kolmogorov s'intéresse au plus petit programme, Lempel et Ziv s'intéressent au processus de production avec le plus petit nombre d'étapes.

Définition I.2.7 (Complexité de Lempel-Ziv)**.**
Soit $X \in \mathcal{A}^*$, alors la complexité de Lempel-Ziv de X est définie comme :

$$L(X) = \min_{D \in \mathcal{D}(X)} c_D(X). \tag{I.10}$$

La complexité de Lempel-Ziv de X, notée $L(X)$, est le nombre minimal d'étapes de production nécessaires pour générer X à partir de ses préfixes.

Contrairement à la complexité de Kolmogorov, la complexité de Lempel-Ziv peut être calculée en temps fini.

Définition I.2.8 (Exhaustivité)**.**
Un composant $D_i(X)$ et l'étape de production $X(1, d_{i-1}) \Rightarrow X(1, d_i)$ sont exhaustifs si

$$X(1, d_{i-1}) \not\to X(1, d_i).$$

Pour comprendre ce qu'est une étape de production exhaustive, plaçons nous à l'étape i. Supposons qu'il existe une sous-chaîne Q de longueur l dans $X(1, d_{i-1})$ telle que $Q = X(d_{i-1} + 1, d_{i-1} + l)$, mais qu'il n'existe aucune sous-chaîne de longueur supérieure à l identique à $X(d_{i-1} + 1, d_{i-1} + l + 1)$(ainsi Q est la plus grande sous-chaîne possible). On peut alors poser $d_i = d_{i-1} + l$, on a alors $X(1, d_{i-1}) \Rightarrow X(1, d_i)$ et $X(1, d_{i-1}) \to X(1, d_i)$. Dans ce cas-là, l'étape de production n'est pas exhaustive. Cependant si nous posons, $d_i = d_{i-1} + l + 1$ on a alors $X(1, d_{i-1}) \Rightarrow X(1, d_i)$ et $X(1, d_{i-1}) \not\to X(1, d_i)$ car il n'existe aucune sous-chaîne de longueur $l+1$. Donc si $X(1, d_{i-1}) \Rightarrow X(1, d_i)$ et $X(1, d_{i-1}) \not\to X(1, d_i)$, cela signifie que l'on a conservé la sous-chaîne la plus longue possible grâce à une opération `copier` et que l'on a ajouté un élément innovant à la fin grâce à une opération `insérer`.

Définition I.2.9 (Historique exhaustif)**.**
Un historique $D(X)$ est exhaustif si chacun de ses composants est exhaustif, avec une exception possible pour le dernier. Cet historique est noté $E(X)$.

Ainsi si toutes les étapes de productions sont exhaustives, alors à chaque étape le processus de production cherche la plus grande sous-chaîne dans son passé et insère un élément innovant à la fin. Un processus de production exhaustif est donc composé à chaque étape de deux opérations : un `copier` et un `insérer`. Il est assez facile de vérifier que chaque chaîne X non nulle possède un historique exhaustif unique. Par exemple l'historique exhaustif de $X = 0001101001000101$ est :

$$0 \cdot 001 \cdot 10 \cdot 100 \cdot 1000 \cdot 101$$

Les composants successifs sont séparés par des points et l'absence de point à la fin de la chaîne signifie que le dernier composant n'est pas exhaustif.

Théorème I.2.1 (Complexité de Lempel-Ziv)**.**
Soit $X \in \mathcal{A}^*$ et E l'historique exhaustif de X, alors la complexité de Lempel-Ziv est définie par :

$$L(X) = c_E(X).$$

La complexité de Lempel-Ziv est le nombre d'étapes exhaustives nécessaires pour générer X. C'est-à-dire que la complexité de Lempel-Ziv est le nombre minimal de couples d'opérations (`copier`,`insérer`) capable de générer X à partir de son passé. La démonstration de ce théorème peut être trouvée dans [LZ76].

I.2.1.3 Propriétés

Dans cette partie, nous donnons quelques propriétés de la complexité de Lempel-Ziv. Tout d'abord il est possible de trouver une borne supérieure autre que la longueur de la chaîne X, $l(X)$, pour la complexité de Lempel-Ziv.

Théorème I.2.2 (Borne supérieure)**.**
Pour tout $X \in \mathcal{A}^n$

$$L(X) < \frac{n}{(1-\epsilon_n)log_\alpha(n)}$$

où

$$\epsilon_n = 2\frac{1+log_\alpha log_\alpha(\alpha n)}{log_\alpha(n)}$$

La preuve repose sur le nombre N maximal de mots distincts possibles dans une chaîne de longueur n sur un alphabet de taille α. En effet on alors $L(X) \leq N$. Pour la preuve complète, voir [LZ76].

Tout comme la complexité de Kolmogorov, il est possible de lier la complexité de Lempel-Ziv à l'entropie de Shannon (normalisée).

Théorème I.2.3 (complexité de Lempel-Ziv et entropie de Shannon)**.**
On pose $H_{norm} = H(X)/log_\alpha(l(X))$ *l'entropie de Shannon normalisée. Alors* $\forall X \in \mathcal{A}^n$ *:*

$$\lim_{n\to\infty} L(X)\frac{log_\alpha(n)}{n} = H_{norm}(X)$$

Pour terminer, le processus de production n'est pas restreint aux simples opérations `copier` et `insérer`. Il est en effet possible de prendre une application arbitraire de f : $\mathcal{A}^m \to \mathcal{A}$, où m constant. La relation de $f-$reproduction généralisée à f, $X \xrightarrow{f} R = XQ$, signifie qu'il existe un pointeur p tel que $1 \leq p \leq l(X) - m + 1$ et que pour $i = 1, 2, ..., l(Q)$ on ait $q_i = f(R(p+i-1, p+i+m-2))$. De même, il est possible de généraliser les définitions de $f-$production et de $f-$exhaustivité. Le théorème I.2.1 reste vrai pour toutes les applications f comme l'ont montré Lempel et Ziv [LZ76]. Cependant les théorèmes I.2.2 et I.2.3 qui sont les bornes de la complexité de Lempel-Ziv ne seront pas les mêmes. Si nous autorisions toutes les opérations possibles alors nous tombons dans le même travers que la complexité de Kolmogorov : la complexité de Lempel-Ziv ne serait plus calculable en un temps fini. Lempel et Ziv ont donc restreint leur complexité à ces deux opérations, car elles couvrent à elles seules une grande partie des cas.

En résumé

Dans [LZ76], Lempel et Ziv proposent une nouvelle mesure de complexité : la complexité de Lempel-Ziv.

- *Principe* : c'est le nombre minimal d'opérations `copier` suivi de l'opération `insérer` qui permet de générer la chaîne X grâce à son passé.
- *Liens* :
  - Avec la complexité de Kolmogorov : la complexité de Lempel-Ziv limite le programme aux deux opérations `copier` et `insérer` ([ZRB05]).
  - Avec l'entropie de Shannon : elles sont équivalentes quand la chaîne est très grande à un facteur $\frac{l(X)}{log_\alpha\, l(X)}$ près.
- *Avantage* : la complexité de Lempel-Ziv est calculable dans un temps fini contrairement à la complexité de Kolmogorov et il est possible d'insérer d'autres opérations que `copier` et `insérer`.

I.2.2 Mesure d'information

Nous avons montré que la complexité de Lempel-Ziv était reliée à la l'entropie de Shannon, que la complexité de Kolmogorov était aussi équivalente à l'entropie de Shannon et que la complexité de Lempel-Ziv peut être vue comme la complexité de Kolmogorov restreinte aux opérations `copier` et `insérer`. Il se trouve que l'entropie de Shannon, la complexité de Kolmogorov et la complexité de Lempel-Ziv sont toutes les trois des mesures d'information. Nous définissons dans cette section la notion de mesure d'information en un sens général, avec pour conséquence que toute mesure d'information devient alors la base d'une théorie de l'information.

Nous travaillons dans cette section sur un réseau fini $(\Omega, \wedge, \vee)$, où Ω est un ensemble, par exemple $\mathcal{A}^*$ dans le cas de la complexité de Lempel-Ziv. $\wedge$ est l'opération union et $\vee$ est l'opération intersection entre les éléments de Ω. Il existe une relation d'ordre sur Ω notée $\leq$. Nous notons 0 l'élément nul du réseau qui est l'intersection de tous les éléments de Ω, par exemple dans $\mathcal{A}^*$ c'est la chaîne vide Λ.

Définition I.2.10 (Mesure d'information, voir [SJS10])**.**
Nous disons que $R : \Omega \to \mathbb{R}$ est une mesure d'information, si elle respecte les trois propriétés suivantes :

1. *normalisation :* $R(0) = 0$,
2. *monotonie :* $\forall s, t \in \Omega, s \leq t \Rightarrow R(s) \leq R(t)$,
3. *sous-modularité :* $\forall s, t \in \Omega, R(s) + R(t) \geq R(s \wedge t)$.

La mesure d'information conditionnelle est alors définie comme :

$$\forall s, t \in \Omega, R(s|t) = R(s \vee t) - R(t).$$

L'information mutuelle basée sur la mesure d'information R est donnée par :

$$\forall s, t \in \Omega, I_R(s,t) = R(s) + R(t) - R(s \vee t). \tag{I.11}$$

Dans [SJS10], les auteurs ont montré que l'entropie de Shannon, la complexité de Kolmogorov et la complexité de Lempel-Ziv (au sens de la Section I.2.3.3) sont toutes les trois des mesures d'information. Il en existe cependant bien d'autres comme la longueur de la période d'une série temporelle ou la taille du vocabulaire dans un texte.

En résumé

Dans leur article [SJS10], Steudel et al. proposent une définition de la *mesure d'information*.

- *Principe* : toute mesure d'information qui respecte la définition I.2.10 constitue la base d'une théorie de l'information.
- L'entropie de Shannon, la complexité de Kolmogorov et la complexité de Lempel-Ziv sont toutes les trois des mesures d'information.

I.2.3 Complexité conditionnelle et jointe

La complexité de Lempel-Ziv est une mesure d'information, il est donc possible de définir une complexité jointe et conditionnelle associées. Ces définitions permettent d'étendre la complexité de Lempel-Ziv à la théorie algorithmique de l'information. Cependant, Lempel et Ziv n'ont pas défini en 1976 la complexité de Lempel-Ziv jointe ou conditionnelle. Ainsi différents auteurs ont proposé leur propre définition, souvent attachée à une application particulière, sans viser d'unification formelle entre ces quantités.

I.2.3.1 Approche basée sur une divergence (1993)

Les premiers à définir une complexité conditionnelle sont Ziv et Merhav en 1993 [ZM93]. Ils cherchent à définir une divergence entre deux chaînes X et Y basée sur la complexité de Lempel-Ziv dont l'objectif est de réaliser une classification universelle. Ils ont alors besoin de définir une complexité conditionnelle. Pour eux, la complexité de X sachant Y implique que le processus de production ne peut générer X qu'à partir de Y. Lors du processus de production exhaustif, à chaque étape nous ne cherchons plus le plus grand `copier` dans le vocabulaire de X mais dans celui de Y et uniquement Y. Cette définition de la complexité conditionnelle est appelée complexité de Ziv-Merhav et est notée $L_Z(X|Y)$.

Par exemple si $X = 01111000110$ et si $Y = 10010100110$, alors le processus de production exhaustif de X connaissant Y (et uniquement Y) est

$$011 \cdot 110 \cdot 00110.$$

Ainsi la complexité de Lempel-Ziv conditionnelle selon Ziv et Merhav vaut $L_Z(X|Y) = 3$.

Maintenant que la complexité conditionnelle a une définition, essayons de définir la complexité jointe. Pour rappel, la complexité de Kolmogorov jointe de X et Y est la taille du plus petit programme capable de générer X et Y et de les séparer. Ainsi, il est possible de définir la complexité de Lempel-Ziv jointe comme étant le plus court processus de production capable de générer X et Y.

Comme nous l'avons vu pour la complexité de Kolmogorov, il est possible d'écrire à une constante près $K(X,Y) = K(X) + K(Y|X)$ (équation I.9). Cette formule est vraie pour toute mesure d'information et donc pour la complexité de Lempel-Ziv. Cependant avec cette définition $L(X) + L_Z(Y|X)$ n'est pas le plus petit processus de production capable de générer X et Y. En effet, si par exemple $\mathcal{A}_X \cap \mathcal{A}_Y = \emptyset$, c'est-à-dire qu'aucun caractère n'est commun aux deux alphabets de X et Y, alors il ne sera possible de faire que des opérations `insérer` lors du processus de production de Y sachant X. On aurait donc $L_Z(Y|X) = l(Y)$ ce qui n'est manifestement pas le processus de production avec le moins d'étapes : il faudrait permettre de faire des opérations `copier` également dans Y ce que nous détaillons plus loin.

I.2.3.2 Approche multidimensionnelle (2005)

Dans [ZRB05], Zozor et al. étendent la complexité de Lempel-Ziv au cas multidimensionnel. Pour cela, considérons k chaînes de taille n, $X_1 \in \mathcal{A}^*{}_1, X_2 \in \mathcal{A}^*{}_2, ..., X_k \in \mathcal{A}^*{}_k$. Le $i^{\text{ème}}$ caractère de la $j^{\text{ème}}$ chaîne X_j est notée $x_{\{j,i\}}$. Il est alors possible de créer $Z = z_1...z_n$ une chaîne de n $k-$uplets définie sur l'alphabet $\mathcal{A}^Z = \mathcal{A}_1 \times ... \times \mathcal{A}_k$. Le $i^{\text{ème}}$ caractère de Z est le k-uplet $z_i = (x_{\{1,i\}}, x_{\{2,i\}}, ..., x_{\{j,i\}}, ..., x_{\{k,i\}})$, c'est-à-dire les $i^{\text{ème}}$ caractères des k chaînes. Prenons un exemple pour illustrer cette transformation : $k = 2$, $X_1 = 10010101$ et $X_2 = 11010001$. Alors nous avons

$$Z = \binom{1}{1}\binom{0}{1}\binom{0}{0}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{0}{0}\binom{1}{1}.$$

Dans ce cadre, Zozor et al ont montré que les définitions de reproduction et de production définies par Lempel et Ziv (définition I.2.3 et définition I.2.4) restent valides pour les $k-$uplets. Cela permet d'étendre le travail de Lempel et Ziv aux des chaînes vectorielles (en particulier le théorème I.2.3). La complexité de Lempel-Ziv de $X_1, ..., X_k$, appelée complexité de Lempel-Ziv multidimensionnelle est alors la complexité de Z : $L_M(X_1, ..., X_k) = L(Z)$.

Pour définir la théorie algorithmique de l'information basée sur la complexité de Lempel-Ziv, il faut se placer dans le cas où $k = 2$. La complexité de Lempel-Ziv multidimensionnelle dans le cas $k = 2$, $L_M(X_1, X_2)$, est la complexité de Lempel-Ziv jointe. Cette définition repose sur l'interprétation de la fonction bijective dans l'équation I.8 comme étant une transformation en $2-$uplets. Cette complexité de Lempel-Ziv jointe respecte les deux propriétés suivantes :

Théorème I.2.4.
Soit X et $Y \in \mathcal{A}^*$

i Cette complexité jointe est symétrique

$$L_M(X,Y) = L_M(Y,X)$$

ii et dans le cas particulier où $X = Y$

$$L_M(X,X) = L(X)$$

De façon analogue à la complexité de Kolmogorov ou à l'entropie de Shannon, la complexité de Lempel-Ziv conditionnelle est définie comme étant $L_M(X|Y) = L_M(X,Y) - L(X)$. L'information mutuelle algorithmique est alors $I_{L_M}(X : Y) = L(X) + L(Y) + L_M(X,Y)$. Cette information mutuelle algorithmique est symétrique par définition (contrairement à celle de Kolmogorov). Cependant elle n'est pas positive : $I_{L_M}(X : Y) \geq -1$.

Si cette approche de la complexité jointe présente donc un intérêt théorique certain, elle se heurte en pratique à des difficultés liées à la grande taille de l'alphabet mise en jeu pour Z comme nous le verrons dans la section II.3.1.

I.2.3.3 Approche basé sur la causalité (2010)

Au lieu de partir de la définition de la fonction $< \cdot, \cdot >$, Steudel et al. [SJS10] partent de l'équation I.9 du théorème I.1.6. Rappelons que ce théorème dit qu'il est équivalent, à une constante près, d'encoder simultanément X et Y ou d'encoder X puis Y connaissant X.

Soit deux alphabets $\mathcal{A}_X$ et $\mathcal{A}_Y$, alors prenons un caractère $\beta \notin \mathcal{A}_X \cup \mathcal{A}_Y$. La complexité de Lempel-Ziv jointe est alors définie [SJS10] par $L_S(X,Y) = L(X\beta Y)$. C'est-à-dire que le processus de production va d'abord décomposer X puis va décomposer Y en se servant non seulement de son passé, mais aussi de X. Le caractère β est là pour empêcher qu'une étape du processus de production soit à cheval entre X et Y. L'inconvénient d'une telle complexité jointe est qu'elle perd sa symétrie. En effet, le processus de production de X puis de Y n'est pas le même que celui de Y puis X. L'information mutuelle algorithmique $I_{L_S}(X : Y) = L(X)+L(Y)+L_S(X,Y)$ n'est donc pas symétrique tout comme l'information mutuelle algorithmique de Kolmogorov. Et tout comme la complexité de Lempel-Ziv multidimensionnelle, l'information mutuelle algorithmique n'est plus positive, $I_{L_S}(X : Y) \geq -1$. Cependant il est facile de l'implémenter et de l'utiliser en pratique.

En résumé

Contrairement à la complexité de Kolmogorov, il existe plusieurs définitions de la complexité de Lempel-Ziv jointe et conditionnelle.

Ziv et Merhav en 1993 dans [ZM93]

- *Principe* : la complexité conditionnelle de X sachant Y est le nombre d'étapes du processus qui ne peut faire que des opérations `insérer` et des opérations `copier` depuis Y et uniquement depuis Y pour générer X.
- *Avantage* : elle permet de savoir si X s'explique bien grâce à Y dans le cadre d'une divergence.
- *Inconvénient* : il est impossible de définir la théorie algorithmique de l'information basée sur la complexité de Lempel-Ziv à partir de cette définition.

Zozor et al. en 2005 dans [ZRB05]

- *Principe* : elle étend la complexité de Lempel-Ziv aux chaînes vectorielles.
- *Avantage* : il est possible de définir la théorie algorithmique de l'information à partir de cette définition.
- *Inconvénient* : elle est difficile à implémenter à cause de la grande taille de l'alphabet.

Steudel et al. en 2013 dans [SJS10]

- *Principe* : la complexité jointe est la complexité de la concaténation de X et Y.
- *Avantage* : il est possible de définir la théorie algorithmique de l'information à partir de cette définition et elle est calculable.
- *Inconvénient* : avec cette définition l'information mutuelle algorithmique n'est pas symétrique.

I.3 Compression et complexité

Comme nous l'avons vu précédemment dans la section I.2.2, il existe différentes façons de mesurer l'information contenue dans une chaîne de caractères X. Une de ces mesures est la complexité de Kolmogorov, cependant elle n'est pas calculable dans un temps fini. Toutefois, puisque le but d'un compresseur est qu'une chaîne occupe le moins de place possible une fois compressée, il n'est pas déraisonnable de vouloir approcher la complexité de Kolmogorov par la taille de la chaîne compressée. Tout d'abord, il convient de passer en revue le fonctionnement des principaux algorithmes de compressions sans perte. Dans un second temps, nous montrons comment il est possible de les utiliser pour construire une théorie algorithmique de l'information.

I.3.1 Compresseurs

La complexité de Lempel-Ziv n'a pas seulement permis de développer une théorie algorithmique de l'information calculable, elle est aussi à la base d'algorithmes de compression que nous utilisons très régulièrement : `GZIP` ([Deu91]) et `LZMA` ([Pav]). Il existe aussi d'autres familles de compresseurs sans perte reposant sur des idées différentes comme la compression par blocs ou par prédiction à reconnaissance partielle ([Sew97]) que nous mentionnons plus brièvement.

I.3.1.1 `GZIP` et `LZMA` : compresseurs basés sur la complexité de Lempel-Ziv

• `LZ77`

En 1977, Lempel et Ziv décrivent dans [ZL77] un algorithme de compression basé sur la complexité de Lempel-Ziv [LZ76]. Cependant, pour que l'algorithme soit exécutable, il faut fixer quelques limites aux opérations `copier` et `insérer`. Tout d'abord, ils fixent un buffer de taille l_b, les opérations de reproduction ne peuvent alors être faites que dans les l_b derniers caractères et non dans l'intégralité de la chaîne comme dans la complexité de Lempel-Ziv. De plus l'opération `copier` ne peut pas excéder la longueur maximale l_{max}, alors que la taille du `copier` n'est pas limitée dans la complexité de Lempel-Ziv.

Hormis ces deux limites, le processus de production reste exactement le même. Se pose maintenant la question de l'encodage de l'historique exhaustif. La i^{eme} étape de production de E, $E_i(X)$, est exhaustive, c'est-à-dire que $X(1, e_{i-1}) \Rightarrow X(1, e_i)$ et $X(1, e_{i-1}) \not\Rightarrow X(1, e_i)$. On peut donc décomposer cette étape en deux sous-étapes :

1. un `copier` depuis le pointeur de reproduction t_i, qui doit être dans le buffer, jusqu'à $t_i + l_i$, où l_i est la longueur du `copier` qui doit être inférieur à l_{max},
2. un `insérer`.

Le symbole C_i qui encode l'étape de production exhaustive E_i est donc décomposé en trois sous-symboles : $C_i = C_{i1}C_{i2}C_{i3}$.

- C_{i1} représente le pointeur t_i écrit sur l'alphabet $\mathcal{A}$ de taille α.
- C_{i2} représente le pointeur l_i écrit sur l'alphabet $\mathcal{A}$ de taille α.
- C_{i3} est le nouveau caractère à `insérer`.

La longueur d'un tel symbole en base α est donc $l(C_i) = 1 + \lceil log_\alpha l_b \rceil + \lceil log_\alpha l_{max} \rceil$, où $\lceil\rceil$ désigne la partie entière supérieure.

Nous allons détailler étape par étape le processus de production de `LZ77`. Pour comprendre comment est réalisée chaque étape, nous nous appuyons sur la figure I.1 et nous expliquons chacun des éléments qui la compose.

- La chaîne de '0' et de '1' est la chaîne X.
- Le symbole $\Downarrow$ nous indique où nous en sommes dans le processus d'encodage.
- Les caractères bleus sont les caractères qui peuvent être générés par un `copier` (dans cet exemple, l_{max}=4).
- Les caractères verts sont les caractères qui peuvent être des références pour un `copier` (ici, l_b=4).

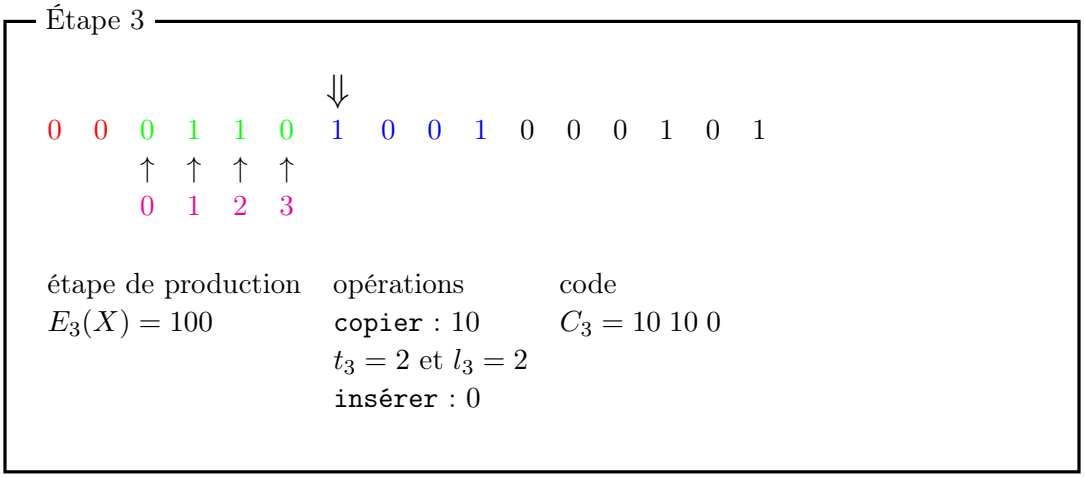


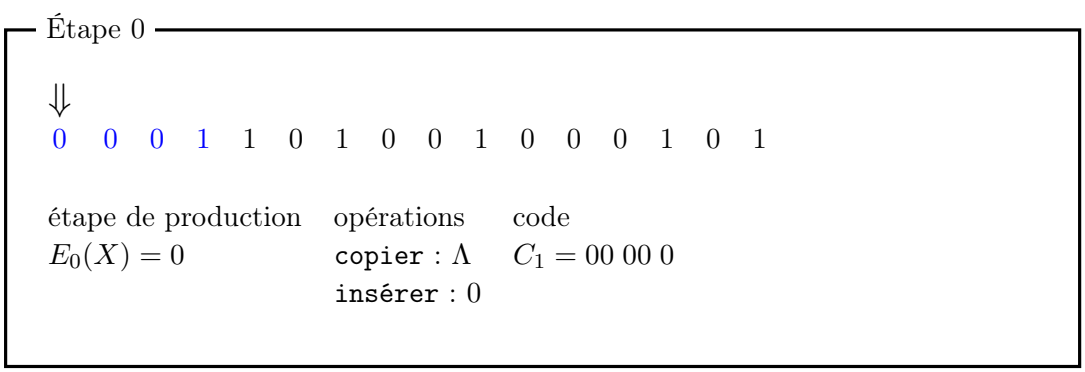
FIGURE I.1 – Exemple d'une étape d'encodage du processus de production de LZ77

- Les caractères rouges sont les caractères en dehors du buffer, il n'est donc plus possible de faire référence à ces derniers, ils ont été "oubliés".
- Les caractères violets sont les indices des caractères à l'intérieur du buffer.
- L'étape de production représente les caractères qui sont encodés dans cette étape.
- Les opérations sont les opérations `copier` et `insérer` réalisées lors de cette étape.
- Le code est le mot généré pour compresser les caractères de l'étape de production.

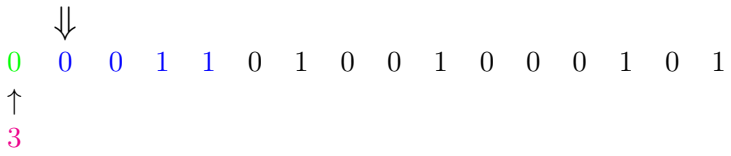
Maintenant, reprenons par exemple l'alphabet $\mathcal{A} = \{0, 1\}$ et la chaîne de caractères $X = 0001101001000101 \in \mathcal{A}^*$. Pour rappel la décomposition exhaustive était :

$$0 \cdot 001 \cdot 10 \cdot 100 \cdot 1000 \cdot 101$$

On pose $l_{max} = 4$ et $l_b = 4$. L'encodage réalisé par LZ77 est représenté ci-dessous, étape par étape.

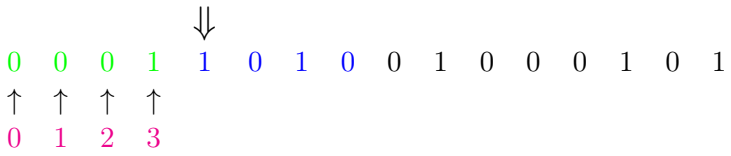


Étape 1



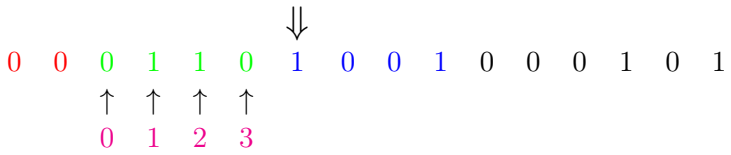
étape de production	opérations	code
$E_0(X) = 001$	`copier` : 00	$C_1 = 11\ 10\ 1$
	$t_1 = 3$ et $l_1 = 2$	
	`insérer` : 1	

Étape 2



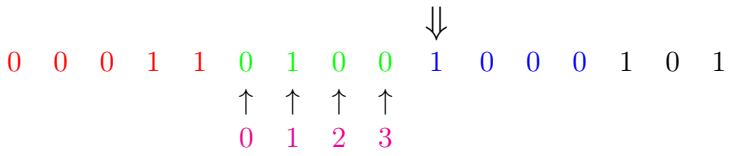
étape de production	opérations	code
$E_2(X) = 10$	`copier` : 1	$C_2 = 11\ 01\ 0$
	$t_2 = 3$ et $l_2 = 1$	
	`insérer` : 0	

Étape 3



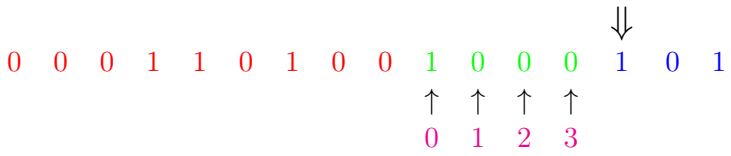
étape de production	opérations	code
$E_3(X) = 100$	`copier` : 10	$C_3 = 10\ 10\ 0$
	$t_3 = 2$ et $l_3 = 2$	
	`insérer` : 0	

Étape 4



étape de production	opérations	code
$E_4(X) = 1000$	`copier` : 100	$C_4 = 01\ 11\ 0$
	$t_4 = 1$ et $l_4 = 3$	
	`insérer` : 0	

Étape 5



étape de production	opérations	code
$E_5(X) = 101$	`copier` : 10	$C_4 = 00\ 10\ 1$
	$t_5 = 0$ et $l_5 = 2$	
	`insérer` : 1	

Ainsi la compression de la chaîne $X = 0001101001000101$ en utilisant `LZ77` est-elle :

$$X_{\texttt{LZ77}} = C_0\{00\ 00\ 0\}C_1\{11\ 10\ 1\}C_2\{11\ 01\ 0\}C_3\{10\ 10\ 0\}C_4\{01\ 11\ 0\}C_5\{00\ 10\ 1\}.$$

La taille de X compressé est plus longue que celle de X dans cet exemple ($l(X_{compress}) = 25$ bits au lieu de $l(X) = 16$) mais cela n'a rien de choquant, car nous avons utilisé une chaîne de petite taille pour illuster notre exemple. En effet, comme l'a souligné Kolmogorov dans [Kol65] "une estimation quantitative individuelle de l'information n'est significative que lorsque la quantité d'informations est suffisamment importante". Ainsi pour que la compression soit effective, c'est-à-dire qu'elle rende la chaîne compressée plus petite et non plus grande, il faut que la chaîne X soit suffisamment longue.

Pour conclure, Lempel et Ziv ont proposé une nouvelle méthode de compression appelée `LZ77`. Cette méthode est basée sur la décomposition de l'historique exhaustif de la complexité de Lempel-Ziv. Elle encode chaque élément de la décomposition en un symbole de taille fixe. Cependant, pour que l'algorithme soit réalisable deux limites sont imposées : les `copier` ne doivent pas faire référence au-delà de la limite l_b et ne doivent pas excéder en taille l_{max}.

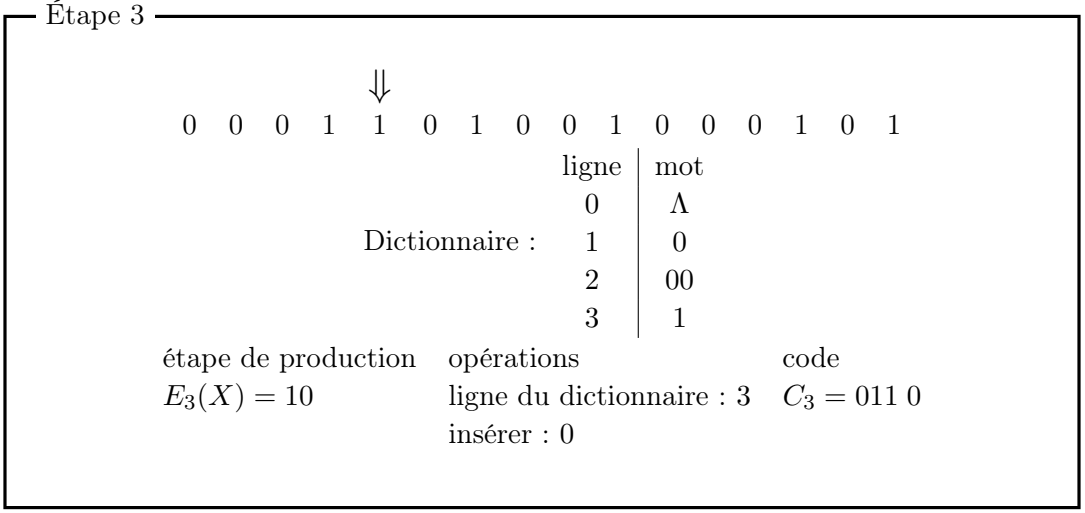


FIGURE I.2 – Exemple d'une étape d'encodage du processus de production de `LZ78`

• `LZ78`

La limite de la taille du buffer dans `LZ77` est due au fait que retrouver la plus longue chaîne de caractères pour faire l'opération `copier` demande une forte puissance de calcul et une certaine quantité de mémoire. C'est pourquoi, l_b ne peut pas être très grand. De plus, `LZ77` ne peut pas retrouver une chaîne de caractères si elle se situe au-delà de l_b. Pour pallier ce problème, Lempel et Ziv ont proposé une nouvelle version de `LZ77` plus économe en mémoire qui utilise un dictionnaire global au lieu d'une fenêtre glissante. La taille du dictionnaire, paramétrable, permet de maîtriser la quantité de mémoire utilisée. Cette méthode de compression est appelée `LZ78` et est décrite dans [ZL78].

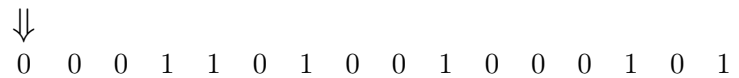
À l'initialisation, le dictionnaire est vide. Puis à l'étape i, au lieu de faire une opération `copier` dans une fenêtre, nous allons chercher dans le dictionnaire le plus long mot possible qui décrit les prochains caractères à encoder. Ensuite, tout comme dans `LZ77`, une opération `insérer` est réalisée. Le symbole émis lors de cette étape est composé de la ligne du dictionnaire, suivi du caractère à insérer. Avant de passer à l'étape $i + 1$, on ajoute une ligne supplémentaire au dictionnaire contenant le nouveau mot, c'est-à-dire le plus long mot du dictionnaire concaténé avec le caractère inséré.

Tout comme pour `LZ77`, nous allons détailler le fonctionnement étape par étape de `LZ78`. Pour comprendre comment est réalisée chaque étape, nous nous appuyons sur la figure I.2 et nous expliquons chacun des éléments qui la compose.

- La chaîne de '0' et de '1' est la chaîne X.
- Le symbole ⇓ nous indique où nous en sommes dans le processus d'encodage.
- Le dictionnaire est composé de deux colonnes, la première est l'index dans le dictionnaire et la deuxième contient la chaîne associée.
- L'étape de production représente les caractères qui sont encodés dans cette étape.
- Les opérations sont les opérations `copier` et `insérer` réalisées lors de cette étape.
- Le code est le mot généré pour compresser les caractères de l'étape de production.

Reprenons notre exemple $X = 0001101001000101$.

Étape 0

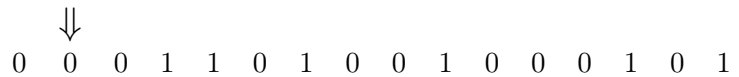


Dictionnaire :

ligne	mot
0	Λ

étape de production	opérations	code
$E_0(X) = 0$	ligne du dictionnaire : 0 insérer : 0	$C_0 = 000\ 0$

Étape 1

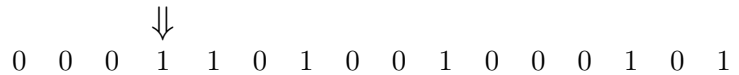


Dictionnaire :

ligne	mot
0	Λ
1	0

étape de production	opérations	code
$E_1(X) = 00$	ligne du dictionnaire : 1 insérer : 0	$C_1 = 001\ 0$

Étape 2



Dictionnaire :

ligne	mot
0	Λ
1	0
2	00

étape de production	opérations	code
$E_2(X) = 1$	ligne du dictionnaire : 0 insérer : 1	$C_2 = 000\ 1$

Étape 3

$$\begin{array}{cccccccccccccccc} & & & & & \Downarrow & & & & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

Dictionnaire :

ligne	mot
0	Λ
1	0
2	00
3	1

étape de production	opérations	code
$E_3(X) = 10$	ligne du dictionnaire : 3 insérer : 0	$C_3 = 011\ 0$

Étape 4

$$\begin{array}{cccccccccccccccc} & & & & & & \Downarrow & & & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

Dictionnaire :

ligne	mot
0	Λ
1	0
2	00
3	1
4	10

étape de production	opérations	code
$E_4(X) = 100$	ligne du dictionnaire : 4 insérer : 0	$C_4 = 100\ 0$

Étape 5

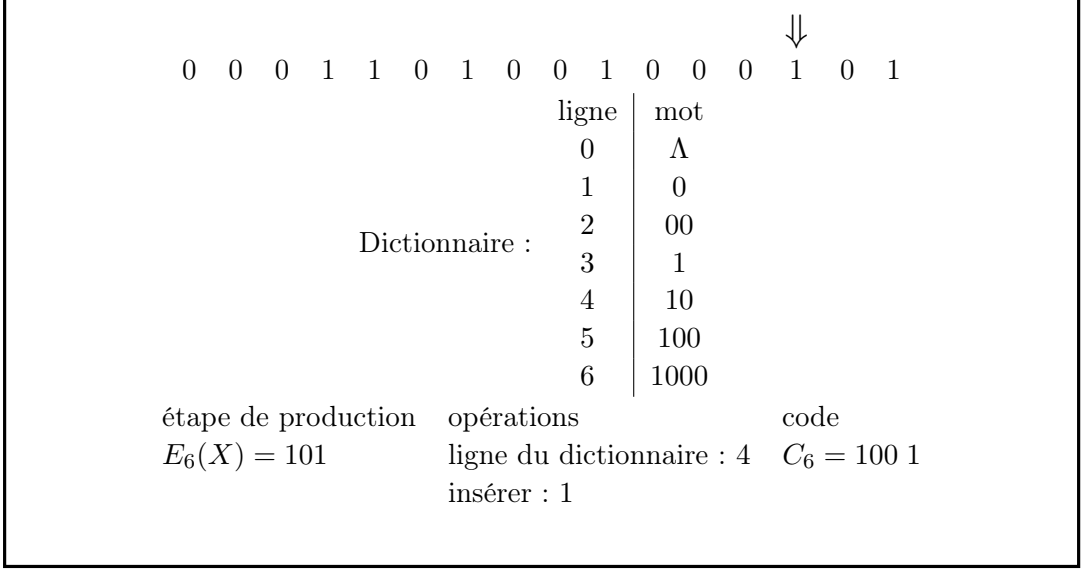
$$\begin{array}{cccccccccccccccc} & & & & & & & & \Downarrow & & & & & & & \\ 0 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 1 \end{array}$$

Dictionnaire :

ligne	mot
0	Λ
1	0
2	00
3	1
4	10
5	100

étape de production	opérations	code
$E_5(X) = 1000$	ligne du dictionnaire : 5 insérer : 0	$C_5 = 101\ 0$

Étape 6



Ainsi la compression de $X = 0001101001000101$ en utilisant `LZ78` est :

$$X_{\texttt{LZ78}} = C_0\{000\ 0\}C_1\{001\ 0\}C_2\{000\ 1\}C_3\{011\ 0\}C_4\{100\ 0\}C_5\{101\ 0\}C_6\{100\ 1\}.$$

La compression avec `LZ78` a donc besoin d'une étape supplémentaire pour la compression de X par rapport à `LZ77`. Cependant, Lempel et Ziv ont montré que les taux de compression de ces deux méthodes sont asymptotiquement égaux.

Pour conclure, deux méthodes de compression, `LZ77` et `LZ78` ont été développées à partir de la complexité de Lempel-Ziv. Ces deux méthodes reposent sur le processus de production développé dans [LZ76], c'est-à-dire qu'elles cherchent à chaque étape le plus grand `copier` possible, soit dans les l_b derniers caractères pour `LZ77` soit dans un dictionnaire pour `LZ78`, puis elles réalisent l'opération `insérer`.

• `GZIP`

En 1991, une implémentation d'un compresseur basée sur `LZ77` est proposée par Phil Kats et est standardisée [Deu91] sous l'appellation `GZIP`. Cet algorithme se décompose en deux étapes. La première est une décomposition basée sur les deux opérations `copier` et `insérer`. Les symboles de cette décomposition sont ensuite compressés grâce à un codage entropique de Huffman ([Huf52]).

Dans un premier temps, regardons comment Deutsch a adapté `LZ77` dans [Deu91]. Le premier choix d'implémentation de `GZIP`, assez naturel, consiste à utiliser des octets pour les caractères de $\mathcal{A}$ ($\alpha = 256$). Ensuite, la longueur de buffer, l_b est fixée à 32 kilo-octets. C'est-à-dire qu'une opération `copier` ne peut pas faire référence à une chaîne située plus de 32 kilo-octets dans le passé. La limite de la taille des opérations `copier` est de $l_{max} = 258$ octets. De surcroît, les opérations `copier` sont soumises à une taille minimale de 3 octets, notée l_{min}.

Étape 3

				⇓												
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	3	2	1													

étape de production	opération	code
$E_3(X) = 1$	`insérer` : 1	L(1)

FIGURE I.3 – Exemple d'une étape d'encodage du processus de production de `GZIP`

Ces limites sont dues à des contraintes d'implémentation. Le fait de prendre des octets est une simplification évidente sur les ordinateurs qui stockent les informations sous cette forme. La limite du buffer est due au fait que l'algorithme de compression doit être rapide. En limitant l_b à $32kO$, la recherche de tous les candidats à l'opération `copier` la plus longue est bien plus rapide. La limite minimum de la taille des opérations `copier` est due à l'implémentation suggérée pour la structure de recherche des sous-chaînes : une table de hachage indexée par les trois premiers octets de la sous-chaîne à rechercher. À l'étape $E_i = X(e_{i-1} + 1, e_i)$ du processus de production, nous cherchons e_i de façon à faire le plus long `copier` parmi les l_b derniers symboles de X. Pour trouver ces candidats, nous devons parcourir un dictionnaire qui ne contient que des mots de taille 3 octets.

De plus `GZIP` n'utilise pas les symboles de `LZ77` qui sont de taille fixe et qui contiennent à chaque fois une opération `copier` et une opération `insérer`. Les deux opérations sont représentées par deux types de symboles distincts :

- `Z`$(-d \to l)$: ce symbole correspond à l'opération `copier`. Si une référence de taille supérieure à l_{min} est trouvée dans les l_b derniers octets, alors le symbole `Z`$(-d \to l)$ est émis, où d est la distance en nombre d'octets où se trouve la référence par rapport à l'octet actuellement encodé et l la longueur de la référence trouvée.
- `L`(o) : ce symbole correspond à l'opération `insérer`. Si aucune référence n'est trouvée alors on émet un littéral avec le symbole `L`(o), où o est l'octet à insérer.

Tout comme pour `LZ77` et `LZ78`, nous allons détailler le fonctionnement étape par étape de `GZIP`. Pour comprendre comment est réalisée chaque étape, nous nous appuyons sur la figure I.3 et nous expliquons chacun des éléments qui la compose.

- La chaîne de '0' et de '1' est la chaîne X.
- Le symbole ⇓ nous indique où nous en sommes dans le processus d'encodage.
- Tous les caractères à gauche de ⇓ sont dans le buffer (car $l_b = 32kO$).
- L'étape de production représente les caractères qui sont encodés à l'étape courante.
- L'opération est l'opération `copier` ou `insérer` réalisée lors de cette étape.
- Le code est le mot généré pour compresser les caractères de l'étape de production.

Reprenons notre exemple $X = 0001101001000101$.

Étape 0

		⇓														
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance																

étape de production	opération	code
$E_0(X) = 0$	`insérer` : 0	L(0)

Étape 1

		⇓														
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	1															

étape de production	opération	code
$E_1(X) = 0$	`insérer` : 0	L(0)

Étape 2

			⇓													
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	2	1														

étape de production	opération	code
$E_2(X) = 0$	`insérer` : 0	L(0)

Étape 3

				⇓												
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	3	2	1													

étape de production	opération	code
$E_3(X) = 1$	`insérer` : 1	L(1)

Étape 4

					⇓											
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	4	3	2	1												

étape de production	opération	code
$E_4(X) = 1$	`insérer` : 1	L(1)

Étape 5

						⇓										
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	5	4	3	2	1											

étape de production	opération	code
$E_5(X) = 0$	`insérer` : 0	`L`(0)

Étape 6

							⇓									
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	6	5	4	3	2	1										

étape de production	opération	code
$E_6(X) = 1$	`insérer` : 1	`L`(1)

Étape 7

								⇓								
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	7	6	5	4	3	2	1									

étape de production	opération	code
$E_7(X) = 001$	`copier` de 6 en arrière et de longueur 3	`Z`$(-6 \to 3)$

Étape 8

											⇓					
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	10	9	8	7	6	5	4	3	2	1						

étape de production	opération	code
$E_8(X) = 0001$	`copier` de 10 en arrière et de longueur 4	`Z`$(-10 \to 4)$

Étape 9

														⇓		
X	0	0	0	1	1	0	1	0	0	1	0	0	0	1	0	1
distance	13	12	11	10	9	8	7	6	5	4	3	2	1			

étape de production	opération	code
$E_9(X) = 101$	`copier` de 9 en arrière et de longueur 3	`Z`$(-9 \to 3)$

La décomposition de X avec GZIP est donc :

$$X_{\texttt{GZIP}} = \texttt{L}(0)\texttt{L}(0)\texttt{L}(0)\texttt{L}(1)\texttt{L}(1)\texttt{L}(0)\texttt{L}(1)\texttt{Z}(-6 \to 3)\texttt{Z}(-10 \to 4)\texttt{Z}(-9 \to 3).$$

La production des symboles par GZIP ne correspond pas à la fin du processus de compression contrairement à LZ77 et LZ78. En effet, ces symboles ne sont pas de taille fixe et sont donc encodés avec un code de Huffman, [Huf52]. Le code de Huffman est déterminé à partir d'une estimation des probabilités d'apparition des symboles de source : un code court étant associé aux symboles de source les plus fréquents. Un code de Huffman est optimal au sens de la plus courte longueur pour un codage par symbole, et une distribution de probabilité connue.

• LZMA

En 2009, un nouvel algorithme de compression basé sur LZ77 et appelé LZMA fait son apparition dans [Pav]. Nous ne rentrerons pas dans les détails poussés de cet algorithme, cependant nous mentionnons ici trois différences majeures entre GZIP et LZMA. La première est que la taille du buffer de LZMA est de 4 GO au lieu de 32 kO pour GZIP. Ainsi LZMA peut chercher des références beaucoup plus loin dans le passé de la chaîne à compresser pour faire des opérations copier. Le prix à payer pour indexer un passé aussi lointain est une quantité de mémoire vive couramment de l'ordre de la centaine de méga-octets. Une autre grande différence est que la recherche de références est "multi-thread", c'est-à-dire en parallèle sur plusieurs cœurs de l'ordinateur, sur les 2, 3 ou 4 prochains octets contrairement à GZIP qui cherche uniquement sur les 3 prochains octets. Ainsi LZMA peut trouver des références de taille 2 contrairement à GZIP. De plus la parallélisation des recherches permet d'accélérer le programme. Enfin la dernière différence concerne l'étape de codage entropique qui utilise un codeur arithmétique au lieu d'un codeur de Huffman.

I.3.1.2 D'autres familles de compresseur : prédiction par reconnaissance partielle et transformée de Burrows-Wheeler

D'autres familles de compresseur, qui ne sont pas basées sur la complexité de Lempel-Ziv, existent. Une première approche est celle basée sur la prédiction par reconnaissance partielle. La seconde famille utilise quant à elle la transformée de Burrows-Wheeler (aussi appelée compression par blocs). Par exemple le compresseur BZIP2, [Sew97], est basé sur la transformée de Burrows-Wheeler et a été publié en 1997. Il possède un meilleur ratio de compression que GZIP pour le texte, mais il est plus lent.

En résumé

Il existe différentes familles de compresseurs dont deux seront utilisées par la suite.

Lempel-Ziv : cette famille est basée sur la complexité de Lempel-Ziv, les deux compresseurs les plus connus de cette famille sont `GZIP` et `LZMA`.

Compression par blocs : cette famille repose sur la transformée de Burrows-Wheeler, le compresseur `BZIP2` donne un meilleur taux de compression que `GZIP` mais va beaucoup plus lentement.

I.3.2 La compression comme estimateur de la complexité de Kolmogorov

Nous avons vu qu'il existe différents types de compresseurs. Cependant pour que la taille du fichier compressé soit une approximation raisonnable de la complexité de Kolmogorov, il faut qu'ils respectent certaines propriétés. Si ces propriétés sont vérifiées alors la taille du fichier compressé constitue une base possible pour une théorie algorithmique de l'information basée sur une approximation calculable de la complexité de Kolmogorov.

I.3.2.1 Compresseur normal

La complexité de Kolmogorov est la taille du plus petit programme capable de générer la chaîne de caractères. Cependant, elle n'est pas calculable en un temps fini. Cilibrasi et Vitányi [CV05] ont donc proposé d'estimer la complexité de Kolmogorov grâce à la taille de la chaîne compressée. En effet, le compresseur parfait serait celui qui créerait le plus petit fichier possible. Nous retrouvons donc une définition très proche de la complexité de Kolmogorov. De plus nous avons vu dans la section I.3.1.1 qu'une famille de compresseurs a été élaborée grâce à la complexité de Lempel-Ziv.

La taille de la chaîne X compressée est notée $C(X)$ pour n'importe quel type de compresseur. Si nous utilisons un compresseur en particulier alors nous noterons $C_{compresseur}(X)$. Cependant pour que le compresseur puisse se substituer à la complexité de Kolmogorov, il faut qu'il respecte certaines propriétés, dites de normalité [CV05].

Définition I.3.1 (Compresseur normal)**.**
Soient X et $Y \in \mathcal{A}^*$, alors un compresseur est normal s'il vérifie (à une constante $\mathcal{O}(log(max\{l(X), l(Y), l(Z)\}))$ près), les propriétés suivantes :

Idempotence *:* $C(XX) = C(X)$,

Monotonicité *:* $C(XY) \geq C(X)$,

Symétrie *:* $C(XY) = C(YX)$,

Distributivité *:* $C(XY) + C(Z) \leq C(XZ) + C(YZ)$.

L'idempotence est due au fait que le compresseur verra la répétition de X et donc la répétition de X n'aura qu'une faible influence sur la taille du fichier compressé. La mo-

notonicité est due au fait que si l'on rajoute de l'information alors forcément la version compressée du fichier sera plus grande. La symétrie signifie qu'il revient au même de compresser X puis Y que de compresser Y puis X. La distributivité est une propriété de la complexité de Kolmogorov et il est donc nécessaire de la respecter pour faire le parallèle entre compresseur normal et complexité de Kolmogorov.

Ainsi, sous cette condition de normalité, la taille d'une chaîne compressée est un estimateur de la complexité de Kolmogorov. Cilibrasi et Vitányi ont montré que tous les compresseurs basés sur Lempel-Ziv (`GZIP` et `LZMA`), sur la transformée de Burrows-Wheeler (`BZIP2`) ou sur la prédiction par reconnaissance partielle sont des compresseurs quasi-normaux en pratique.

Dans la figure I.4, nous cherchons à quantifier l'asymétrie en pratique des compresseurs normaux. Pour cela, nous prenons l'ensemble du jeu de données que nous allons conserver tout au long de ce manuscrit [2] : des chaînes i.i.d. d'une loi uniforme discrète avec différentes tailles d'alphabet, des chaînes de Markov avec différentes tailles d'alphabet, de l'ADN et la Déclaration Universelle des Droits de l'Homme dans différentes langues. Pour deux chaînes de ce jeu de données, nous allons observer l'asymétrie d'un compresseur normal. Pour deux chaînes quelconques X et Y de notre jeu de données, les mesures que nous effectuons sont :

Asymétrie : $C(XY)-C(YX)$. Cette quantité, qui doit être le plus proche de zéro possible, permet d'évaluer la différence entre compresser d'abord X puis Y ou Y puis X.

Normalisation : $\frac{C(XY)-C(YX)}{C(X)+C(Y)}$. Afin de pouvoir comparer la complexité de chaînes de taille différente, il faut normaliser en utilisant la borne supérieure $C(XY) \leq C(X) + C(Y)$.

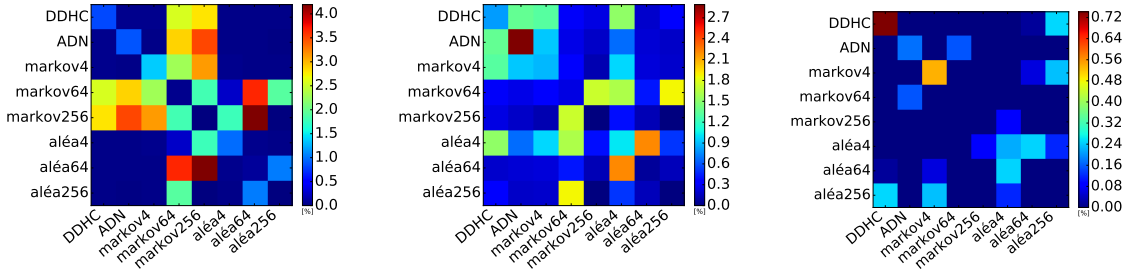
Les résultats sont synthétisés dans la figure I.4. L'abscisse représente le jeu de données dont la chaîne X est extraite et l'axe des ordonnées représente celui dont provient la chaîne Y. Nous représentons, dans la figure I.4, l'asymétrie maximale pour tous les couples de chaînes prises dans le jeu de données. La variance et la moyenne pour chaque jeu de données sont présentées dans les tableaux B.1, B.2 et B.3 présents dans l'annexe B.

L'asymétrie des compresseurs normaux ne dépasse pas 4% pour `GZIP`, 3% pour `LZMA` et même 1% pour `BZIP2`. Comme ces compresseurs ont été considérés comme normaux par Cilibrasi et Vitányi [CV05], nous considérerons par la suite que des erreurs de l'ordre de 4% sont donc négligeables.

I.3.2.2 La théorie algorithmique de l'information avec un compresseur normal

Un compresseur normal est un estimateur de la complexité de Kolmogorov. Il est cependant impossible de compresser deux chaînes simultanément afin d'estimer la complexité de Kolmogorov jointe ou de compresser une chaîne sachant une autre pour estimer la complexité conditionnelle. Pour définir la taille des chaînes X et Y compressées conjointement, $C(X, Y)$, Cilibrasi et Vitányi dans [CV05], utilisent la même approche que Steudel et al. décrite à la section I.2.3.3 ainsi que l'équation I.9 de la complexité de Kolmogorov.

2. Pour la description des données voir l'annexe D.



(a) `GZIP`. (b) `LZMA`. (c) `BZIP2`.

FIGURE I.4 – Asymétrie des compresseurs normaux réels : calcul de $\frac{C(XY)-C(YX)}{C(X)+C(Y)}$ pour X et Y appartenant aux différents jeux de données. Représentation des maxima pour l'ensemble des chaînes X du jeu de données en abscisse connaissant chaque chaîne Y du jeu de données en ordonnée. Si la case est bleue alors l'asymétrie est faible mais si la case est rouge alors l'asymétrie est très prononcée.

Définition I.3.2 (Compression jointe)**.**
L'estimation de la complexité de Kolmogorov jointe de deux chaînes X et $Y \in \mathcal{A}^*$ par un compresseur normal est :

$$C(X,Y) = C(XY)$$

La compression jointe de X et Y est équivalente à la compression de la concaténation de X et Y. Grâce à cette définition et au théorème I.1.6, il est alors possible de définir la compression conditionnelle.

Définition I.3.3 (Compression conditionnelle)**.**
L'estimation de la complexité de Kolmogorov conditionnelle de $X \in \mathcal{A}^*$ sachant $Y \in \mathcal{A}^*$ par un compresseur normal est :

$$C(X|Y) = C(YX) - C(Y). \tag{I.12}$$

En effet, le processus de compression va d'abord compresser Y puis il va compresser X en utilisant autant d'information de Y que son implémentation le permet. Ainsi la compression de YX peut être vue comme étant constituée, dans un premier temps, de la compression de Y, puis la compression de X connaissant Y dans un second temps. C'est la raison pour laquelle, il suffit de retirer la compression de Y à la compression de YX pour estimer la compression de X sachant Y. Il est donc tout à fait possible d'utiliser des compresseurs normaux comme estimateurs de la complexité de Kolmogorov. Grâce à eux, il est possible d'estimer la complexité jointe et conditionnelle et donc de calculer une information mutuelle algorithmique basée sur les compresseurs.

En résumé

Comme l'ont montré Cilibrasi et Vitányi, dans [CV05], les compresseurs normaux sont des bonnes approximations de la complexité de Kolmogorov.

- *Principe* : le compresseur cherche à créer le plus petit fichier possible à partir de la chaîne X, la taille du fichier compressé est donc une bonne approximation de la complexité de Kolmogorov.
- *Avantage* : l'implémentation des compresseurs permet d'approcher la complexité de Kolmogorov en pratique.
- *Inconvénient* : un compresseur ne compresse qu'un seul fichier, il faut donc faire des approximations pour calculer la compression jointe et conditionnelle.

Conclusion

Nous avons vu qu'une manière pour évaluer l'information contenue dans une chaîne de caractères de taille finie sur un alphabet de taille finie est l'entropie de Shannon, plus la chaîne est aléatoire plus son l'entropie est élevée, plus la chaîne est ordonnée plus l'entropie est faible. Grâce à l'entropie de Shannon, la théorie probabiliste de l'information s'est développée. Cette théorie permet de calculer l'information d'une chaîne en connaissant une autre, l'information de deux chaînes simultanément ou encore l'indépendance algorithmique entre deux chaînes grâce à l'information mutuelle algorithmique. Cependant, l'entropie de Shannon n'est pas la seule mesure d'information. Dans les années 60, Solomonoff, Kolmogorov et Chaitin proposent une nouvelle mesure d'information appelée complexité de Kolmogorov qui est la taille du plus petit programme capable de générer la chaîne de caractères. Cette mesure ne repose plus sur la théorie des probabilités comme entropie de Shannon mais bien sur l'algorithmique. Tout comme pour l'entropie de Shannon, il est possible de définir le pendant algorithmique de la théorie probabiliste de l'information qui est alors appelé théorie algorithmique de l'information.

Cependant la complexité de Kolmogorov n'est pas calculable en temps fini, c'est pourquoi en 1976 Lempel et Ziv ont proposé une nouvelle complexité. Cette complexité repose sur un processus de production qui ne peut faire que des opérations `copier` depuis le passé de la chaîne et `insérer` un nouvel élément. La complexité est alors le nombre minimal d'opérations nécessaires pour générer la chaîne. La complexité de Lempel-Ziv fait partie de l'ensemble des mesures d'information et il est donc possible d'écrire la théorie algorithmique de l'information avec la complexité de Lempel-Ziv. Cependant il n'existe pas une unique définition de la complexité de Lempel-Ziv jointe et conditionnelle.

Une autre façon d'estimer la complexité de Kolmogorov est d'utiliser un compresseur. Il existe différents types de compresseurs, en particulier, toute une famille de compresseurs (`GZIP`, `LZMA`) basée sur la complexité de Lempel-Ziv. Ces compresseurs ont un bon ratio de compression dans un temps acceptable. Il existe cependant d'autres compresseurs comme `BZIP2`. Ces trois compresseurs sont des compresseurs normaux et peuvent donc

servir d'estimateur de la complexité de Kolmogorov. En effet, le compresseur parfait serait celui qui créerait le plus petit fichier à partir du fichier d'entrée. Cette définition est donc très proche de la complexité de Kolmogorov, et la taille du fichier peut être utilisée comme un bon estimateur de cette dernière. Cependant, il est impossible de compresser un fichier connaissant un autre fichier et il faut avoir recours à des approximations pour estimer la complexité conditionnelle.

Les compresseurs sont très utiles pour approcher la complexité de Kolmogorov inconditionnelle. Cependant, ils ne peuvent pas compresser des fichiers simultanément. Il est donc primordial de définir correctement toutes les possibilités de la complexité de Lempel-Ziv conditionnelles. Ainsi nous pourrons définir une théorie algorithmique de l'information basée sur la complexité de Lempel-Ziv. De plus, compter uniquement le nombre de symboles comme dans la complexité de Lempel-Ziv peut être restrictif, car cela ne tient pas compte des données et est sensible au bruit.

Chapitre II

SALZA : une nouvelle mesure d'information

Sommaire

La complexité de Kolmogorov n'est pas calculable en un temps fini comme nous l'avons vu dans le chapitre I. Deux possibilités permettent de l'estimer : la complexité de Lempel-Ziv et les compresseurs. Nous avons vu que l'approche conditionnelle était mal définie pour ces estimateurs et que les compresseurs avaient de fortes contraintes d'implémentations. Afin de pallier ces problèmes, nous proposons un nouvel estimateur de complexité appelé SALZA

Cet estimateur SALZA est basé sur la complexité de Lempel-Ziv et en particulier sur son implémentation GZIP (section II.1.1). Nous proposons différentes possibilités pour réaliser une décomposition de Lempel-Ziv conditionnelle. Ceci permet d'étendre la complexité de Lempel-Ziv à toutes ces possibilités (section II.1.2). Nous montrons enfin que compter le nombre d'opérations copier et insérer, comme le fait la complexité de Lempel-Ziv, ne retient pas toute l'information et qu'il est utile de prendre en compte la longueur des opérations copier (section II.1.3).

Grâce à ces remarques, nous proposons un nouvel estimateur de complexité SALZA dans la section II.2.1. Cet estimateur est basé sur le nombre et la longueur des opérations copier de la décomposition. Puis nous montrons que SALZA est bien une mesure d'information (section II.2.3), ce qui permet donc définir une théorie algorithmique de l'information

basée sur SALZA. Pour cela dans un premier temps, l'estimation de la complexité jointe basée sur SALZA est définie (section II.3.1). SALZA jointe nous permet alors de construire une information mutuelle algorithmique qui mesure l'indépendance algorithmique de deux chaînes ainsi qu'une information dirigée qui mesure l'indépendance directionnelle de deux chaînes de caractères (section II.3.2).

Enfin nous comparons en théorie le comportement de SALZA et de la complexité de Lempel-Ziv (section II.4.1). Puis une comparaison basée sur des simulations permet de comparer les comportement de SALZA avec ceux des compresseurs (section II.4.2).

II.1 Interprétation de X sachant Y

Comme nous l'avons vu précédemment, il existe différentes façons de calculer la complexité de Lempel-Ziv, soit avec la complexité introduite en 1976 [LZ76], soit en comptant le nombre de symboles de LZ77, LZ78 ou GZIP. GZIP étant un code connu et déjà implémenté, nous proposons d'utiliser le nombre de symboles de GZIP pour calculer la complexité de Lempel-Ziv. Il faut cependant faire quelques changements dans le code de GZIP car notre objectif est une nouvelle mesure d'information conditionnelle. De plus il existe différentes façons d'interpréter la complexité de Lempel-Ziv conditionnelle comme nous l'avons vu dans la section I.2.3. Nous allons ainsi introduire différentes approches permettant de décomposer, en opérations copier et insérer, une chaîne X connaissant une autre chaîne Y. Nous répertorions toutes les possibilités pour décomposer une chaîne X connaissant une autre chaîne Y. Par ailleurs, compter le nombre de symboles à la manière de la complexité de Lempel-Ziv ne permet pas de retenir toute l'information. C'est pourquoi nous introduisons une nouvelle quantité appelée *quantité significative* qui prend en compte la longueur des opérations copier.

II.1.1 Une décomposition basée sur GZIP

Notre objectif est de fournir un code fonctionnel qui permet de calculer la nouvelle mesure d'information conditionnelle SALZA. Pour cela, nous reprenons la décomposition de GZIP avec quelques modifications. Tout d'abord, la limite sur la taille de l'alphabet, α, imposée par GZIP est gardée. Ainsi α ne peut pas excéder 256, $\alpha \leq 256$. La contrainte qui force les opérations copier à faire une taille supérieure à trois est par contre supprimée, nous avons donc $l_{min} = 2$. De plus, le buffer n'existe plus, ce qui permet de poser $l_b = l_{max} = \underset{X \in \mathcal{X}}{max}\{l(X)\}$ où $\mathcal{X}$ est l'ensemble des chaînes étudiées. Pour plus de détails sur l'implémentation voir l'annexe E.

Notre objectif principal est de faire un estimateur de complexité conditionnelle. Cependant, GZIP n'est pas fait pour compresser une chaîne sachant une autre chaîne. Il faut donc modifier quelque peu le symbole Z de GZIP (présenté dans la section I.3.1.1) pour réaliser le conditionnement. Maintenant ces symboles s'écriront sous la forme :

$$\mathtt{Z}(s : i \to l),$$

où

- s est la chaîne à laquelle nous faisons référence, 0 si c'est à elle-même ;
- i est l'indice où commence l'opération `copier`, au lieu de la distance auparavant (voir section I.3.1.1) ;
- l est toujours la longueur de l'opération `copier`.

L'ensemble des longueurs de la décomposition de la chaîne est composé de la valeur 1 pour chaque opération `insérer` et de la longueur l de chaque opération `copier`. L'ensemble de ces longueurs est noté $\mathcal{L}$.

En résumé

La décomposition de `SALZA` est basée sur la décomposition `GZIP` (voir section I.3.1.1) avec quelques modifications.

- La taille de l'alphabet est toujours limitée à 256.
- Il n'y a plus de borne minimale ni de borne maximale pour la longueur des opérations `copier`.

Les symboles de `SALZA` sont très proches de ceux de `GZIP` mais quelques modifications ont été apportées car nous décomposons X sachant Y.

`insérer` : $\mathtt{L}(o)$: insère l'octet o.

`copier` : $\mathtt{Z}(s : i \to l)$: copier depuis le signal s les octets numéro i à numéro $i+l$.

$\mathcal{L}$: l'ensemble des longueurs de la décomposition, les opérations `insérer` ont une longueur 1 et les opérations `copier` ont une longueur $l > 1$.

II.1.2 Que signifie conditionner X sachant Y ?

Comme nous l'avons vu dans la section I.2.3, il existe plusieurs définitions de la complexité de Lempel-Ziv conditionnelle. Il existe en réalité bien plus de deux façons différentes pour décomposer X sachant Y. Pour bien comprendre quelles peuvent être ces différentes définitions, regardons la figure II.1 où chaque petit rectangle est un octet. Les deux chaînes X et Y sont alignées, c'est-à-dire que l'octet 0 de X est juste au-dessus de l'octet 0 de Y, que l'octet 1 de X est juste au-dessus de l'octet 1 de Y, ... et ainsi de suite. Dans cette figure, nous avons déjà décomposé plusieurs octets de X. Ce sont tous les octets de la première ligne à gauche de la barre verticale rouge qui représente l'instant présent. À cette étape, le but est de trouver la plus longue opération `copier` qui explique l'octet grisé et ceux qui suivent.

Cette opération `copier` peut commencer depuis 4 zones distinctes.

i Le passé de X : ce sont tous les octets déjà décomposés de X.

ii Le passé de Y : ce sont les octets simultanés au passé de X

iii Le présent de Y : c'est l'octet de Y qui correspond à l'octet qui va être encodé dans X.

iv Le futur de Y : ce sont les octets de Y après l'octet présent.

Il est donc possible de faire trois conditionnements différents par rapport à Y. Le type de conditionnement par rapport à Y est indiqué en haut à droite du signe *sachant que* $|$.

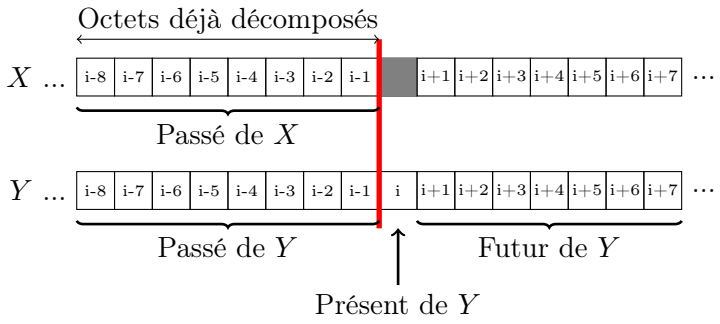


FIGURE II.1 – Schéma qui permet d'expliquer une étape de décomposition.

i $|^{-}$: le premier octet du `copier` peut être dans le passé de Y.

ii $|^{p}$: le premier octet du `copier` peut être dans le passé ou le présent de Y.

iii $|^{+}$: le premier octet du `copier` peut se situer n'importe où dans Y.

En plus de la connaissance de Y, il est possible d'ajouter ou non la connaissance de X. La connaissance de X est ajoutée en bas à gauche du signe *sachant que* $|$.

i $|$: on ne peut pas faire de `copier` dans X.

ii $\_|$: le premier octet du `copier` peut être dans le passé de X.

Il est tout à fait possible de combiner la connaissance de Y et la connaissance de X. Par exemple le conditionnement $\_|^{+}$ signifie que le premier octet du `copier` peut se situer n'importe où dans Y ou dans le passé de X. Lorsque nous ne voulons pas parler d'un conditionnement spécifique, mais du conditionnement en général, alors on note

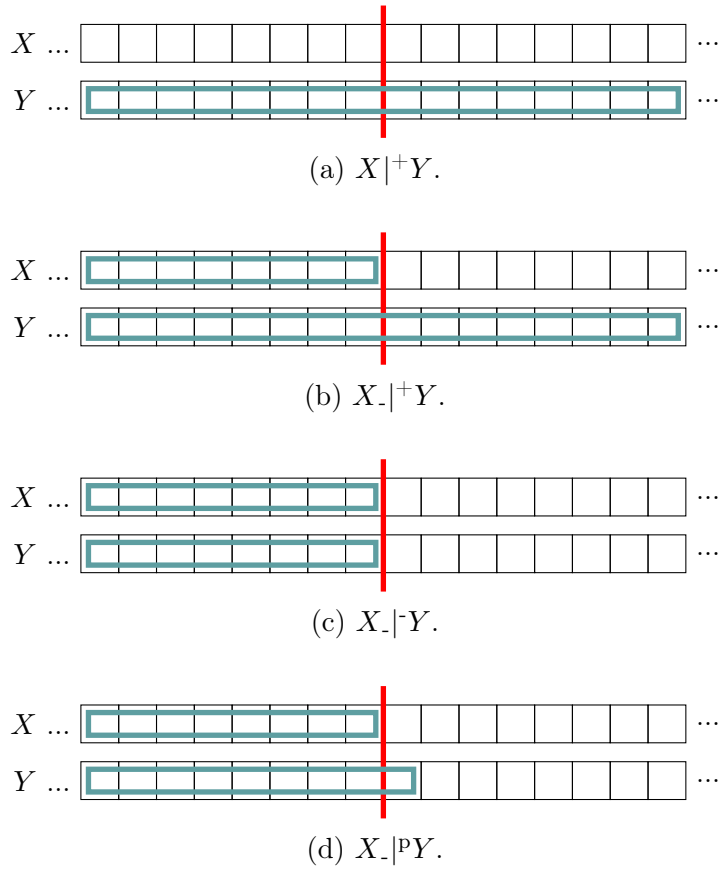
$$X \wr Y.$$

Nous allons maintenant décrire quatre conditionnements particulièrement utiles pour la suite de ces travaux. Les octets qui peuvent servir de référence selon le conditionnement choisi sont encadrés en bleu dans la figure II.2.

(a) $|^{+}$: cette décomposition est celle proposée par Ziv et Merhav dans [ZM93] pour créer une divergence (voir section I.2.3.1) et est représentée dans la figure II.2a. Dans cette décomposition, les opérations `copier` ne peuvent provenir que de Y, la redondance de X ne peut pas être prise en compte. Si, par exemple, $\mathcal{A}_X \cap \mathcal{A}_Y = \emptyset$, c'est-à-dire que les deux chaînes n'ont aucun symbole en commun, alors dans la décomposition $X|^{+}Y$ il ne sera possible que de faire des opérations `insérer`.

(b) $\_|^{+}$: cette décomposition est très proche de celle proposée par Steudel et al. dans [SJS10] (voir section I.2.3.3) et est représentée dans la figure II.2b. En effet, Steudel et al. concatènent les deux chaînes Y et X en insérant un symbole β au milieu puis retranchent la décomposition liée à Y. Ainsi il ne reste plus que $\beta \notin \mathcal{A}$ et la décomposition de X connaissant Y dans son intégralité et le passé de X. La décomposition $X\_|^{+}Y$ est alors la décomposition avec le moins de symboles possible pour expliquer X connaissant Y. En effet si nous nous plaçons à nouveau dans le cas où $\mathcal{A}^X \cap \mathcal{A}^Y = \emptyset$, alors cela revient à décomposer X uniquement grâce à son passé.

(c) $\_|^{-}$: cette décomposition est représentée dans la figure II.2c. La décomposition $|^{+}$ suppose que nous possédons Y dans sa totalité. Cette assertion peut être fausse pour



(a) $X|^{+}Y$.

(b) $X\_|^{+}Y$.

(c) $X\_|^{\text{-}}Y$.

(d) $X\_|^{\text{P}}Y$.

FIGURE II.2 – Représentation de différents conditionnements. Les octets qui peuvent servir de localisation à des opérations `copier` sont encadrés en bleu

deux raisons. Tout d'abord les chaînes peuvent arriver en temps réel sur l'ordinateur. Ainsi au début de la décomposition, Y n'est pas connu dans sa totalité. Deuxièmement les chaînes peuvent dépendre du temps, c'est-à-dire l'octet i est antérieur dans le temps à l'octet $i+1$. Faire une opération `copier` dans le futur de Y pour décomposer le présent de X n'a donc aucun sens physique.

(d) $\_|^{\text{P}}$: cette décomposition est représentée dans la figure II.2d et permet de faire référence à l'octet présent de Y. Il suffit d'inclure l'octet juste après la barre verticale dans la figure II.2c pour avoir les octets qui peuvent servir de référence.

Pour bien comprendre ces différents conditionnements, prenons comme exemple les chaînes X, Y et Z suivantes.

X=`ABCD EFGH IJKL`

Y=`ABCD ABCD ABCD`

Z=`MNOP MNOP MNOP`

Nous allons décomposer Y et Z sachant X avec deux types de conditionnement : $|^{+}$ et $\_|^{+}$. Pour appréhender au mieux comment sont réalisées ces quatre décompositions, nous allons prendre le temps de les détailler chacune comme nous l'avons fait pour `LZ77`, `LZ78` et `GZIP`. Nous nous appuyons sur la figure II.3, qui représente l'étape 2 de production de la décomposition $Y|^{+}X$, pour comprendre chaque étape.

$Y|^+X$: étape 2

$$\begin{array}{lcccccccccccc}
 & & & & & \Downarrow & & & & & & & \\
Y: & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} \\
X: & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{E} & \mathrm{F} & \mathrm{G} & \mathrm{H} & \mathrm{I} & \mathrm{J} & \mathrm{K} & \mathrm{L} \\
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11
\end{array}$$

étape de production $E_3(Y|^+X) = ABCD$

opération `copier` : signal : X, indice : 0, longueur : 4

code `Z`$(1:0 \to 4)$

FIGURE II.3 – Exemple d'une étape d'encodage du processus de production de `SALZA`

- La première chaîne de caractères `A`,`B`,... est la chaîne Y que nous sommes en train de décomposer.
- La deuxième chaîne de caractères `A`,`B`,... est la chaîne X qui est un a priori pour la décomposition de Y.
- La flèche $\Downarrow$ nous indique où nous en sommes dans le processus d'encodage.
- Tous les caractères de couleur bleue sont des endroits où il est possible de commencer une opération `copier`.
- L'étape de production représente les caractères qui sont encodés dans cette étape.
- L'opération est le `copier` ou l'`insérer` réalisé lors de cette étape.
- Le code est le mot généré pour expliquer les caractères de l'étape de production.

Nous allons dans un premier temps utiliser le conditionnement $|^+$.

- **$Y|^+X$: décomposition de Y connaissant X dans son intégralité**

$Y|^+X$: étape 1

$$\begin{array}{lcccccccccccc}
 & \Downarrow & & & & & & & & & & & \\
Y: & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} \\
X: & \mathrm{A} & \mathrm{B} & \mathrm{C} & \mathrm{D} & \mathrm{E} & \mathrm{F} & \mathrm{G} & \mathrm{H} & \mathrm{I} & \mathrm{J} & \mathrm{K} & \mathrm{L} \\
 & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & 11
\end{array}$$

étape de production $E_1(Y|^+X) = ABCD$

opération `copier` : signal : X, indice : 0, longueur : 4

code `Z`$(1:0 \to 4)$

$Y|^+X$: étape 2

					⇓							
Y :	A	B	C	D	A	B	C	D	A	B	C	D
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production $E_2(Y|^+X) = ABCD$ — opération `copier` : signal : X, indice : 0, longueur : 4 — code $\texttt{Z}(1:0\to 4)$

$Y|^+X$: étape 3

									⇓			
Y :	A	B	C	D	A	B	C	D	A	B	C	D
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production $E_3(Y|^+X) = ABCD$ — opération `copier` : signal : X, indice : 0, longueur : 4 — code $\texttt{Z}(1:0\to 4)$

On a donc $Y|^+X \rightsquigarrow \texttt{Z}(1:0\to 4)\ \texttt{Z}(1:0\to 4)\ \texttt{Z}(1:0\to 4)$. L'ensemble des longueurs de cette décomposition est $\mathcal{L}_{Y|^+X} = \{4,4,4\}$.

- **$Z|^+X$: décomposition de Z connaissant X dans son intégralité**

$Z|^+X$: étape 1

	⇓											
Z :	M	N	O	P	M	N	O	P	M	N	O	P
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production $E_1(Z|^+X) = M$ — opération `insérer` : M — code $\texttt{L}(M)$

Il est assez évident qu'on ne peut jamais faire de `copier` depuis X pour décomposer Z, car $\mathcal{A}^*{}_X \cap \mathcal{A}^*{}_Z = \emptyset$. Donc à chaque étape de production, nous avons une opération `insérer`. On a donc $Z|^+X \rightsquigarrow \texttt{L}(M)\ \texttt{L}(N)\ \texttt{L}(O)\ \texttt{L}(P)\ \texttt{L}(M)\ \texttt{L}(N)\ \texttt{L}(O)\ \texttt{L}(P)\ \texttt{L}(M)\ \texttt{L}(N)\ \texttt{L}(O)\ \texttt{L}(P)$. L'ensemble des longueurs de cette décomposition est $\mathcal{L}_{Z|^+X} = \{1,1,1,1,1,1,1,1,1,1,1,1\}$.

Maintenant étudions le cas où nous rajoutons la connaissance de la chaîne en elle-même, c'est à dire $\_|^+$.

- **$Y\_|^+X$: décomposition de Y connaissant son passé et X dans son intégralité**

$Y\_|^+X$: étape 1

	⇓										
Y :	A	B	C	D	A	B	C	D	A	B	C	D
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production $E_1(Y\_|^+X) = ABCD$ — opération `copier` : signal : X, indice : 0, longueur : 4 — code $\mathtt{Z}(1:0 \to 4)$

$Y\_|^+X$: étape 2

					⇓						
Y :	A	B	C	D	A	B	C	D	A	B	C	D
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production $E_2(Y\_|^+X) = ABCDABCD$ — opération `copier` : signal : Y, indice : 0, longueur : 8 — code $\mathtt{Z}(0:0 \to 8)$

On a donc $Y\_|^+X \rightsquigarrow \mathtt{Z}(1:0 \to 4)\mathtt{Z}(0:0 \to 8)$. L'ensemble des longueurs de cette décomposition est $\mathcal{L}_{Y\_|^+X} = \{4, 8\}$.

- **$Z\_|^+X$: décomposition de Z connaissant son passé et X dans son intégralité**

$Z\_|^+X$: étape 1

	⇓										
Z :	M	N	O	P	M	N	O	P	M	N	O	P
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production $E_3(Z\_|^+X) = M$ — opération `insérer` : M — code $\mathtt{L}(M)$

$Z_{\text{-}}|^{+}X$: étape 2

		⇓									
Z :	M	N	O	P	M	N	O	P	M	N	O	P
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production : $E_2(Z_{\text{-}}|^{+}X) = N$ — opération : `insérer` : N — code : $\texttt{L}(N)$

$Z_{\text{-}}|^{+}X$: étape 3

			⇓								
Z :	M	N	O	P	M	N	O	P	M	N	O	P
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production : $E_3(Z_{\text{-}}|^{+}X) = O$ — opération : `insérer` : O — code : $\texttt{L}(O)$

$Z_{\text{-}}|^{+}X$: étape 4

				⇓							
Z :	M	N	O	P	M	N	O	P	M	N	O	P
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production : $E_4(Z_{\text{-}}|^{+}X) = P$ — opération : `insérer` : P — code : $\texttt{L}(P)$

$Z_{\text{-}}|^{+}X$: étape 5

					⇓						
Z :	M	N	O	P	M	N	O	P	M	N	O	P
X :	A	B	C	D	E	F	G	H	I	J	K	L
	0	1	2	3	4	5	6	7	8	9	10	11

étape de production : $E_5(Z_{\text{-}}|^{+}X) = MNOPMNOP$ — opération : `copier` : signal : Z, indice : 0, longueur : 8 — code : $\texttt{Z}(0:0 \to 8)$

On a donc $Z_{\text{-}}|^{+}X \rightsquigarrow \texttt{L}(M)\texttt{L}(N)\texttt{L}(O)\texttt{L}(P)\texttt{Z}(0:0 \to 8)$. L'ensemble des longueurs de cette décomposition est $\mathcal{L}_{Y_{\text{-}}|^{+}X} = \{1, 1, 1, 1, 8\}$.

Ces deux exemples montrent bien que le conditionnement influence beaucoup la décomposition. Ainsi, en plus des deux complexités de Lempel-Ziv conditionnelles vues dans la section I.2.3, il existe donc quatre autres conditionnements possibles. Nous disposons donc de six décompositions différentes et nous pouvons étendre la complexité de Lempel-Ziv de façon conditionnelle à ces six conditionnements. Pour cela, il suffit de compter le nombre de symboles émis par les différentes décompositions.

Propriété II.1.1 (Complexité de Lempel-Ziv conditionnelle)**.**
Soit X et $Y \in \mathcal{A}^*$, alors la complexité de Lempel-Ziv conditionnelle est définie comme suit :

$$L(X \wr Y) = |\mathcal{L}_{X \wr Y}|.$$

où $|\mathcal{L}_{X \wr Y}|$ est le cardinal de l'ensemble $\mathcal{L}_{X \wr Y}$, c'est-à-dire le nombre d'éléments de la décomposition $X \wr Y$.

Ainsi en plus des deux complexités de Lempel-Ziv conditionnelles présentées dans la section I.2.3, il est maintenant possible d'avoir six complexités conditionnelles qui permettent de répondre à différents besoins. Si par exemple, nous voulons faire de la classification comme Ziv et Merhav alors nous pouvons utiliser le conditionnement $|^+$. Par contre si nous voulons faire une mesure d'information, il faut utiliser le conditionnement $\_|^+$ comme l'ont montré Steudel et al. Enfin, nous verrons dans la section II.3.2.2 que pour définir la dépendance directionnelle entre deux chaînes X et Y nous avons besoin d'utiliser les conditionnements $\_|^-$ et $\_|^p$

En résumé

En fonction de la position du premier octet des opérations `copier`, il existe différents conditionnements de la chaîne X par la chaîne Y. Voici les notations présentant les différentes localisations du premier octet des opérations `copier` pour décomposer X sachant Y :

- dans Y :
  - dans le passé de Y $|^-$,
  - dans le passé et le présent de Y $|^p$,
  - dans l'intégralité de Y $|^+$,
- dans X :
  - impossible de faire référence à X $|$,
  - dans le passé de X $\_|$.

Il est possible de combiner la connaissance X et de Y (par exemple $\_|^+$). Pour parler du conditionnement en général parmi les six décompositions qui existent, nous utilisons la notation $X \wr Y$.
La complexité de Lempel-Ziv conditionnelle étendue à tous ces cas de contionnement est le nombre de symboles dans la décomposition considérée.

II.1.3 Les opérations `copier` significatives

La complexité de Lempel-Ziv étendue, uniquement fondée sur le nombre de symboles de la décomposition, ne capture pas toute l'information. Pour cela, prenons un exemple. Nous avons deux chaînes X et Y dont la décomposition $X \wr Y$ est composée de $n = 10000$ symboles de longueur $l = 6$, $\mathcal{L}_{X \wr Y} = \{6, 6, ..., 6\}$ donc $L(X \wr Y) = n$. Prenons deux cas pour comprendre pourquoi la complexité de Lempel-Ziv ne capture pas toute l'information.

- La taille de l'alphabet de X et Y est deux, $\alpha = 2$. Il est très probable dans une chaîne de longueur $n * l = 60000$ de faire de longues opérations `copier`. Lempel et Ziv ont montré (I.2.2 et I.2.3) que la longueur moyenne des opérations `copier` d'une chaîne aléatoire de longueur $n * l = 60000$ sur un alphabet de taille 2 est 15 (pour plus de détails voir la définition II.1.3). Donc dans ce cas Y n'apporte aucune information à X. En effet si X ne connaissait pas Y, la décomposition aurait comme longueur moyenne des opérations `copier` 15 au lieu de 6.
- Prenons cette fois, l'alphabet de X et Y de taille 256, $\alpha = 256$. Il est peu probable dans une chaîne de longueur $n * l = 60000$ de faire de longues opérations `copier`. En effet, comme l'ont montré Lempel et Ziv (I.2.2 et I.2.3), la longueur moyenne des opérations pour une chaîne aléatoire est 2 (pour plus de détails voir la définition II.1.3). Ici, la longueur des opérations est bien plus grande que pour une chaîne aléatoire, donc Y apporte beaucoup d'information à X.

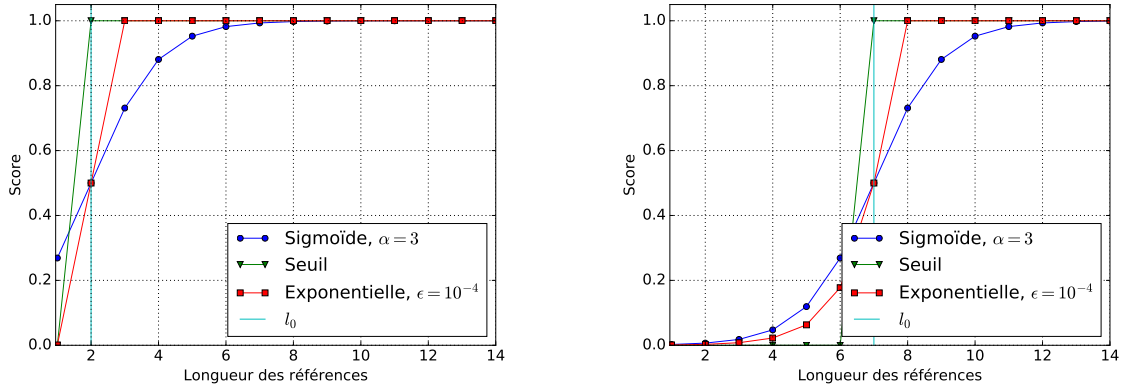
Dans ces deux cas, le nombre de symboles est identique : la complexité de Lempel-Ziv ne fait donc pas de différence. Cependant dans le deuxième cas Y apporte beaucoup d'information à X et dans le premier c'est le contraire. Il est donc nécessaire de pouvoir différencier ces deux cas.

Pour pallier ce problème, nous proposons de prendre également en compte la longueur des symboles obtenus lors de la décomposition $X \wr Y$ et pas uniquement leur nombre. Ainsi si la longueur de l'opération `copier` est grande alors cette longueur aura un poids fort. Par contre si cette longueur est petite alors nous lui associons un poids faible. Pour attribuer un poids à chaque longueur, nous définissons une fonction admissible comme suit.

Définition II.1.1 (Fonction admissible)**.**
Soient une chaîne $X \in \mathcal{A}^*$ de longueur $l(X)$ et un seuil $T \in [0, l(X)]$, alors $f : \mathbb{N}^* \to [0, 1]$ est une fonction admissible si et seulement si

1. *f est croissante monotone,*
2. *$\forall l \in \mathbb{N}^*$ tel que $l \geq T \quad f(l) = 1$.*

La fonction admissible nous permet de quantifier si une opération `copier` d'une décomposition $X \wr Y$ permet de significativement bien expliquer X par Y. En effet, si le poids de la longueur de l'opération `copier` est supérieur à 0,5 alors nous considérons que cette opération `copier` permet de significativement bien expliquer X par Y. Grâce à la fonction admissible nous pouvons donc quantifier si une longueur est significative ou non.



(a) $l^\star = 2$. (b) $l^\star = 7$.

FIGURE II.4 – Différentes fonctions admissibles avec différent $l^\star$.

Définition II.1.2 (Opération `copier` significative)**.**
Soient une chaîne X et $Y \in \mathcal{A}^*$ de longueur $l(X)$, $l \in \mathcal{L}_{X\wr Y}$ et une fonction admissible f avec un seuil T, alors on dit qu'une opération ***`copier`*** *de longueur l est significative si $f(l) > 0{,}5$.*

Au lieu de définir le seuil T à partir duquel une fonction admissible vaut 1, il est souvent préférable de définir une longueur $l^\star$ à partir de laquelle les longueurs sont considérées comme significatives. C'est-à-dire que si $l > l^\star$ alors l'opération `copier` associée est une opération `copier` significative.

Voici donc quatre exemples de fonctions admissibles définies grâce à $l^\star$. Ces fonctions sont représentées dans la figure II.4.

fonction indicatrice : $\mathbb{1} : l \rightarrow 1$: cette fonction donne le même poids à toutes les longueurs, on ne fait donc aucune différence entre les petites et les grandes longueurs.

fonction seuil : $\mathbb{T} : l \rightarrow \begin{cases} 0 & \text{si } l < l^\star \\ 1 & \text{sinon} \end{cases}$: cette fonction donne un poids nul à toutes les longueurs plus petites qu'un certain seuil $l^\star$. Toutes ces longueurs sont donc considérées comme non significatives.

fonction sigmoïde : $\mathbb{S} : l \rightarrow \begin{cases} \frac{1}{1+e^{-l+l^\star}} & \text{si } l < \beta l^\star \\ 1 & \text{sinon} \end{cases}$: cette fonction donne un poids inférieur à 0,5 à toutes les longueurs plus petites que $l^\star$. La dérivée est correctement définie autour de $l^\star$ et permet de pondérer chaque longueur. β permet quant à lui de respecter la deuxième condition des fonctions admissibles. En pratique une valeur de $\beta = 3$ est largement suffisante.

fonction exponentielle : $\mathbb{E} : l \rightarrow \begin{cases} exp(ln\ \epsilon - (l-1)\frac{ln\ 2 + Ln\ \epsilon}{l^\star - 1}) & \text{si } l < \frac{ln\ 2 + l^\star ln\ 2}{ln\ 2 + ln\ \epsilon} \\ 1 & \text{sinon} \end{cases}$: tout comme la fonction sigmoïde, la fonction exponentielle permet de faire une transition moins abrupte que la fonction seuil. Le ϵ est très faible car la fonction ln n'est pas définie en 0.

La longueur seuil $l^\star$ peut être fixée manuellement par l'utilisateur s'il a un a priori

sur les données. Cependant dans la majorité des cas, nous voulons que $l^\star$ soit calculée automatiquement sans intervention extérieure.

Définition II.1.3 (l_0)**.**
Soit une chaîne $X \in \mathcal{A}^*$ de longueur $l(X)$, α étant la taille de l'alphabet $\mathcal{A}$, alors la longueur seuil automatique est :

$$l_0 = log_\alpha l(X).$$

Pour trouver cette valeur, nous utilisons le théorème I.2.2. Il montre que pour une chaîne très grande le nombre de symboles de la décomposition est inférieur à $\frac{l(X)}{log_\alpha l(X)}$. Le théorème I.2.3 affirme que pour une chaîne aléatoire très grande, le nombre de symboles est asymptotiquement égal à $\frac{l(X)}{log_\alpha l(X)}$. Ainsi, en moyenne, les symboles pour une chaîne aléatoire très grande sont de longueur $l_0 = log_\alpha l(X)$. Donc si la longueur de l'opération `copier` est plus petite que l_0 alors la longueur est peu significative. Par contre si la longueur de l'opération `copier` est supérieure à l_0 cela signifie que dans la décomposition $X \wr Y$, Y apporte de l'information à X via cette "longue" sous-chaîne identique.

Ainsi dans une décomposition $X \wr Y$, nous pouvons maintenant associer chaque longueur de la décomposition à un poids par une fonction admissible f. Ce poids quantifie si l'opération `copier` est significative ou non. Maintenant que nous savons dire si une opération `copier` est significative, nous voulons quantifier à quel point toute la décomposition de X connaissant Y est significative. Nous définissons donc une mesure qui représente la quantité de caractères de X bien expliquée par Y.

Définition II.1.4 (*Quantité significative*)**.**
Soient deux chaînes X et $Y \in \mathcal{A}^*$ et une fonction admissible f avec une longueur seuil $l^\star$, alors la quantité de X significativement bien expliquée par Y est :

$$W_f(X \wr Y) = \sum_{l \in \mathcal{L}_{X \wr Y}} l f(l). \tag{II.1}$$

W est le nombre pondéré de caractères de X qui a été correctement expliqué grâce à Y dans la décomposition $X \wr Y$. Il est facile d'obtenir les bornes de la *quantité significative*.

Propriété II.1.2 (Bornes de la *quantité significative*)**.**
Soient deux chaînes Y et $X \in \mathcal{A}^*$ et une fonction admissible f centrée en $l^\star$, alors

$$0 \leq W_f(X \wr Y) \leq l(X).$$

La démonstration est présentée dans l'annexe A. Quand la *quantité significative* est proche de sa borne supérieure, alors nous dirons que X est bien expliquée par Y. À l'inverse si la *quantité significative* est proche de 0, alors nous dirons que X est mal expliquée par Y.

Pour bien comprendre ce que cette quantité représente reprenons les exemples vus précédemment et prenons par exemple la fonction admissible seuil $\mathbb{T}$ avec $l^\star = 2$.

X=ABCD EFGH IJKL

Y=ABCD ABCD ABCD

Z=MNOP MNOP MNOP

- $Y|^+X$
  - · Décomposition : $Y|^+X \rightsquigarrow \mathtt{Z}(1:0\to 4)\ \mathtt{Z}(1:0\to 4)\ \mathtt{Z}(1:0\to 4)$.
  - · Ensemble des longueurs : $\mathcal{L}_{Y|^+X} = \{4,4,4\}$.
  - · *Quantité significative* avec la fonction admissible seuil $\mathbb{T}$: $W_{\mathbb{T}}(Y|^+X) = 4\mathbb{T}(4)+ 4\mathbb{T}(4) + 4\mathbb{T}(4) = 12$.
  - · Explication : 12 caractères de Y sont bien expliqués par X, ce qui est le maximum possible donc Y est très bien expliquée par X.
- $Z|^+X$
  - · Décomposition : $Z|^+X \rightsquigarrow \mathtt{L}(M)\ \mathtt{L}(N)\ \mathtt{L}(O)\ \mathtt{L}(P)\ \mathtt{L}(M)\ ...\ \mathtt{L}(P)$.
  - · Ensemble des longueurs :$\mathcal{L}_{Z|^+X} = \{1,1,1,1,1,1,1,1,1,1,1,1\}$.
  - · *Quantité significative* avec la fonction admissible seuil $\mathbb{T}$: $W_{\mathbb{T}}(Z|^+X) = 1\mathbb{T}(1)+ 1\mathbb{T}(1) + ... + 1\mathbb{T}(1) = 0$.
  - · Explication : 0 caractères de Z sont bien expliqués par X, ce qui est minimal donc Z est très mal expliquée par X.
- $Y_{\text{-}}|^+X$
  - · Décomposition : $Y_{\text{-}}|^+X \rightsquigarrow \mathtt{Z}(1:0\to 4)\mathtt{Z}(0:0\to 8)$.
  - · Ensemble des longueurs : $\mathcal{L}_{Y_{\text{-}}|^+X} = \{4,8\}$.
  - · *Quantité significative* avec la fonction admissible seuil $\mathbb{T}$: $W_{\mathbb{T}}(Y_{\text{-}}|^+X) = 4\mathbb{T}(4)+ 8\mathbb{T}(8) = 12$.
  - · Explication : 12 caractères de Y sont bien expliqués par eux-mêmes et X, c'est maximal donc Y est très bien expliquée par elle-même et X. Dans la décomposition $Y_{\text{-}}|^+X$, il y a un symbole en moins que dans la décomposition $Y|^+X$. Cependant la *quantité significative* ne fait pas de différence entre ces deux cas.
- $Z_{\text{-}}|^+X$
  - · Décomposition : $Z_{\text{-}}|^+X \rightsquigarrow \mathtt{L}(M)\mathtt{L}(N)\mathtt{L}(O)\mathtt{L}(P)\mathtt{Z}(0:0\to 8)$.
  - · Ensemble des longueurs : $\mathcal{L}_{Y_{\text{-}}|^+X} = \{1,1,1,1,8\}$.
  - · *Quantité significative* avec la fonction admissible seuil $\mathbb{T}$: $W_{\mathbb{T}}(Y_{\text{-}}|^+X) = 1\mathbb{T}(1)+ 1\mathbb{T}(1) + 1\mathbb{T}(1) + 1\mathbb{T}(1) + 8\mathbb{T}(8) = 8$.
  - · Explication : 8 caractères de Z sont bien expliqués par eux-mêmes et X, Z est donc assez bien expliquée par elle-même et X. Comme nous avons vu que Z est très mal expliquée par X et uniquement X, nous pouvons en déduire que dans ce cas là Z s'explique uniquement grâce à son passé.

En résumé

Prendre en compte uniquement le nombre de symboles dans la décomposition de $X \wr Y$ n'est pas exhaustif. C'est pour cela que nous avons introduit de nouvelles quantités.

- *Longueur seuil, $l^\star$* : cette longueur est le seuil à partir de la quelle l'opération `copier` est *significative*.
- *Fonction admissible, f* : cette fonction, centrée sur $l^\star$, associe à chaque longueur un poids. Plus la longueur est significative plus le poids est proche de 1.
- *Quantité significative, W_f* : c'est nombre de caractères de X bien expliqués par Y. Elle est basée sur la fonction admissible f.

II.2 SALZA : mesurer la ressemblance entre deux chaînes

Basé sur la décomposition de `GZIP`, nous avons créé un nouvel estimateur de complexité appelé `SALZA`. Cet estimateur est construit à partir de la complexité de Lempel-Ziv conditionnelle étendue et de la *quantité significative*. Cette complexité est une mesure d'information et il est donc possible de construire une théorie algorithmique de l'information basée sur `SALZA`.

II.2.1 Construction de SALZA

Pour construire l'estimateur `SALZA`, nous nous appuyons sur la complexité de Lempel-Ziv et la *quantité significative*. En effet, comme nous l'avons évoqué dans la section précédente la complexité de Lempel-Ziv ne révèle pas toute l'information dans une décomposition, mais la *quantité significative* seule ne suffit pas non plus. Par exemple si une première décomposition de X sachant Y donne k opérations `copier` de taille n avec $f(n) = 1$, alors $W_f = l(X)$. Cependant si une autre décomposition donne une unique opération `copier` de taille kn, alors on a aussi $W_f = l(X)$. Or nous voudrions pouvoir faire une différence entre ces deux cas. C'est pourquoi nous avons défini notre estimateur de complexité `SALZA` comme suit.

Définition II.2.1 (`SALZA`).
Soient X et Y deux chaînes $\in \mathcal{A}^*$ et la fonction admissible f centrée en $l^\star$, alors l'estimateur de complexité `SALZA` est défini comme suit :

$$S_f(X \wr Y) = l(X) - 1 - \sum_{l \in \mathcal{L}_{X \wr Y}} (l-1) f(l). \tag{II.2}$$

Les bornes de `SALZA` sont :

Propriété II.2.1 (Bornes de SALZA)**.**
Soient deux chaînes Y et $X \in \mathcal{A}^*$ et une fonction admissible f centrée en $l^\star$, alors l'estimateur de complexité ***SALZA****, S_f, satisfait :*

$$0 \leq S_f(X \wr Y) \leq l(X) - 1$$

La preuve se trouve dans l'annexe A. Quand SALZA est proche de sa borne supérieure, alors nous dirons que X est très complexe selon le conditionnement $X \wr Y$. À l'inverse si SALZA est proche de zéro, alors nous dirons que X est peu complexe selon le conditionnement $X \wr Y$.

Reprenons l'exemple des trois chaînes X, Y et Z ainsi que la fonction admissible sigmoïde $ avec $l^\star = 2$. Le choix de la fonction admissible optimale sera détaillé dans la section II.4.1.

Y=ABCD ABCD ABCD

Z=MNOP MNOP MNOP

- $Y|^+X$
  - Décomposition : $Y|^+X \rightsquigarrow \mathtt{Z}(1:0 \to 4)\ \mathtt{Z}(1:0 \to 4)\ \mathtt{Z}(1:0 \to 4)$.
  - Ensemble des longueurs : $\mathcal{L}_{Y|^+X} = \{4, 4, 4\}$.
  - SALZA avec la fonction admissible sigmoïde $: $S_\$(Y|^+X) = 12 - [(4-1)\$(4) + (4-1)\$(4) + (4-1)\$(4)] - 1 = 3,1$.
  - Explication : Y est relativement peu complexe en connaissant X et uniquement X.
- $Z|^+X$
  - Décomposition : $Z|^+X \rightsquigarrow \mathtt{L}(M)\ \mathtt{L}(N)\ \mathtt{L}(O)\ \mathtt{L}(P)\ \mathtt{L}(M)\ ...\ \mathtt{L}(P)$.
  - Ensemble des longueurs :$\mathcal{L}_{Z|^+X} = \{1,1,1,1,1,1,1,1,1,1,1,1,1\}$.
  - SALZA avec la fonction admissible sigmoïde $: $S_\$(Z|^+X) = 12 - [(1-1)\$(1) + (1-1)\$(1) + ... + (1-1)\$(1)] - 1 = 11$.
  - Explication : la complexité de Z sachant X est maximale donc Z est très complexe en connaissant X et uniquement X.
- $Y_\text{-}|^+X$
  - Décomposition : $Y_\text{-}|^+X \rightsquigarrow \mathtt{Z}(1:0 \to 4)\mathtt{Z}(0:0 \to 8)$.
  - Ensemble des longueurs : $\mathcal{L}_{Y_\text{-}|^+X} = \{4, 8\}$.
  - SALZA avec la fonction admissible sigmoïde $: $S_\$(Y_\text{-}|^+X) = 12 - [(4-1)\$(4) + (8-1)\$(8)] - 1 = 1,4$.
  - Explication : si on rajoute le passé de Y comme a priori, Y est encore moins complexe qu'en connaissant uniquement X. Nous voyons ici l'intérêt de rajouter le nombre de symboles à la *quantité significative*. En effet lorsque nous avions rajouté la connaissance du passé de Y la *quantité significative* n'avait pas évolué contrairement à SALZA. Or nous voyons bien que le passé de Y permet de mieux expliquer Y. Il est donc logique que SALZA diminue.

- $Z_{\text{-}}|^{+}X$
  - Décomposition : $Z_{\text{-}}|^{+}X \rightsquigarrow \mathtt{L}(M)\mathtt{L}(N)\mathtt{L}(O)\mathtt{L}(P)\mathtt{Z}(0:0\to 8)$.
  - Ensemble des longueurs : $\mathcal{L}_{Y_{\text{-}}|^{+}X} = \{1,1,1,1,8\}$.
  - SALZA avec la fonction admissible sigmoïde $\mathbb{S}$: $S_{\mathbb{S}}(Y_{\text{-}}|^{+}X) = 12-[(1-1)\mathbb{S}(1)+(1-1)\mathbb{S}(1)+(1-1)\mathbb{S}(1)+1\mathbb{S}(1)+(8-1)\mathbb{S}(8)]-1 = 4,0$.
  - Explication : si on permet de faire des références dans le passé de Z, finalement Z est assez peu complexe en connaissant X mais reste bien plus complexe que Y connaissant X car X et Z n'ont rien en commun.

Maintenant que nous avons un estimateur SALZA pour la complexité conditionnelle, nous pouvons introduire la complexité simple basée sur SALZA comme avec la complexité de Kolmogorov (voir théorème I.1.3).

Définition II.2.2 (SALZA simple)**.**
La complexité SALZA simple de la chaîne $X \in \mathcal{A}^*$ selon la fonction admissible f dénoté $S_f(X)$ est donnée par :

$$S_f(X) = S_f(X_{\text{-}}|^{+}\Lambda).$$

En résumé

Nous proposons une nouvelle mesure de complexité conditionnelle combinant le nombre de symboles de la décomposition $X \wr Y$ et la *quantité significative*.

$$S_f(X \wr Y) = l(X) - \sum_{l\in\mathcal{L}_{X\wr Y}} (l-1)f(l).$$

Il est possible de définir SALZA simple comme étant :

$$S_f(X) = S_f(X_{\text{-}}|^{+}\Lambda).$$

II.2.2 Ajout de connaissance

Il est nécessaire d'observer comment réagit SALZA quand on rajoute de la connaissance, c'est-à-dire comparer $S_f(X \wr Y, Z)$ et $S_f(X \wr Y)$. Intuitivement, nous voudrions que la complexité diminue quand nous rajoutons de la connaissance. C'est le cas pour la fonction admissible indicatrice $\mathbb{1}$.

Théorème II.2.1 (Ajout de connaissance)**.**
Soient X, Y et $Z \in \mathcal{A}^*$ et l'estimateur de complexité SALZA avec la fonction admissible indicatrice $\mathbb{1}$. L'estimateur $S_{\mathbb{1}}$ satisfait alors :

$$S_{\mathbb{1}}(X \wr Y, Z) \leq S_{\mathbb{1}}(X \wr Y).$$

Si nous rajoutons de la connaissance a priori pour décomposer X, alors forcément nous ferons moins d'opérations `copier` et `insérer` que sans cette information. La démonstration est présentée dans l'annexe A.

Il est cependant possible de trouver des contre-exemples pour les autres fonctions admissibles proposées dans la section II.1.3. Prenons par exemple les chaînes suivantes, où $(x_1..x_k)^n$ signifie que la sous-chaîne $x_1...x_k$ est répétée n fois :

X=(`ABCD EFGH`)^n

Y=`ABCD X EFGH`

Z=`ABCD EFXX`

Nous allons comparer les deux conditionnements $X|^+Y$ et $X|^+Y,Z$ pour voir l'influence de l'ajout de la connaissance de Z. Les deux décompositions obtenues sont :

$$X|^+Y \rightsquigarrow \underbrace{\mathtt{Z}(1:0\to 4)\ \mathtt{Z}(1:5\to 4)...\mathtt{Z}(1:0\to 4)\ \mathtt{Z}(1:5\to 4)}_{n \text{ fois}}$$

$$X|^+Y,Z \rightsquigarrow \underbrace{\mathtt{Z}(2:0\to 6)\ \mathtt{Z}(1:7\to 2)...\mathtt{Z}(2:0\to 6)\ \mathtt{Z}(1:7\to 2)}_{n \text{ fois}}.$$

Pour calculer `SALZA`, nous devons choisir une fonction admissible avec une longueur seuil $l^\star$. Nous choisissons la fonction admissible seuil $\mathbb{T}$ avec $l^\star$=3. Cependant le contre-exemple qui va suivre est valable pour toutes les fonctions admissibles présentées dans la section II.1.3 avec $l^\star = 3$.

$$S_{\mathbb{T}}(X|^+Y) = 8n - (3n\mathbb{T}(4) + 3n\mathbb{T}(4)) = 2n$$

$$S_{\mathbb{T}}(X|^+Y,Z) = 8n - (5n\mathbb{T}(6) + n\mathbb{T}(2)) = 3n.$$

Nous avons donc $S_{\mathbb{T}}(X|^+Y) = 2n < 3n = S_{\mathbb{T}}(X|^+Y,Z)$. Dans ce cas-là, X est plus complexe quand nous connaissons Y et Z qu'en connaissant uniquement Y. Si nous avions pris la fonction admissible indicatrice, nous aurions bien $S_{\mathbb{1}}(X|^+Y) = 8n - 3n - 3n = 2n = 8n - 5n - n = S_{\mathbb{1}}(X|^+Y,Z)$.

En pratique il est extrêmement rare que cette anomalie se produise. Nous utilisons la fonction admissible exponentielle pour montrer qu'en pratique l'ajout de connaissance fait presque toujours diminuer la complexité `SALZA`. En effet, nous verrons dans la section II.4.1 que c'est la fonction admissible exponentielle qui a le meilleur pouvoir discriminant autour de $l^\star$ parmi celles que nous avons présentées (section II.1.3).

Pour observer l'influence de l'ajout de connaissance sur `SALZA`, nous prenons l'ensemble des jeux de données suivants [1] : des chaînes i.i.d. d'une loi uniforme discrète avec différentes tailles d'alphabet, des chaînes de Markov avec différentes tailles d'alphabet, de l'ADN et le texte de la Déclaration des Droits de l'Homme dans différentes langues. Pour chacune des chaînes de ces jeux de données nous regardons l'influence de l'ajout de la connaissance des autres chaînes du jeu de données. Voici les calculs que nous effectuons pour chaque chaîne X et Y du jeu de données :

Complexité simple : $S_{\mathbb{E}}(X_\text{-}|^+\Lambda) = S_{\mathbb{E}}(X)$.

Complexité conditionnelle : $S_{\mathbb{E}}(X_\text{-}|^+\Lambda,Y) = S_{\mathbb{E}}(X_\text{-}|^+Y)$.

Ajout de connaissance : $S_{\mathbb{E}}(X) - S_{\mathbb{E}}(X_\text{-}|^+Y)$. Cette différence, qui doit être positive, permet d'évaluer ce qu'apporte la connaissance de Y à la connaissance du passé de X.

1. Pour une description détaillée des données voir l'annexe D.

Normalisation : $\frac{S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda)-S_{\mathbb{E}}(X_{\text{-}}|^{+}Y)}{S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda)}$, pour pouvoir comparer la complexité de chaînes de taille différente, il faut normaliser l'ajout de connaissance par la borne supérieure.

Les résultats sont synthétisés sur la figure II.5. L'abscisse représente le jeu de données dont la chaîne X est tirée et l'axe des ordonnées représente celui dont est issue la chaîne Y. Nous représentons, dans la figure II.5, le minimum d'ajout de connaissance pour tous les couples de chaînes prises dans les jeux de données. La variance et la moyenne pour chaque jeu de données sont présentées dans le tableau B.4 présent dans l'annexe B.

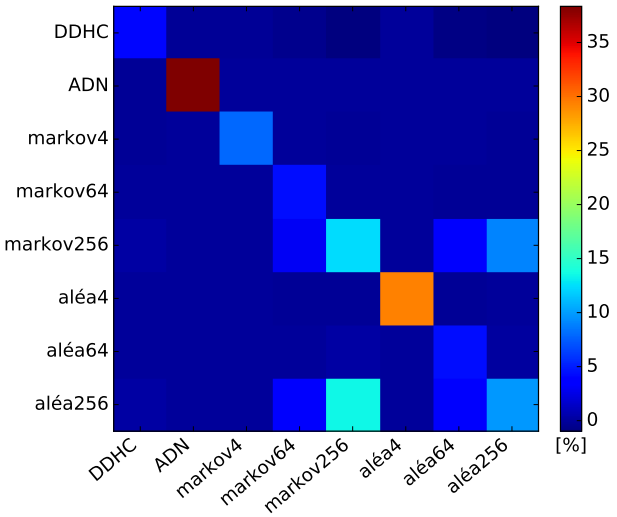


FIGURE II.5 – Influence de l'ajout de connaissance : calcul de $\frac{S_{\mathbb{E}}(X)-S_{\mathbb{E}}(X_{\text{-}}|^{+}Y)}{S_{\mathbb{E}}(X)}$ pour X et Y appartenant aux différents jeux de données, représentation des minima pour l'ensemble des chaînes X du jeu de données en abscisse, connaissant chaque chaîne Y du jeu de données en ordonnée. Si la couleur est bleue alors l'ajout de la connaissance Y fait augmenter la complexité de X, si la couleur est rouge alors l'ajout de la connaissance Y fait diminuer la complexité de X .

Comme nous pouvons le voir dans la figure II.5, si nous comparons des données du même type, alors l'ajout d'une connaissance permet toujours de faire diminuer la complexité. Cependant, lorsque nous comparons des données qui n'ont rien à voir entre elles alors de temps à autre il se peut que l'ajout d'une nouvelle connaissance fasse augmenter légèrement la complexité. Cet ajout de complexité est maximal lorsqu'on décompose la Déclaration des Droits de l'Homme connaissant des chaînes de Markov sur un alphabet de taille 256. Cependant cette erreur ne dépasse pas 0,97% et nous pouvons considérer qu'elle est négligeable (voir section I.3.2). Ainsi, même si le théorème II.2.1 n'est pas vrai pour les fonctions admissibles autres que la fonction indicatrice, nous faisons l'hypothèse qu'empiriquement l'ajout de connaissance fait toujours diminuer la complexité SALZA pour toutes les fonctions admissibles.

En résumé

Nous proposons un nouvel estimateur de complexité que nous appellons SALZA, noté $S_f(X \wr Y)$.

- *Principe* : SALZA est la combinaison entre le nombre de symboles de la décomposition $X \wr Y$ et la *quantité significative*.
- *Avantage* : SALZA prend en compte la longueur des symboles.
- *Inconvénient* : si nous rajoutons de la connaissance a priori, SALZA peut augmenter mais en pratique cela n'est pas observé.

II.2.3 SALZA est (presque) une mesure d'information

Comme nous l'avons vu dans la section I.2.2, si une mesure R est une mesure d'information alors cette mesure peut être la base d'une théorie de l'information. C'est pourquoi nous allons par la suite montrer que SALZA est presque une mesure d'information.

Il faut dans un premier temps définir le réseau fini $(\Omega, \wedge, \vee)$. Nous utilisons la même définition que Steudel et al. dans [SJS10] pour prouver que la complexité de Lempel-Ziv est une mesure d'information.

- Ω est l'ensemble $\mathcal{A}^*$ des chaînes définies sur l'alphabet $\mathcal{A}$.
- L'élément nul de Ω est la chaîne vide Λ.
- L'intersection de deux chaînes X et $Y \in \mathcal{A}^*$ est la concaténation des deux chaînes avec un symbole $\beta \notin \mathcal{A}$ entre les deux : $X \wedge Y = X\beta Y$. En fait, $S_{\mathbb{1}}(X\beta Y) = S_{\mathbb{1}}(X) + S_{\mathbb{1}}(Y_{\text{-}}|^{+}X) + 1$. Le caractère β est là pour empêcher qu'une opération `copier` ne chevauche X et Y. Nous posons donc que $S_f(X \wedge Y)$ est égale à $S_f(X_{\text{-}}|^{+}Y)$.
- La relation d'ordre $X \leq Y$ entre deux chaînes X et $Y \in \mathcal{A}^*$ signifie que X est un préfixe de Y, c'est-à-dire qu'il existe une chaîne $Z \in \mathcal{A}^*$ telle que $Y = XZ$.

Théorème II.2.2 (Mesure d'information)**.**
Soit $\mathcal{A}$ un alphabet, $\mathcal{A}^*$ l'ensemble des chaînes définies sur $\mathcal{A}^*$ et f une fonction admissible. Alors SALZA est une mesure d'information pour la fonction admissible indicatrice $\mathbb{1}$ et vérifie les trois propriétés suivantes :

1. *normalisation :* $S_f(\Lambda) = 0$,
2. *monotonie :* $\forall X, Y \in \mathcal{A}^*, X \leq Y \Rightarrow S_{\mathbb{1}}(X) \leq S_{\mathbb{1}}(Y)$,
3. *sous-modularité :* $\forall X, Y \in \mathcal{A}^*, S_{\mathbb{1}}(X) + S_{\mathbb{1}}(Y) \geq S_{\mathbb{1}}(X) + S_{\mathbb{1}}(Y_{\text{-}}|^{+}X)$.

La démonstration se trouve dans l'annexe A.

Si SALZA n'est pas une mesure d'information pour toutes les fonctions admissibles, c'est parce que le théorème II.2.1 n'est pas respecté pour toutes les fonctions admissibles. Or nous avons vu qu'empiriquement sur tous nos jeux de données, le théorème II.2.1 n'est que rarement contredit (voir figure II.5). Nous considérons donc qu'empiriquement SALZA est une mesure d'information pour toutes les fonctions admissibles.

En résumé

SALZA est une mesure d'information pour la fonction admissible indicatrice. Cependant, empiriquement, SALZA respecte toujours les propriétés d'une mesure d'information pour toutes les fonctions admissibles (voir figure II.5). Nous considérons dans la suite que SALZA est une mesure d'information avec le conditionnement $\_|^+$.

II.3 SALZA et la théorie algorithmique de l'information

Comme SALZA est une mesure d'information, SALZA peut être utilisée pour définir une théorie algorithmique de l'information. Il faut dans un premier temps définir SALZA jointe qui est la complexité des deux chaînes X et Y. Ce qui nous permet dans un second temps de définir l'information mutuelle algorithmique entre deux chaînes qui mesure la dépendance alogrithmique entre elles.

II.3.1 SALZA jointe

L'information mutuelle algorithmique étant basée sur la complexité jointe, il faut donc la définir avant de pouvoir poser l'information mutuelle algorithmique. Pour rappel, la complexité de Kolmogorov jointe de X et Y est la taille du plus petit programme capable de générer X et Y et de les séparer. Cependant rien n'est dit sur comment encoder X et Y (voir la fonction $< \cdot, \cdot >$ dans la section I.1.3). Par exemple est-ce qu'il faut encoder X et Y simultanément ou l'un après l'autre ? Nous proposons donc trois méthodes différentes pour calculer la complexité SALZA jointe, qui reposent sur trois encodages différents de X et Y. Pour que ces mesures soient acceptables, il faut qu'elles respectent les propriétés suivantes.

Propriété II.3.1 (SALZA jointe)**.**
L'encodage de deux chaînes X et $Y \in \mathcal{A}^*$, (X, Y), permet de calculer une complexité jointe notée $S_f(X, Y)$ avec f une fonction admissible, si SALZA respecte les propriétés ci-après :

1. *Positivité : $S_f(X, Y) \geq 0$,*
2. *Symétrie : $S_f(X, Y) = S_f(Y, X)$,*
3. *Invariance : $S_f(X, X) = S_f(X)$,*
4. *Borne inférieure : $S_f(X, Y) \geq max\{S_f(X), S_f(Y)\}$,*
5. *Borne supérieure : $S_f(X, Y) \leq S_f(X) + S_f(Y)$.*

II.3.1.1 Encodage cardinal

La première complexité jointe, que nous introduisons, repose sur l'approche proposée par Zozor et al. dans [ZRB05] et est présentée dans la section I.2.3.2.

Définition II.3.1 (Encodage cardinal)**.**
Soit X, respectivement Y, une chaîne définie sur l'alphabet $\mathcal{A}_X$ de taille α_X, respectivement $\mathcal{A}_Y$ et α_Y, de longueur $l(X)$, respectivement $l(Y)$. Alors la chaîne $Z = z_1...z_n$ est l'encodage cardinal de $X = x_1...x_n$ et $Y = y_1...y_n$ si

$$\forall i = 1..n \quad z_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}.$$

où $n = max\{l(X), l(Y)\}$ et si $i > l(X)$, respectivement $i > l(Y)$ alors le symbole x_i, respectivement y_i, est le symbole vide.

Nous allons tout d'abord illustrer l'encodage cardinal à travers un exemple. Prenons les deux chaînes $X = 10010101$ et $Y = 11010001$. Alors l'encodage cardinal de X et Y est

$$Z = \binom{1}{1}\binom{0}{1}\binom{0}{0}\binom{1}{1}\binom{0}{0}\binom{1}{0}\binom{0}{0}\binom{1}{1}.$$

La taille de l'alphabet de Z est

$$\alpha_Z = \begin{cases} \alpha_X\ \alpha_Y & \text{si } l(X) = l(Y) \\ (\alpha_X + 1)\alpha_Y & \text{si } l(X) < l(Y) \\ \alpha_X(\alpha_Y + 1) & \text{si } l(X) > l(Y) \end{cases}$$

Pour l'exemple précédent, la taille de l'alphabet de Z est donc de 4.

Avec cette approche on peut donc définir une complexité jointe basée sur cet encodage cardinal. En effet, faire la décomposition de la chaîne Z revient à faire la décomposition jointe de X et Y.

Définition II.3.2 (SALZA jointe cardinale)**.**
*Soit Z une chaîne qui est l'encodage cardinal des deux chaînes X et Y. Alors **SALZA** jointe basée sur l'encodage cardinal est appelée **SALZA** jointe cardinale et est définie comme*

$$S_f(X, Y)^{|c} = S_f(Z).$$

Pour la fonction admissible indicatrice $\mathbb{1}$, Zozor et al. ont montré que SALZA jointe cardinale respectait les propriétés de positivité, de symétrie, d'invariance et de supériorité. Il est facile de montrer pour toutes les autres fonctions admissibles, SALZA respecte ces quatre propriétés (voir annexe A). Cependant la borne supérieure de SALZA jointe cardinale n'est vraie qu'à 1 près : $S_f(X, Y) \leq S_f(X) + S_f(Y) + 1$.

Rappelons que la taille de l'alphabet sur SALZA est restreinte à 256. Il ne faut pas que la taille de l'alphabet de Z soit supérieure à 256, $\alpha_Z \leq 256$. Si X et Y font la même taille, alors $\alpha_Z = \alpha_X\ \alpha_Y$. De plus si les tailles de l'alphabet de X et Y sont identiques, alors pour que $\alpha_Z \leq 256$, il faut que $\alpha_X = \alpha_Y \leq 16$. Cette encodage est donc très restrictif sur la taille de l'alphabet des chaînes X et Y.

II.3.1.2 Encodage concaténé

Nous venons de décrire une première complexité jointe basée sur l'encodage cardinal. Une autre possibilité pour estimer la complexité jointe est l'approche proposée par Steudel et al. dans [SJS10] et présentée dans la section I.2.3.3. Elle repose sur le théorème I.1.6 de complexité de Kolmogorov qui est $K(X,Y) = K(X) + K(Y|X) + \mathcal{O}(logK(X,Y))$. Ainsi expliquer simultanément X et Y avec un encodage cardinal revient à expliquer d'abord X puis à expliquer Y connaissant X. Nous avons vu dans la section I.3.2 que les compresseurs utilisaient cette même méthode pour estimer la complexité jointe. Il est donc possible de définir SALZA jointe basée sur cette méthode appelée encodage concaténé.

Définition II.3.3 (SALZA jointe concaténée)**.**
*Soient deux chaînes X et Y, alors **SALZA** jointe basée sur l'encodage concaténé est appelée **SALZA** jointe concaténée et est définie comme*

$$S_f(X,Y)^{|u} = S_f(X) + S_f(Y\_|^{+}X). \tag{II.3}$$

On décompose d'abord X puis Y connaissant son passé et l'intégralité de X.

Lorsque nous mesurons la complexité jointe de X et Y avec SALZA jointe concaténée, nous perdons la propriété de symétrie. En effet, il n'est pas équivalent d'encoder d'abord X puis Y ou de faire l'inverse. Nous allons illustrer cette asymétrie à travers l'exemple suivant.

- $X = 1000110101$,
- $Y = 110010$,
- $X \rightsquigarrow \mathtt{L}(1)\mathtt{L}(0)\mathtt{Z}(0:1\to 2)\mathtt{L}(1)\mathtt{Z}(0:0\to 2)\mathtt{Z}(0:5\to 3)$,
- $Y \rightsquigarrow \mathtt{L}(1)\mathtt{L}(1)\mathtt{L}(0)\mathtt{L}(0)\mathtt{Z}(0:1\to 2)$,
- $X\_|^{+}Y \rightsquigarrow \mathtt{Z}(1:1\to 3)\mathtt{Z}(1:4\to 2)\mathtt{Z}(0:0\to 2)\mathtt{Z}(0:5\to 3)$,
- $Y\_|^{+}X \rightsquigarrow \mathtt{Z}(1:4\to 3)\mathtt{Z}(1:6\to 3)$

En prenant la fonction admissible indicatrice, on a donc $S_{\mathbb{1}}(X,Y)^{|u} = 8 \neq 9 = S_{\mathbb{1}}(Y,X)^{|u}$. Nous perdons donc la symétrie, mais les autres propriétés (invariance, positivité et bornes) restent vérifiées (la démonstration est présentée dans l'annexe A). Cette version est moins contraignante sur les données que l'encodage cardinal car il suffit que la taille de l'alphabet de X et Y, $\mathcal{A}_{XY} = \mathcal{A}_X \cup \mathcal{A}_Y$, soit 256.

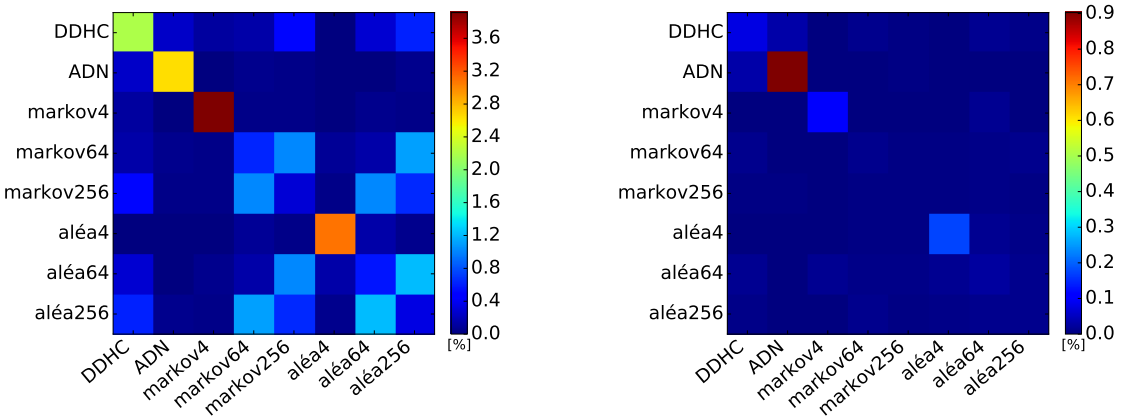
Nous allons maintenant observer l'asymétrie de SALZA jointe concaténée à travers le même jeu de données que nous avions utilisé pour tester l'ajout de connaissance (section II.2.2). De plus, nous utilisons de nouveau la fonction admissible exponentielle. En effet, nous verrons dans la section II.4.1 que c'est la fonction admissible exponentielle qui a le meilleur pouvoir discriminant autour de $l^\star$ parmi celles que nous avons présentées (section II.1.3). Pour chacune des chaînes de ces jeux de données, nous observons l'asymétrie de SALZA jointe concaténée selon les différentes catégories du jeu de données. Pour cela, nous effectuons pour chaque chaîne X et Y du jeu de données les calculs suivants :

Asymétrie : $S_{\mathbb{E}}(X,Y)^{|u} - S_{\mathbb{E}}(Y,X)^{|u}$, cette différence permet d'évaluer l'asymétrie de SALZA jointe concaténée pour X et Y.

Normalisation : $\frac{S_{\mathbb{E}}(X,Y)^{|u} - S_{\mathbb{E}}(Y,X)^{|u}}{S_{\mathbb{E}}(X)+S_{\mathbb{E}}(Y)}$, pour pouvoir comparer la complexité de chaînes de tailles différentes, il faut normaliser l'asymétrie. Pour cela nous divisons par la borne supérieure de la complexité jointe.

Les résultats sont synthétisés dans la figure II.6a. L'abscisse représente le jeu de données dont la chaîne X est tirée et l'axe des ordonnées représente celui dont est issue la chaîne Y. Nous représentons, dans la figure II.6, l'asymétrie maximale pour tous les couples de chaînes prises dans les jeux de données. La variance et la moyenne pour chaque catégorie sont présentées dans le tableau B.5 présent dans l'annexe B.

L'asymétrie de SALZA jointe concaténée ne dépasse jamais 3,5%. Cette asymétrie est plus faible que l'asymétrie obtenue avec les compresseurs réels (voir section I.3.2). Comme nous l'avons vu dans la section I.3.2, les compresseurs réels ne sont pas symétriques et ne devraient pas être des compresseurs normaux. Cependant Li et Vitányi les considèrent comme tels malgré une asymétrie allant jusqu'à 4%. Ainsi nous pouvons considérer qu'en pratique, l'encodage concaténé permet d'estimer SALZA jointe de manière satisfaisante.



(a) Encodage concaténé, $S_{\mathbb{E}}(X,Y) = S_{\mathbb{E}}(X,Y)^{|u}$.

(b) Encodage simultané, $S_{\mathbb{E}}(X,Y) = S_{\mathbb{E}}(X,Y)^{|s}$.

FIGURE II.6 – Asymétrie de SALZA jointe : calcul de $\frac{S_{\mathbb{E}}(X,Y) - S_{\mathbb{E}}(Y,X)}{S_{\mathbb{E}}(X)+S_{\mathbb{E}}(Y)}$ pour X et Y appartenant aux différents jeux de données. Représentation des maxima pour l'ensemble des chaînes X du jeu de données en abscisse connaissant chaque chaîne Y du jeu de données en ordonnée. Si la couleur est bleu alors SALZA jointe est symétrique, si elle est rouge SALZA n'est pas symétrique.

II.3.1.3 Encodage simultané

Il existe d'autres alternatives que l'encodage concaténé pour décomposer X et Y de la plus petite façon qu'il soit. Par exemple, au lieu de coder entièrement X puis Y connaissant X comme dans l'encodage concaténé, nous pouvons imaginer coder un symbole de X, puis un symbole de Y, puis de nouveau un symbole de X, puis un symbole de Y, ... et ainsi de suite jusqu'à ce que X et Y soient entièrement décomposés. Les deux chaînes sont alors encodées simultanément. Pour décomposer X nous connaissons son propre passé et celui de Y. Pour décomposer Y nous connaissons son propre passé et celui de X mais aussi son

présent. En effet, comme l'octet présent de X vient d'être décomposé nous pouvons y faire référence. On peut donc définir une complexité jointe basée sur cet encodage.

Définition II.3.4 (SALZA jointe simultanée)**.**
Soit deux chaînes X et Y, alors ***SALZA*** *jointe basée sur l'encodage simultané est appelée* ***SALZA*** *jointe simultanée et est définie comme*

$$S_f(X,Y)^{|s} = S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y_{\text{-}}|^{p}X) \tag{II.4}$$

On décompose X connaissant son propre passé et celui de Y puis on décompose Y connaissant son propre passé et le présent de X.

Comme nous l'avions vu pour l'encodage concaténé, nous perdons la propriété de symétrie pour la même raison. Cependant, les autres propriétés restent vérifées et tout comme pour l'encodage concaténé, il suffit que la taille de l'alphabet de X et Y, $\mathcal{A}_{XY} = \mathcal{A}_X \cup \mathcal{A}_Y$, soit $\alpha = 256$.

Nous avons représenté sur la figure II.6b l'asymétrie de SALZA jointe simultanée tout comme nous l'avions fait pour SALZA jointe concaténée. Nous pouvons observer que l'asymétrie de SALZA jointe simultanée ne dépasse pas 0,9% et se situe plus souvent autour de 0,2%. Son asymétrie est donc beaucoup moins forte que celle de SALZA jointe concaténée et celle des compresseurs. Ainsi SALZA jointe simultanée possède bien les mêmes bornes qu'une mesure jointe, est positive et invariante et son asymétrie est plus faible que les estimations classiques de la complexité jointe. Nous pouvons donc affirmer que l'encodage simultané permet d'estimer SALZA jointe.

II.3.1.4 Comparaison des différentes formes d'encodage

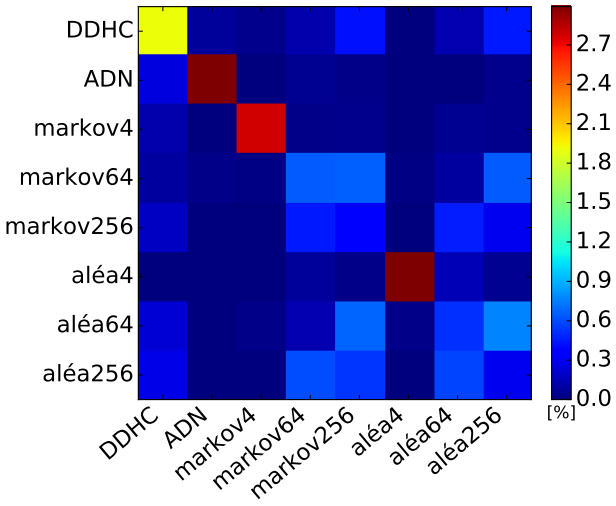
Les trois encodages que nous venons de décrire interprètent chacun à leur manière la meilleure décomposition de X et Y. Ils présentent chacun des avantages et des inconvénients. Nous proposons donc de les comparer en pratique.

Pour cela, nous utilisons les mêmes données que nous avions utilisées pour tester l'ajout d'information (section II.2.2) ou mesurer l'asymétrie de l'encodage concaténé et l'encodage simultané. Pour chacune des chaînes de ces jeux de données nous calculons SALZA jointe basée sur un encodage k et SALZA jointe basée sur un autre encodage l et nous observons la différence. Pour cela nous nous basons sur la méthode décrite ci dessous.

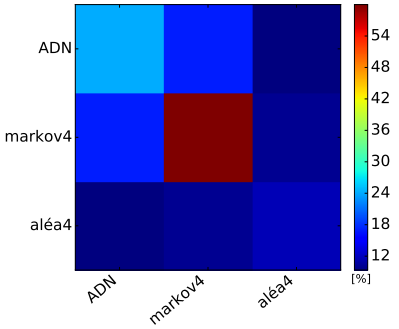
Comparaison : $S_{\mathbb{E}}(X,Y)^{|k} - S_{\mathbb{E}}(X,Y)^{|l}$, cette différence permet évaluer la différence de SALZA jointe entre l'encodage k et l'encodage l.

Normalisation : $\frac{S_{\mathbb{E}}(X,Y)^{|k} - S_{\mathbb{E}}(X,Y)^{|l}}{S_{\mathbb{E}}(X)+S_{\mathbb{E}}(Y)}$, pour pouvoir comparer la complexité de chaînes de taille différente, il faut normaliser la comparaison. Pour cela nous divisons par la borne supérieur de la complexité jointe.

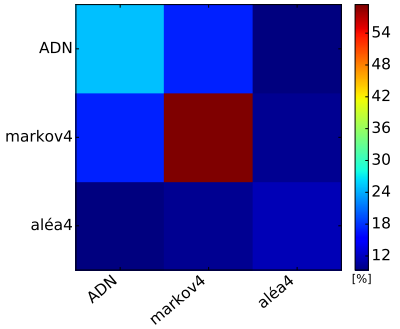
Les résultats obtenus sont synthétisés dans la figure II.7. L'abscisse représente le jeu de données dont la chaîne X est tirée et l'axe des ordonnées représente celui dont est issue la chaîne Y. Nous représentons, dans la figure II.7, le maximum de la différence entre les deux encodages pour tous les couples de chaînes prises dans les jeux de données. La variance et



(a) Encodage concaténé vs encodage simultané, $S_{\mathbb{E}}(X,Y)^{|l} = S_{\mathbb{E}}(X,Y)^{|u}$ et $S_{\mathbb{E}}(X,Y)^{|k} = S_{\mathbb{E}}(X,Y)^{|s}$.



(b) Encodage concaténé vs encodage cardinal, $S_{\mathbb{E}}(X,Y)^{|l} = S_{\mathbb{E}}(X,Y)^{|u}$ et $S_{\mathbb{E}}(X,Y)^{|k} = S_{\mathbb{E}}(X,Y)^{|c}$.



(c) Encodage simultané vs encodage cardinal, $S_{\mathbb{E}}(X,Y)^{|l} = S_{\mathbb{E}}(X,Y)^{|c}$ et $S_{\mathbb{E}}(X,Y)^{|k} = S_{\mathbb{E}}(X,Y)^{|s}$.

FIGURE II.7 – Différence entre les différents encodages de SALZA jointe : calcul de $\frac{S_{\mathbb{E}}(X,Y)^{|k} - S_{\mathbb{E}}(X,Y)^{|l}}{S_{\mathbb{E}}(X)+S_{\mathbb{E}}(Y)}$ pour k et l deux méthodes d'encodage et X et Y appartenant aux différents jeux de données. Représentation des maxima pour l'ensemble des chaînes X du jeux de données en abscisse connaissant chaque chaîne Y du jeu de données en ordonnée. Si la case est bleue alors les deux encodages donnent des résultats numériques proches pour SALZA jointe, si la case est rouge alors les deux encodages donnent des résultats numériques différents.

la moyenne pour chaque jeu de données sont présentées dans les tableaux B.7, B.8 et B.9 présents dans l'annexe B.

Nous voyons sur la figure II.7a que l'encodage concaténé et l'encodage simultané donnent des résultats relativement proches l'un de l'autre. Ce qui nous conforte dans l'idée qu'il n'y a pas une seule meilleure décomposition de X et Y. Par contre nous pouvons voir dans les figures II.7b et II.7c que l'encodage cardinal donne des résultats très éloignés des deux autres encodages.

En résumé

Il existe trois encodages différents pour calculer SALZA jointe, c'est-à-dire la plus petite décomposition de X et Y .

L'encodage cardinal :

- *Principe* : nous décomposons la chaîne Z où chaque élément z_i de Z est le vecteur qui contient l'élément i de X et Y, $z_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix}$.

$$S_f(X,Y)^{|c} = S_f(Z)$$

- *Avantage* : il est le seul encodage qui garde strictement la symétrie.
- *Inconvénient* : il restreint la taille des alphabets et ses résultats pratiques sont très éloignés des résultats des autres encodages.

L'encodage concaténé :

- *Principe* : nous décomposons d'abord X puis Y sachant X.

$$S_f(X,Y)^{|u} = S_f(X) + S_f(Y_{\text{-}}|^{+}X)$$

- *Avantage* : il ne restreint pas la taille de l'alphabet de X et Y en dessous de 256 et c'est cet encodage qui est principalement utilisé dans la littérature.
- *Inconvénient* : il n'est plus symétrique.

L'encodage simultané :

- *Principe* : nous décomposons simultanément X connaissant Y et Y connaissant X.

$$S_f(X,Y)^{|s} = S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y_{\text{-}}|^{\text{p}}X)$$

- *Avantage* : tout comme l'encodage concaténé il ne restreint pas les alphabets et il est beaucoup moins asymétrique que l'encodage concaténé.
- *Inconvénient* : il ne peut pas être reproduit par des compresseurs.

II.3.2 Information basée sur SALZA

Nous avons déjà défini SALZA simple, conditionnelle et jointe. Il est donc possible de définir maintenant l'information mutuelle algorithmique de X et Y, qui mesure la dépendance algorithmique entre ces deux chaînes de caractères. De plus nous pouvons mesurer la dépendance algorithmique directionnelle entre ces deux chaînes grâce à l'information dirigée algorithmique de X et Y.

II.3.2.1 Information mutuelle algorithmique

SALZA étant une mesure d'information et SALZA étant définie, il est possible d'écrire l'information mutuelle algorithmique basée sur SALZA. Par définition, l'information mutuelle algorithmique basée sur une mesure d'information R s'écrit comme suit (voir section I.2.2 et l'équation I.11) :

$$I_R(X:Y) = R(X) + R(Y) - R(X,Y).$$

Propriété II.3.2 (Information mutuelle algorithmique)**.**
Une information mutuelle algorithmique basée sur la mesure d'information R respecte les propriétés suivantes :

1. *Positive : $I_R(X:Y) \geq 0$,*
2. *Symétrique : $I_R(X:Y) = I_R(Y:X)$,*
3. *Invariance : $I_R(X,X) = R(X)$,*

Il est donc possible de définir une information mutuelle algorithmique basée sur les trois définitions de SALZA jointe (cardinale, concaténation et simultanée).

$$I_{S_f}(X:Y)^{|c} = S_f(X) + S_f(Y) - S_f(X,Y)^{|c} \tag{II.5}$$

$$I_{S_f}(X:Y)^{|u} = S_f(X) + S_f(Y) - S_f(X,Y)^{|u} = S_f(Y) - S_f(Y_{\text{-}}|^{+}X) \tag{II.6}$$

$$I_{S_f}(X:Y)^{|s} = S_f(X) + S_f(Y) - S_f(X,Y)^{|s} = S_f(X) - S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y) - S_f(Y_{\text{-}}|^{\text{p}}X) \tag{II.7}$$

Grâce aux propriétés des SALZA jointes, il est facile de montrer que les trois informations mutuelles algorithmiques respectent les propriétés II.3.2 à quelques erreurs négligeables près. En effet, comme nous l'avons vu précédemment l'encodage concaténé et l'encodage simultané ne sont pas totalement symétriques mais cette asymétrie peut être considérée comme minime. L'encodage cardinal lui ne respecte pas la positivité à 1 près.

L'information mutuelle algorithmique permet de mesurer l'indépendance algorithmique entre deux chaînes de caractères. Si l'information mutuelle algorithmique est nulle alors les deux chaînes sont indépendantes algorithmiquement. Plus l'information mutuelle se rapproche de sa borne maximale et plus les deux chaînes sont considérées comme dépendantes.

II.3.2.2 Information dirigée

L'information mutuelle probabiliste ou algorithmique permet de mesurer la dépendance des deux variables X et Y [Kol65]. Cependant, comme elle est symétrique, elle ne peut pas servir à mesurer la dépendance directionnelle : est-ce que X est à l'origine de Y ou l'inverse. C'est pourquoi en 1990, Massey introduit une information dirigée probabiliste basée sur l'entropie de Shannon qui mesure la dépendance directionnelle [Mas90]. Dans un premier temps, nous allons donc présenter l'information dirigée probabiliste. Puis, nous montrerons comment définir l'information dirigée algorithmique avec SALZA.

• Information dirigée probabiliste [Mas90]

Nous travaillons ici sur deux variables aléatoires $\mathbf{X}$ et $\mathbf{Y}$. Les séquences X et Y de longueur $n = l(X) = l(Y)$ sont n tirages consécutifs des variables aléatoires associées. Tout d'abord rappelons une propriété de la probabilité jointe de deux variables aléatoires $\mathbf{X}$ et $\mathbf{Y}$:

$$p(X,Y) = \overleftarrow{P}(X|Y)\overrightarrow{P}(Y|X)$$

où

$$\overleftarrow{P}(X|Y) = \prod_{i=1}^{n} P(x_i|x_{1:i-1}, y_{1:i-1}),$$

$$\overrightarrow{P}(Y|X) = \prod_{i=1}^{n} P(y_i|x_{1:i}, y_{1:i-1}).$$

où $n = l(X) = l(Y)$.

On peut alors définir l'entropie de Shannon basée sur ces deux distributions

$$\overleftarrow{H}(X|Y) = -\sum_{i=1}^{n} P(x_i|x_{1:i-1}, y_{1:i-1}) log_2 P(x_i|x_{1:i-1}, y_{1:i-1}) \tag{II.8}$$

et

$$\overrightarrow{H}(Y|X) = -\sum_{i=1}^{n} P(y_i|x_{1:i}, y_{1:i-1}) log_2 P(y_i|x_{1:i}, y_{1:i-1}). \tag{II.9}$$

La mesure de l'indépendance directionnelle de X vers Y peut être mesurée grâce à l'entropie $\overrightarrow{H}(Y|X)$ comme suit.

Définition II.3.5 (Information dirigée basée sur l'entropie de Shannon)**.**
L'information dirigée basée sur l'entropie de Shannon est :

$$I_H(X \to Y) = H(Y) - \overrightarrow{H}(Y|X). \tag{II.10}$$

L'information dirigée probabiliste est l'information apportée à Y par X, si $I_H(X \to Y) = 0$ alors Y est indépendante de X dans le sens $X \to Y$

Nous allons maintenant définir l'opérateur `retard` qui décalle une chaîne d'un symbole et se note `R`. La chaîne `R`X est la chaîne X retardée de 1, c'est-à-dire que le $i^{\text{ième}}$ symbole de `R`X est le $i+1^{\text{ième}}$ symbole de X. Nous disons aussi qu'il y a rétroaction entre X et Y, si X apporte de l'information à Y et que Y apporte de l'information à X ([Gra69]).

Propriété II.3.3 (Information dirigée basée sur l'entropie de Shannon)**.**
Toute information dirigée I doit respecter ces propriétés :

1. *Positivité : $I(X \to Y) \geq 0$,*
2. *Borne supérieure : $I(X \to Y) \leq I(X : Y)$, l'égalité est vérifiée quand il n'y a pas de rétroaction entre X et Y,*
3. *Décomposition : $I(X \to Y) + I(\texttt{R}Y \to X) = I(X : Y)$.*

• Information dirigée algorithmique

Pour les mêmes raisons que dans le domaine probabiliste, nous proposons de définir l'information dirigée algorithmique qui mesure la dépendance directionnelle algorithmique. Pour cela nous allons transposer les relations probabilistes en relations algorithmiques.

Le pendant algorithmique basé sur `SALZA` de $\overrightarrow{H}(Y|X)$ est le conditionnement $\_|^{\text{p}}$. En effet, comme nous le voyons dans l'équation II.9, $\overrightarrow{H}(Y|X)$ est la probabilité de chaque réalisation y_i connaissant toutes les réalisations passées de X et Y ainsi que la réalisation présente de X. De même le pendant algorithmique de $\overleftarrow{H}(Y|X)$ est le conditionnement $\_|^{\text{-}}$.

Nous avons montré dans la section II.3.1 que l'encodage simultané permet de calculer `SALZA` jointe. Ainsi, `SALZA` jointe simultanée est égale à la somme des deux complexités dirigées tout comme l'entropie de Shannon jointe est égale à la somme des deux entropies dirigées.

$$S_f(X,Y)^{|s} = S_f(X\_|^{\text{-}}Y) + S_f(Y\_|^{\text{p}}X).$$

Nous pouvons donc, de façon analogue à la théorie probabiliste (equation II.10), définir l'information dirigée basée sur `SALZA`, qui mesure l'indépendance directionnelle.

Définition II.3.6 (Information dirigée algorithmique basée sur `SALZA`)**.**
L'information dirigée basée sur `SALZA` est

$$I_{S_f}(X \to Y) = S_f(Y) - S_f(Y\_|^{p}X). \tag{II.11}$$

L'information dirigée algorithmique est l'information apportée à Y par son propre passé et le présent de X, cette information vaut zéro si Y est algorithmiquement indépendante de X dans le sens $X \to Y$. Cette définition respecte bien toutes les propriétés II.3.3. La démonstration figure dans l'annexe A.

En résumé

Information mutuelle algorithmique basée sur SALZA :

- *Principe* : l'information mutuelle algorithmique mesure la dépendance algorithmique entre X et Y

$$I_{S_f}(X : Y) = S_f(X) + S_f(Y) - S_f(X, Y).$$

- *Avantage* : elle ne repose sur aucune hypothèse probabiliste.
- *Inconvénient* : les propriétés sont vraies à une erreur négligeable près.

Information dirigée basée sur SALZA :

- *Principe* : l'information dirigée mesure la dépendance directionnelle algorithmique entre X et Y

$$I_S(X \to Y) = S_f(Y) - S_f(Y\_|^{\mathrm{p}}X).$$

- *Avantage* : l'information dirigée algorithmique ne repose sur aucune hypothèse probabiliste et elle ne peut pas être calculée avec un estimateur de complexité autre que SALZA.
- *Inconvénient* : les propriétés sont vraies à une erreur négligeable près.

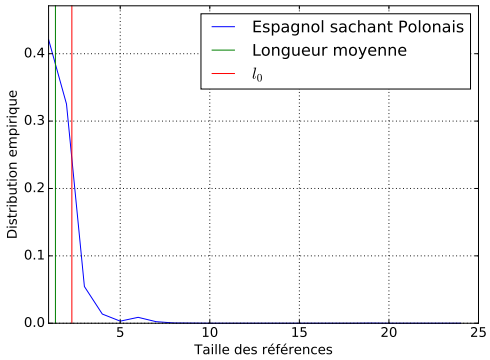
II.4 Les performances de SALZA

Pour estimer la complexité de Kolmogorov, nous pouvons utiliser la complexité de Lempel-Ziv (voir section I.2), la taille des fichiers compressés (voir section I.3) ou SALZA. Nous allons donc maintenant comparer notre estimateur SALZA aux deux autres estimateurs de complexité.

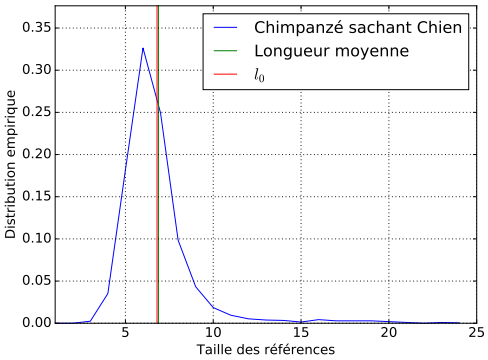
Dans un premier temps, nous comparons la complexité de Lempel-Ziv et SALZA d'un point de vue théorique. Cette analyse nous permettra de choisir la fonction admissible optimale pour SALZA. Cependant, l'approche choisie pour comparer théoriquement la complexité de Lempel-Ziv et SALZA ne peut pas être appliquée pour les compresseurs. Nous ne pouvons donc pas comparer théoriquement les compresseurs à la complexité de Lempel-Ziv et SALZA. Dans un deuxième temps, nous effectuons donc une comparaison pratique de ces trois estimateurs de la complexité de Kolmogorov. Pour cela nous réalisons une simulation qui permet de faire varier la longueur moyenne des symboles. Ces comparaisons pratiques et théoriques nous permettent alors de mettre en évidence l'utilité de SALZA.

II.4.1 Influence de la fonction admissible

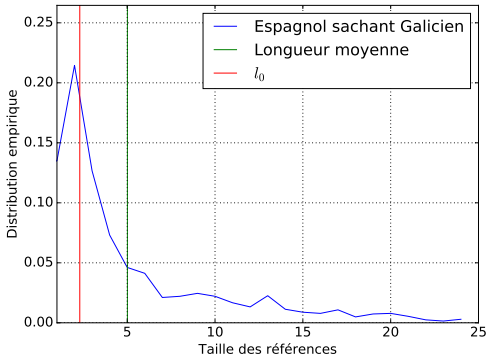
Supposer un modèle probabiliste sur les chaînes X et Y ne permet pas de déduire facilement les performances théoriques de $S_f(X \wr Y)$. Cependant il est possible de calculer et d'analyser ces performances si nous supposons un modèle probabiliste sur l'ensemble



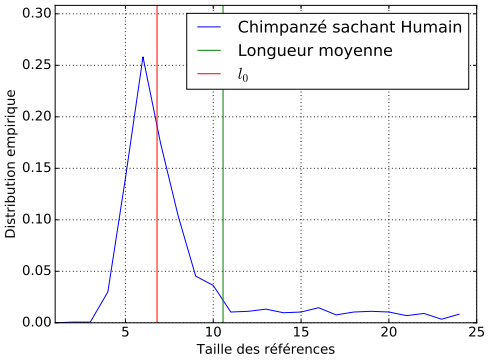
(a) Texte en espagnol connaissant le même texte en polonais.



(b) ADN de chimpanzé connaissant l'ADN du chien.



(c) Texte en espagnol connaissant le même texte en galicien galician.



(d) ADN de chimpanzé connaissant l'ADN d'humain.

FIGURE II.8 – Différentes distributions de longueurs empiriques pour la décomposition $|^+$.

$\mathcal{L}_{X\wr Y}$ des longueurs de la factorisation $X\wr Y$. Ce modèle probabiliste est estimé de manière empirique grâce à nos jeux de données.

Nous présentons la distribution des longueurs empiriques pour différentes données dans la figure II.8. Nous avons choisi la décomposition $|^+$ mais des résultats similaires peuvent être obtenus avec les autres décompositions. Dans un premier temps, nous utilisons le texte de la Déclaration Universelle des Droits de l'Homme rédigé en différentes langues indo-européennes [2]. Le premier article de la Déclaration Universelle des Droits de l'Homme en espagnol, polonais et galicien est présenté ci-après.

Espagnol, [VEsp] : Todos los seres humanos nacen libres e iguales en dignidad y derechos y, dotados como están de razón y conciencia, deben comportarse fraternalmente los unos con los otros.

Galicien, [VGal] : Tódolos seres humanos nacen libres e iguais en dignidade e dereitos e, dotados como están de razón e conciencia, díbense comportar fraternalmente uns cos outros.

2. Pour plus de détails sur les données voir l'annexe D

Polonais, [VPol] : Wszyscy ludzie rodzą się wolni i równi pod względem swej godności i swych praw. Są oni obdarzeni rozumem i sumieniem i powinni postępować wobec innych w duchu braterstwa.

Nous observons que l'espagnol et le galicien sont très proches : certaines parties du premier article de la Déclaration Universelle des Droits de l'Homme sont identiques ([VEsp] et [VGal]). Par contre, l'espagnol est très éloigné du polonais. Il est impossible dans la version [VEsp] de faire des opérations `copier` depuis la version polonaise [VPol] pour décomposer la version espagnole. Nous décomposons alors la version espagnole sachant le polonais ou le galicien avec la décomposition $|^+$. La décomposition de la version espagnole connaissant la version galicienne est représentée dans la figure II.8c et celle connaissant la version polonaise est représentée dans la figure II.8a. On observe que les opérations `copier` depuis la version galicienne pour expliquer la version espagnole sont assez longues. Mais depuis la version polonaise, il est presque nécessaire de réaliser uniquement des opérations `insérer`.

Dans un deuxième temps, nous observons une distribution empirique des longueurs basée sur des fichiers ADN. L'ADN est sous la forme d'une chaîne de quatre nucléotides symbolisées par $\{A, C, G, T\}$ [2]. Nous décomposons alors l'ADN de chimpanzé connaissant l'ADN de chien ou l'ADN d'humain. Le chimpanzé et l'humain étant proches génétiquement, les longueurs sont en moyenne élevées ; à l'inverse pour le chimpanzé et le chien les longueurs sont en moyenne plus courtes. Cependant, en moyenne, la longueur des opérations `copier` reste plus grande pour l'ADN que pour les langues. En effet l'ADN a un alphabet de taille 4 tandis que les textes en différentes langues ont un alphabet de taille 64. Comme les deux chaînes ont presque la même taille, il est logique de faire les opérations `copier` plus longues dans l'ADN que dans les langues (voir la définition II.1.3).

Les observations expérimentales (figure II.8) semblent suggérer qu'un modèle de type discret, unimodale, tronqué en zéro et à support fini pourait être pertinent. En effet les longueurs des opérations `copier` sont des entiers compris entre 1 et $l(X)$ et sur toutes les représentations des distributions nous voyons bien apparaître un seul mode. Nous avons donc utilisé une loi de Poisson tronquée en zéro. Cette loi a cependant un support infini, mais il est possible de restreindre cette loi sur un support fini. Cependant, nous allons utiliser des moyennes entre 1 et 20 et les chaînes étudiées auront une longueur 30000. Ainsi si nous utilisons une loi de Poisson à support fini ou infini, nous avons les mêmes résultats numériques. C'est pourquoi nous avons choisi d'utiliser une loi de Poisson tronquée en zéro à support infini. La distribution de Poisson tronquée en zéro en fonction du paramètre λ est :

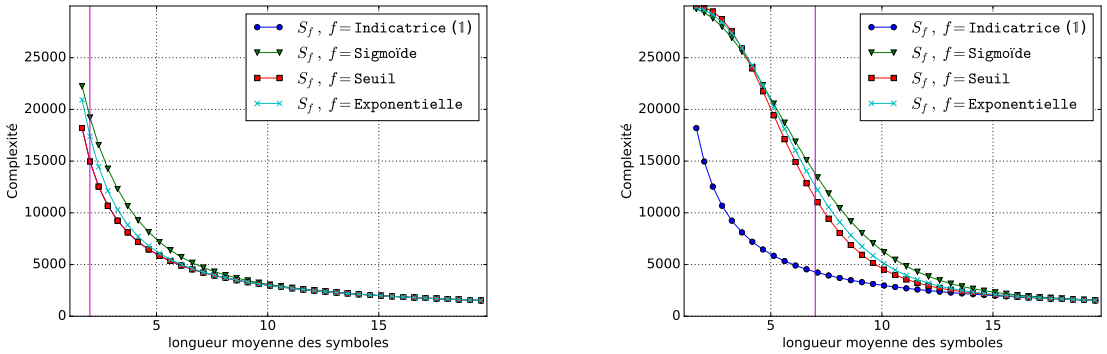
$$P(X = k) = \frac{\lambda^k}{(e^\lambda - 1)k!}.$$

Sa moyenne est :

$$m(\lambda) = \frac{\lambda e^\lambda}{e^\lambda - 1}.$$

Il est possible de calculer `SALZA` pour une telle distribution des longueurs. Alors, S s'écrit en fonction de λ comme suit, le développement des calculs est présenté en annexe A :

$$S_f(X \wr Y)_{|\lambda} = l(X) \left(1 - \sum_{k=1}^{l(X)} e^{-\lambda} \frac{\lambda^{k-1}}{k!} (k-1) f(k) \right).$$



(a) $l^\star$=2. (b) $l^\star$=7.

FIGURE II.9 – Évolution théorique de la complexité SALZA en fonction de la longueur moyenne pour une distribution de Poisson tronquée en zéro.

Étudions maintenant l'évolution théorique de SALZA en fonction de λ. Pour cela, nous fixons $l(X) = 30000$ et nous choisissons deux longueurs seuils différentes : $l^\star = 2$ et $l^\star = 7$. Ces deux longueurs nous permettent d'observer le comportement dans le cas d'une petite longueur seuil et dans celui où la longueur seuil est grande. Ces deux extrêmes nous permettent donc d'illustrer une tendance. La figure II.9 représente l'évolution de la complexité SALZA avec différentes fonctions admissibles en fonction de la longueur moyenne des opérations `copier`, λ.

Dans la figure II.9a, la longueur seuil est égale à 2. Nous pouvons constater que, quelle que soit la fonction admissible, la complexité SALZA reste presque la même. Le choix de la fonction admissible n'est donc pas très important avec une longueur seuil très petite.

Par contre si la longueur seuil est plus grande comme dans la figure II.9b, nous observons une nette différence entre la fonction indicatrice, c'est-à-dire la complexité de Lempel-Ziv étendue, et les autres fonctions admissibles. Pour la complexité de Lempel-Ziv, la courbe ne change pas entre les figures II.9a et II.9b. Pour les autres fonctions admissibles, la valeur de la complexité est centrée autour de la longueur seuil $l^\star$. Ainsi si les longueurs sont en moyenne *significatives* (voir section II.1.3), c'est-à-dire qu'en moyenne elles sont plus grandes que $l^\star$, alors X est peu complexe sachant Y selon la décomposition $X \wr Y$. À l'inverse, si les longueurs sont en moyenne plus petites que $l^\star$ alors les longueurs sont en moyennes *non significatives*. Donc X est complexe connaissant Y selon le conditionnement $\wr$.

Pour bien comprendre l'influence des longueurs seuils, examinons la figure II.10. Sur cette figure nous supposons toujours que $\mathcal{L}_{X\wr Y}$ suit une loi de Poisson tronquée, mais nous fixons ici la longueur moyenne à 10. Nous supposons maintenant que la taille de l'alphabet de X évolue. Pour rappel, nous avons vu qu'il est possible de calculer la longueur seuil de façon automatique avec $l_0 = log_\alpha l(X)$ (voir section II.1.3). En posant $l^\star = l_0$, la longueur seuil va donc évoluer en fonction de la taille de l'alphabet. La complexité de Lempel-Ziv ne change pas : le nombre de symboles reste le même. Par contre SALZA avec les autres fonctions admissibles évolue en fonction de $l^\star$. Pour comprendre pourquoi cela est important, prenons l'exemple de deux chaînes : l'une sur un alphabet de taille 2 et

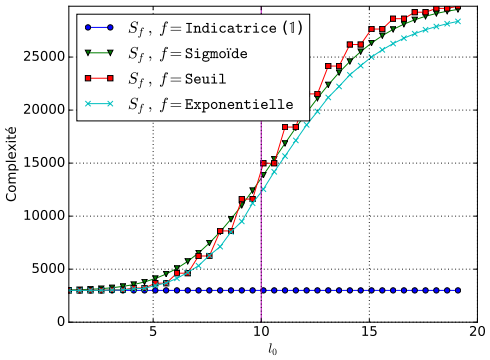


FIGURE II.10 – Évolution théorique de la complexité SALZA en fonction de la longueur seuil de la fonction admissible pour une distribution de Poisson tronquée en zéro avec une longueur moyenne m=10.

l'autre sur un alphabet de taille 256. Nous nous attendons pour une chaîne aléatoire à des longueurs en moyenne de taille $l_0 = log_2(30000) = 13,9$ pour l'alphabet de taille 2 (voir définition II.1.3) et $l_0 = log_{256}(30000) = 1,7$ pour l'alphabet de taille 256. Donc dans le premier cas, avoir une moyenne des longueurs de 10 n'est donc pas exceptionnel et la chaîne est donc relativement complexe. Par contre dans le deuxième cas, avoir une moyenne des longueurs de 10 est exceptionnel et la chaîne est donc très peu complexe. La complexité de Lempel-Ziv ne fait aucune différence entre ces deux cas. Par contre SALZA a une complexité différente entre les deux cas si nous réglons correctement la longueur seuil $l^\star$. Nous observons toutefois que la fonction seuil établit des paliers qui peuvent être gênants pour la précision de la complexité.

Évaluons maintenant les performances des différentes fonctions admissibles. Pour cela nous utilisons le pouvoir discriminant qui est défini comme la dérivée de S_f par rapport à la moyenne de la distribution des longueurs.

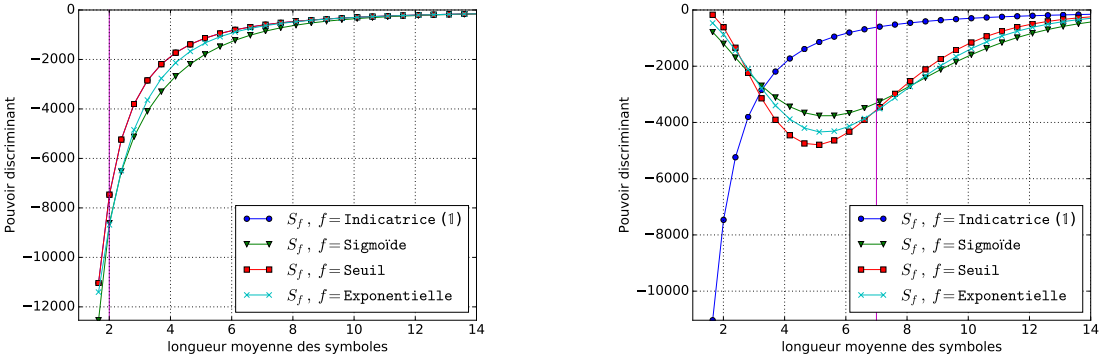
Définition II.4.1 (Pouvoir discriminant)**.**
Soient

- *p une densité de probabilité où m est son espérance.*
- *$S_f(X \wr Y)_{|m}$ la valeur de **SALZA** telle que l'ensemble des longueurs des opérations suivent la densité de probabilité p.*

*Le pouvoir discriminant $P_{S_f}(m)$ de **SALZA** en fonction de la longueur moyenne des symboles est défini comme :*

$$P_{S_f}(m) = \frac{d\, S_f(X \wr Y)_{|m}}{d\, m}$$

Il est possible de calculer le pouvoir discriminant de SALZA pour une loi de Poisson tronquée en zéro.



(a) $l^\star$=2. (b) $l^\star$=7.

FIGURE II.11 – Évolution théorique du pouvoir discriminant de la complexité SALZA en fonction de la longueur moyenne pour une distribution de Poisson tronquée en zéro.

Théorème II.4.1 (Pouvoir discriminant pour une loi de Poisson tronquée en zéro)**.** *Si $\mathcal{L}_{X|Y}$ suit une loi de Poisson tronquée en zéro de paramètre λ, alors*

$$P_{S_f}(\lambda) = -l(X)\frac{(1-e^{-\lambda})^2 e^{-\lambda}}{1-e^{-\lambda}(1+\lambda)}\sum_{k=1}^{\infty}\frac{\lambda^{k-2}}{k!}(k-1-\lambda)(k-1)f(k).$$

En particulier si f est la fonction indicatrice $\mathbb{1}$ alors nous obtenons le pouvoir discriminant de la complexité de Lempel-Ziv étendue :

$$P_{S_{\mathbb{1}}}(\lambda) = -l(X)\frac{(1-e^{-\lambda})^2}{\lambda^2}$$

La démonstration est présentée dans l'annexe A.

Nous pouvons observer le pouvoir discriminant de SALZA pour différentes fonctions admissibles sur la figure II.11 pour deux longueurs seuils $l^\star$ différentes. Tout comme nous l'avions observé lors de l'évolution de la complexité, il y a assez peu de différences entre les fonctions admissibles lorsque la longueur seuil vaut 2 (voir figure II.11a). Pour une longueur seuil qui vaut 7 (voir figure II.11b), nous observons que la fonction indicatrice possède toujours le même pouvoir discriminant, ce qui est normal car elle ne dépend pas de $l^\star$. Cependant, les autres fonctions admissibles ont un meilleur pouvoir discriminant autour de $l^\star$ ce qui permet de séparer les opérations `copier` *significatives* des *non-significatives*. Nous observons que la fonction exponentielle possède le plus grand pouvoir discriminant après la fonction seuil. Cependant, nous avons vu précédemment que la fonction seuil introduisait des palliers ce qui n'est pas souhaitable. Par conséquent, nous considérons que la fonction admissible par défaut est pour la suite de nos travaux la fonction exponentielle.

En résumé

Observation de l'influence des fonctions admissibles.

- *La fonction indicatrice* : donne les mêmes résultats pour toutes les longueurs seuils $l^\star$, elle ne permet pas de faire la différence entre les opération `copier` *significatives* et *non-significatives*.
- *La fonction seuil* : comme la fonction n'est pas continue selon $l^\star$, `SALZA` ne l'est pas non plus ce qui peut engendrer des problèmes de précision.
- *La fonction exponentielle et la fonction sigmoïde* : la fonction exponentielle a un pouvoir discriminant supérieur à celui de la sigmoïde, c'est donc cette dernière que nous utilisons comme fonction admissible par défaut.

II.4.2 Comparaison des performances de SALZA avec les compresseurs

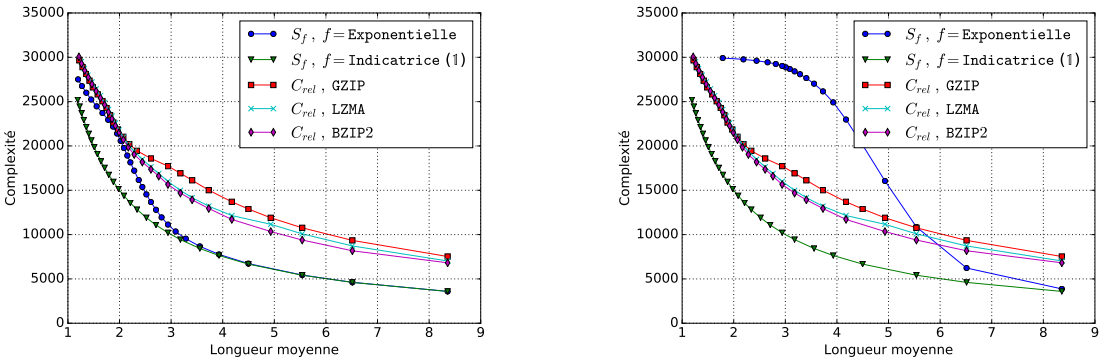
Nous avons pu observer dans la section précédente les différences entre notre estimateur de complexité `SALZA` et la complexité de Lempel-Ziv. Nous allons maintenant comparer `SALZA` à d'autres estimateurs de la complexité de Kolmogorov : les compresseurs. Nous allons pour cela étudier les trois compresseurs que nous avons vus dans la section I.3.1 : `GZIP`, `LZMA` et `BZIP2`. Nous ne pouvons cependant pas modéliser la distribution des longueurs pour les différents compresseurs comme nous l'avons fait pour la complexité de Lempel-Ziv et `SALZA` dans la section précédente. En effet, les logiciels de compression sont souvent composés de deux parties : une décomposition et un encodage (pour plus de détails, voir la section I.3). Si nous voulons contrôler la longueur des symboles, il faudrait avoir uniquement l'accès aux étapes d'encodage ce qui n'est pas évident. Nous avons donc réalisé une comparaison pratique entre `SALZA`, la complexité de Lempel-Ziv et les compresseurs en simulant des données plus ou moins complexes.

Pour simuler des données plus ou moins complexes, nous avons fait évoluer la taille de l'alphabet. En effet, comme nous l'avons vu dans la définition II.1.3, la complexité d'une chaîne aléatoire i.i.d. va dépendre de la taille de son alphabet. Nous avons donc généré 100 chaînes aléatoires uniformes i.i.d. pour une taille d'alphabet allant de 2 à 256 et de 30000 caractères [3]. Puis pour chacune de ces chaînes, nous calculons :

- `SALZA` avec la fonction admissible exponentielle,
- la complexité de Lempel-Ziv avec `SALZA` et la fonction admissible indicatrice, et
- la taille du fichier compressé avec `GZIP`, `LZMA` et `BZIP2`.

Pour chacune des tailles de l'alphabet, nous avons mesuré la complexité des cent chaînes grâce aux cinq méthodes que nous avons listées ci-dessus. Nous calculons alors la moyenne de complexité pour les cent chaînes ayant la même taille d'alphabet. Les résultats obtenus

3. Nous avons choisi la limite de 30000 caractères, car au delà tous les symboles ne rentrent pas dans le buffer de `GZIP` et fausserait donc les résultats



(a) $l^\star$=2. (b) $l^\star$=5.

FIGURE II.12 – Évolution de la complexité en fonction de la longueur moyenne des symboles de la décomposition de SALZA.

sont représentés en fonction des longueurs moyennes des symboles de la décomposition de SALZA pour les différentes tailles d'alphabet. Les résultats sont présentés sur la figure II.12.

La complexité de Lempel-Ziv (courbe verte sur la figure II.12) se comporte de manière similaire à ce que nous avions étudié en théorie dans la section II.4.1 (courbe bleue sur la figure II.9). Une discrimination forte apparaît entre les chaînes dont la longueur moyenne des symboles est faible. Par contre, cette discrimination est assez faible si la longueur moyenne des symboles est grande. Par exemple la différence de complexité entre une chaîne dont la longueur moyenne des symboles est 1,96 et une autre chaîne dont la longueur moyenne des symboles est 1,97 est presque la même que la différence de complexité entre une chaîne dont la longueur moyenne des symboles est 7 et une autre chaîne dont la longueur moyenne des symboles est 8. Comme la complexité de Lempel-Ziv ne dépend pas d'une longueur seuil $l^\star$, ce manque de discrimination peut être gênant. En effet, dans un exemple où toutes les données ont des longueurs d'opérations `copier` assez longues en moyenne, la complexité de Lempel-Ziv sera pratiquement identique pour toutes les chaînes.

Les compresseurs (courbes rouge, bleue ciel et violette de la figure II.12) se comportent approximativement comme la complexité de Lempel-Ziv à une constante près. La taille du fichier compressé pour GZIP n'évolue pas entre la longueur moyenne 2 et 3. Comme nous l'avions vu dans la section I.3.1.1, ce compresseur ne peut pas faire d'opérations `copier` de longueur 2. C'est pourquoi tant que les longueurs moyennes sont entre 2 et 3, la compression de la chaîne n'est pas améliorée.

Enfin, nous pouvons observer que SALZA (courbe bleue de la figure II.12) est centrée autour de sa longueur seuil $l^\star$. C'est autour de cette longueur qu'elle a le plus grand pouvoir discriminant. Ainsi pour $l^\star$=2, elle discrimine beaucoup les chaînes qui ont des longueurs moyennes proches de 2 et moins les autres. On peut aussi noter qu'à partir de $\lambda = 4$ la complexité de Lempel-Ziv et SALZA se rejoignent, car la fonction admissible exponentielle se comporte alors comme la fonction indicatrice. Pour une longueur seuil $l^\star$=5, SALZA fait très peu de différence entre des chaînes qui ont en moyenne des opérations `copier` de taille 2 ou 3 mais discrimine beaucoup les chaînes qui ont en moyenne des opérations `copier` de taille 4 ou 5.

En résumé

Comparaison des différents estimateurs de complexité.

- *La complexité de Lempel-Ziv* : elle discrimine beaucoup deux chaînes dont les longueurs des opérations `copier` sont très faibles mais elle discrimine très peu celles dont les longueurs des opérations `copier` sont très grandes.
- *Les compresseurs* : leurs comportement est proche de celui de la complexité de Lempel-Ziv à une constante près et ils tendent difficilement vers 0.
- *SALZA* : pour une longueur seuil $l^\star$ faible, SALZA se comporte de manière similaire à la complexité de Lempel-Ziv, mais pour une longueur seuil élevée, elle permet de vraiment discriminer deux chaînes ayant des longueurs d'opérations `copier` proches de $l^\star$, ce qui permet d'être plus robuste aux petites longueurs et de s'adapter aux données.

Conclusion

Nous avons proposé un nouvel estimateur de complexité de Kolmogorov basé sur la complexité de Lempel-Ziv. Cet estimateur appelé SALZA est implémenté sur les bases de l'algorithme GZIP basé lui-même sur la complexité de Lempel-Ziv. Nous avons dans un premier temps clarifié les différents types de conditionnements qui peuvent être établis dans la décomposition de Lempel-Ziv. Puis nous avons mis en évidence qu'il est utile de ne pas simplement compter le nombre de symboles de la décomposition comme le fait la complexité de Lempel-Ziv, mais de prendre aussi en compte leur longueur. Nous avons donc introduit les fonctions admissibles qui attribuent un score à chaque longueur des opérations `copier`. Plus la longueur de l'opération `copier` est grande, plus le score est proche de 1, plus l'opération `copier` est considérée comme significative.

En partant de la décomposition de GZIP, nous proposons l'estimateur SALZA basé sur le nombre de symboles comme dans la complexité de Lempel-Ziv mais également sur les longueurs des opérations `copier` significatives. Nous avons prouvé que SALZA est une mesure d'information pour la fonction admissible indicatrice. Il est possible de trouver des contre-exemples pour les autres fonctions admissibles. En pratique, nous avons cependant observé que SALZA respecte empiriquement les propriétés des mesures d'information.

Il est possible de définir l'estimateur de la complexité jointe basée sur SALZA. Nous avons vu qu'il peut exister plusieurs définitions de SALZA jointe selon la manière dont nous interprétons la meilleure décomposition de X et Y. Cependant, les résultats obtenus en comparant ces définitions sont assez proches en pratique. Grâce à ces définitions, il a alors été facile de définir l'information mutuelle algorithmique qui mesure la dépendance entre les deux chaînes de caractères X et Y. Puis nous avons aussi défini l'information dirigée

algorithmique qui mesure la dépendance directionnelle entre ces deux chaînes.

Enfin nous avons comparé le comportement de SALZA avec la complexité de Lempel-Ziv et les compresseurs. Dans un premier temps, nous avons réalisé une étude théorique en supposant une distribution sur la longueur des symboles. Nous avons établi que pour une longueur seuil $l^\star$ de la fonction admissible, SALZA et la complexité de Lempel-Ziv se comportent de manière similaire. Cependant si nous augmentons la longueur seuil $l^\star$, SALZA devient très discriminant autour de cette longueur. Cette propriété de SALZA présente l'avantage de permettre une discrimination entre deux chaînes plus fine autour de $l^\star$ que la complexité de Lempel-Ziv qui elle ne discrimine que les petites longueurs. Pour pouvoir comparer les compresseurs à SALZA, nous avons réalisé une simulation. Nous avons pu voir que l'estimation de la complexité par les compresseurs était proche de la celui de la complexité de Lempel-Ziv à une constante près.

SALZA apparaît donc comme un bon estimateur de complexité. Nous verrons dans les prochains chapitres comment l'appliquer à des problèmes de classification et d'inférence de causalité.

Chapitre III

Classifier grâce à la complexité

Sommaire

Introduction

Au cours des chapitres précédents, nous avons présenté la complexité de Kolmogorov et la complexité de Lempel-Ziv. Nous avons ensuite introduit `SALZA`, un nouvel estimateur de la complexité de Kolmogorov. Dans ce chapitre nous allons montrer comment utiliser la complexité à des fins de classification. Comme nous l'avons vu dans le chapitre I, la complexité est une mesure absolue de l'information, c'est-à-dire qu'elle ne dépend pas de l'ensemble auquel appartient la chaîne de caractères. La complexité d'une chaîne de nucléotides qui forment un ADN se calcule de la même façon que la complexité d'un texte numérisé de la littérature française. Ainsi la complexité permet de faire une classification totalement indépendante du type de données à classifier ou d'un modèle probabiliste les représentant.

Dans un premier temps (section III.1), nous présentons les différentes métriques entre chaînes basées sur la complexité de Lempel-Ziv ou la complexité de Kolmogorov. Puis nous introduisons une semi-distance entre chaînes de caractères basée sur `SALZA`. Dans un second temps (section III.2), nous expliquons la méthode de classification ascendante hiérarchique qui permet de classifier des données à partir d'une matrice de dissimilarité. Enfin (section III.3), nous calculons des matrices de dissimilarité grâce aux différentes métriques basées sur la complexité que nous avons étudiées précédemment. Ce qui nous permet alors d'appliquer une classification ascendante hiérarchique.

Dans ce chapitre, nous allons tenter de classifier quatre jeux de données. Tout d'abord, nous essayons de regrouper des chaînes de Markov qui sont générées à partir de différentes matrices de transition. Nous classifions ensuite des ADN de mammifères et nous essayons de retrouver les différentes familles génétiques. Les deux derniers jeux de données sont des textes numérisés. Le premier est la Déclaration Universelle des Droits de l'Homme écrite en différente langue. L'objectif est alors de retrouver les différentes familles de langue. Le deuxième est un ensemble de romans de la littérature française, nous cherchons à regrouper les œuvres par auteur.

III.1 Métriques basées sur la complexité

Le calcul de la complexité d'une chaîne ne dépend pas de sa nature. Il est possible de calculer la complexité d'une séquence d'ADN de la même façon que nous calculons la complexité d'un fichier de musique MIDI ou d'un texte littéraire numérisé. Cela permet de classifier sans aucune connaissance particulière du domaine d'application ou sans émettre un modèle probabiliste sur le jeu de données. Nous allons donc présenter quatre métriques qui reposent sur la complexité et qui permettent de classifier de façon non paramétrique tout ensemble de chaînes, quelle que soit la provenance des chaînes.

Il existe d'autres approches basées sur la compression et les dictionnaires comme [BCL02], [CKV06] ou encore [Mac+08]. Ces méthodes étant bien présentées dans la littérature, nous n'allons pas les reprendre ici.

Rappelons qu'une métrique n'est une distance que si elle respecte ces quelques propriétés :

Propriété III.1.1 (Distance)**.**
Soit X, Y et Z trois chaînes définies sur $\mathcal{A}^*$, alors la métrique $D : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}^+$ est une distance si et seulement si elle respecte les propriétés suivantes :

1. *Positivité : $D(X,Y) \geq 0$,*
2. *Symétrie : $D(X,Y) = D(Y,X)$,*
3. *Égalité des indiscernables : $D(X,Y) = 0$ si et seulement si $X = Y$,*
4. *Inégalité triangulaire : $D(X,Z) \leq D(X,Y) + D(Y,Z)$.*

Une métrique n'étant pas forcément une distance, il en existe d'autres types. Si aucune des propriétés d'une distance n'est respectée alors on dit que la métrique est une "divergence". Si la symétrie (propriété III.1.1) n'est pas respectée alors on dit que la métrique est une "quasi-distance". Si l'égalité des indiscernables n'est pas respectée alors on dit que la métrique est une "pseudo métrique". Si l'inégalité triangulaire n'est pas respectée alors on dit que la métrique est une "semi-distance". Si la métrique ne respecte pas plusieurs propriétés, alors il est possible de combiner les termes. Ainsi si la métrique ne respecte pas les propriétés de symétrie et l'égalité des indiscernables alors on dit que la métrique est une "quasi-pseudo-distance"

III.1.1 Une divergence basée sur la complexité de Lempel-Ziv

En 1993, Ziv et Merhav proposent une nouvelle divergence entre deux chaînes dans [ZM93]. Cette divergence est calculée à l'aide de la complexité de Lempel-Ziv non conditionnelle et de la complexité de Lempel-Ziv conditionnelle. Comme nous l'avons vu dans les sections I.2.3.1 et II.1.2, pour eux, la complexité de X connaissant Y correspond au conditionnement $|^+$. De plus ils supposent que $l(X) = l(Y) = n$.

Définition III.1.1 (divergence de Ziv-Merhav)**.**
Soient X et Y deux chaînes de $\mathcal{A}^n$, alors la divergence de Ziv-Merhav est :

$$\Delta(X||Y) = \frac{1}{n}\left[L(X|^+Y)log_\alpha n - L(X)log_\alpha L(X)\right]. \tag{III.1}$$

La divergence de Ziv-Merhav est la différence de complexité entre X qui n'est expliquée que par Y et X qui s'explique par lui-même. Cette métrique est bien une divergence car elle ne respecte aucune des propriétés III.1.1. Les contre-exemples pour chacune des propriétés sont présentés en annexe C. La divergence de Ziv-Merhav a par exemple été utilisée pour classifier des textes [CF05].

III.1.2 Une distance basée sur la complexité de Kolmogorov

En 1998, Bennett et al. [Ben+98] proposent une distance qui ne nécessite aucune connaissance spécifique au domaine d'application. Cette distance est la taille du plus petit programme qui génère X à partir de Y ainsi que Y à partir de X. Elle repose donc logiquement sur la complexité de Kolmogorov. Ils ont alors montré que cette distance est donnée par :

$$ID(Y,Y) = max\{K(X|Y), K(Y|X)\}.$$

Cependant, une distance normalisée est plus pertinente. En effet, si deux chaînes sont très petites alors il y a de fortes chances que leur complexité et donc leur distance soient plus faibles que pour deux chaînes cent fois plus grandes. Donc si nous voulons pouvoir comparer des chaînes de différentes longueurs, il faut que la distance entre ces chaînes soit indépendante de la taille de la chaîne. C'est pourquoi Li et Vitányi proposent dans [Li+04] une normalisation de cette distance appelée Normalized Information Distance (*NID*).

Définition III.1.2 (Normalized Information Distance (*NID*))**.**
Soient X et Y deux chaînes dans $\mathcal{A}^*$, alors la Normalized Information Distance est :

$$NID(X,Y) = \frac{max\{K(X|Y), K(Y|X)\}}{max\{K(X), K(Y\}}. \tag{III.2}$$

La *NID* est la taille du plus petit programme qui génère X à partir de Y et Y à partir de X normalisée par la taille du plus petit programme capable de générer X ou Y. Comme nous l'avons vu dans le chapitre I, la plupart des propriétés de la complexité de Kolmogorov ne sont valables qu'à une constante près. Ainsi la *NID* n'est une distance qu'à une constante près comme l'ont montré Li et Vitányi dans [Li+04].

III.1.3 Une semi-distance basée sur les compresseurs

La *NID* reposant sur la complexité de Kolmogorov, elle n'est pas calculable en un temps fini. On ne peut donc pas l'appliquer sur des données réelles. C'est pourquoi Cilibrasy et Vitányi [Li+04] proposent d'estimer la complexité de Kolmogorov à l'aide de compresseurs. La Normalized Information Distance devient alors la Normalized Compression Distance (*NCD*). Cependant comme nous l'avons vu dans la section I.3.2, il n'est pas possible de compresser une chaîne X connaissant une autre chaîne Y. Il est donc nécessaire d'utiliser une approximation pour calculer la complexité conditionnelle avec un compresseur : $C(X|Y) = C(YX) - C(Y)$ (voir section I.3.2, équation I.12).

Définition III.1.3 (Normalized Compression Distance (*NCD*))**.**
Soient X et Y deux chaînes dans $\mathcal{A}^*$, alors la Normalized Compression Distance est :

$$NCD(X,Y) = \frac{C(XY) - min\{C(X), C(Y)\}}{max\{C(X), C(Y\}}. \tag{III.3}$$

La taille de la compression de Y et X concaténées à laquelle on soustrait la taille de la plus petite des deux compressions de X et Y. Le désavantage de cette méthode est que la *NCD* n'est plus alors qu'une "pseudo-quasi-semi-distance" (pour la démonstration et les contre-exemples, voir dans l'annexe C).

Cette métrique a été utilisée dans de nombreux domaines d'application comme la classification d'ADN ([CV05]), de langues [Li+04] ou même encore de musique [FDK15].

III.1.4 Une semi-distance basée sur `SALZA`

Comme nous l'avons vu dans la section II.4.2, `SALZA` donne de meilleurs résultats de discrimination que les compresseurs. Ainsi, au lieu d'estimer la complexité de Kolmogorov à l'aide de compresseurs dans la *NID*, il est possible d'utiliser `SALZA`. La Normalized Information Distance devient alors la Normalized `SALZA` Distance (*NSD*). Pour cela nous devons nous poser deux questions.

i La première porte sur le type de conditionnement que nous voulons utiliser. Dans le cas d'une mesure de ressemblance nous voulons savoir si X ressemble à Y et non si X est redondant. C'est pourquoi nous optons pour le conditionnement $|^+$ comme Ziv et Merhav. Le choix de ce conditionnement change aussi la normalisation. Dans ce cas-là, $S_f(X|Y)$ n'est plus borné par $S_f(X)$ mais par $l(X)$. En effet si par exemple X et Y n'ont aucun symbole d'alphabet commun, alors nous devons réaliser des opérations `insérer` tout au long de la décomposition.

ii La deuxième question pose le choix de la longueur seuil, $l^\star$. Il est tout à fait possible que l'utilisateur choisisse lui-même la longueur seuil, mais l'objectif est de faire de la classification sans aucun a priori sur les données. Soit $\mathcal{X}$, l'ensemble des chaînes pour lesquelles nous calculons une distance, nous réalisons toutes les décompositions $|^+$ possibles. Alors $l^\star$ est la moyenne de ces longueurs. Ainsi, si une référence est plus longue que la longueur moyenne du jeu de données alors pour ce jeu de données (et

uniquement ce jeu de données), elle est *significative*, à l'inverse si elle est plus petite cela signifie qu'elle n'est pas *significative* pour ce jeu de données.

Définition III.1.4 (Normalized `SALZA` Distance (*NSD*))**.**
Soient X et Y deux chaînes dans $\mathcal{A}^*$, alors la Normalized ***`SALZA`*** *Distance est :*

$$NSD(X,Y) = \frac{max\{S_f(X|^+Y), S_f(Y|^+X)\}}{max\{l(X), l(Y\}}. \tag{III.4}$$

La *NSD* est la plus grande décomposition qui permet de décomposer X juste avec Y ou inversement.

Contrairement à la *NCD*, la *NSD* est une semi-distance (la démonstration se trouve dans l'annexe C). Reprenons ici le contre-exemple de l'inégalité triangulaire pour le détailler et comprendre pourquoi la *NSD* ne respecte pas l'inégalité triangulaire. Pour rappel $(X)^n$ signifie que nous répétons n fois la chaîne X. Pour inverser les caractères de la chaîne X, nous notons $\overline{X} = x_n x_{n-1}...x_2 x_1$. Soit trois alphabets $\mathcal{A}_A$, $\mathcal{A}_B$ et $\mathcal{A}_C$ distincts de taille α, i.e. $\mathcal{A}_A \cap \mathcal{A}_B = \mathcal{A}_A \cap \mathcal{A}_C = \mathcal{A}_B \cap \mathcal{A}_C = \emptyset$ et $|\mathcal{A}_A| = |\mathcal{A}_B| = |\mathcal{A}_C| = \alpha$. Prenons trois chaînes A, B et C constituées des α éléments de $\mathcal{A}_A$, respectivement $\mathcal{A}_B$ et $\mathcal{A}_C$. On pose alors :

- $X = (A)^n B\overline{C}$,
- $Y = B(C)^n\overline{A}$,
- $Z = AC(B)^n$.

On a alors pour la fonction admissible indicatrice :

- $S_{\mathbb{1}}(X|^+Y) = S_{\mathbb{1}}(Y|^+X) = (n+1)\alpha$;
- $S_{\mathbb{1}}(X|^+Z) = S_{\mathbb{1}}(Y|^+Z) = S_{\mathbb{1}}(Z|^+Y) = S_{\mathbb{1}}(Z|^+X) = n + \alpha$.

Ce qui donne :

- $NSD_{\mathbb{1}}(X,Y) = \frac{n+1}{n+2}$
- $NSD_{\mathbb{1}}(X,Z) = \frac{n+\alpha}{(n+2)\alpha}$
- $NSD_{\mathbb{1}}(Z,Y) = \frac{n+\alpha}{(n+2)\alpha}$

Si on prend l'application numérique $n = 1000$ et $\alpha = 10$, on a

$$NSD_{\mathbb{1}}(X,Z) + NSD_{\mathbb{1}}(Z,Y) - NSD_{\mathbb{1}}(X,Y) = -0,78 < 0.$$

L'inégalité triangulaire n'est donc pas respectée.

Ce contre-exemple vient du fait que nous avons restreint les opérations de notre programme aux opérations `copier` et `insérer`. Or Lempel et Ziv ont souligné qu'il était possible d'étendre la décomposition d'une chaîne à tout autre type d'opérations. Ainsi en ajoutant l'opération `copier retourner` qui copie puis inverse tous les symboles à notre ensemble des opérations possibles (composé jusqu'à là uniquement des opérations `copier` et `insérer`), l'inégalité triangulaire est alors respectée dans le contre-exemple. Cependant, si nous posons que l'opération $\overline{X}$ n'inverse plus l'ordre des symboles, mais fait une autre permutation, alors le contre-exemple ne respecte plus de nouveau l'inégalité triangulaire. Pour éviter le problème, nous pourrions ajouter la permutation à notre ensemble d'opérations possibles. Cependant, à force de rajouter des opérations possibles, nous tendrons

alors vers la complexité de Kolmogorov et donc `SALZA` ne serait plus calculable en un temps fini. Empiriquement, nous pouvons observer que la *NSD* respecte toujours l'inégalité triangulaire. En effet comme l'ont souligné Lempel et Ziv, les opérations `copier` et `insérer` sont les plus répandues et couvrent la majorité des cas.

En résumé

Les différentes métriques entre chaînes de caractères basées sur la complexité et leurs propriétés :

		Calculable	Potisitve	Symétrique	Indiscernables	Inégalité triangulaire
Δ	divergence de Ziv-Merhav	✓	✗	✗	✗	✗
NID	Normalized Information Distance	✗	✓	✓	✓	✓
NCD	Normalized Compression Distance	✓	✓	✗	✗	✗
NSD	Normalized `SALZA` Distance	✓	✓	✓	✓	✗

III.2 Classification ascendante hiérarchique

Il existe de nombreuses méthodes de classifications à partir d'une matrice de distance. Nous nous sommes concentrés sur la méthode de classification hiérarchique. Étant utilisée par Li et Vitányi [CV05], nous pouvons ainsi comparer les résultats obtenus avec la *NCD* à ceux obtenus avec la *NSD*.

La classification ascendante hiérarchique est une méthode de classification automatique qui répartit les éléments d'un ensemble $\mathcal{X}$ dans différentes classes ([Roh73], [Dia69], [SS+73]). Cette classification s'appuie sur une mesure de dissimilarité entre les éléments à classifier (par exemple la distance euclidienne dans l'espace, mais aussi la *NSD* ou *NCD* pour des chaînes de caractères).

Algorithme 3 Classification ascendante hiérarchique de $\mathcal{X}$

Entrée : D : matrice de dissimilarité entre les éléments de $\mathcal{X}$.

1: $k \leftarrow |\mathcal{X}|$: le nombre de classes.
2: **tant que** $k > 1$ **faire**
3: Chercher dans la matrice de dissimilarité la paire (i, j) la plus proche.
4: Fusionner les classes i et j en une classe h, aussi appelée nœud.
5: Mettre à jour D en supprimant les lignes et colonnes i et j et en rajoutant une ligne et une colonne pour h.
6: **fin tant que**

L'algorithme générique de la classification ascendante hiérarchique d'un ensemble d'éléments $\mathcal{X}$ est présenté dans l'algorithme 3. Nous partons d'une situation où tous les éléments sont seuls dans une classe. En utilisant la matrice de distance D, nous regroupons les deux éléments i et j les plus proches dans une seule et même classe h. La matrice de distance D est alors mise à jour en calculant la distance de la nouvelle classe h aux autres classes. Ce processus est itéré jusqu'à ce qu'il ne reste plus qu'une seule et unique grande classe qui regroupe tous les éléments. La différence entre les algorithmes de classification ascendante hiérarchique réside dans l'étape de la mise à jour de la matrice de dissimilarité D (ligne 5 de l'algorithme 3). En effet, pour ajouter la ligne correspondante à la nouvelle classe h dans D, il faut calculer la distance de h aux autres classes et il existe pour cela différentes méthodes. Soit U et V deux classes contenant respectivement u et v éléments, voici une liste non exhaustive des méthodes qui peuvent exister :

Nearest Point Algorithm [Roh73] : $D(U,V) = \min_{i=1..u\ ,\ j=1..v} D(U[i],V[j])$,

Farthest Point Algorithm [Dia69] : $D(U,V) = \max_{i=1..u\ ,\ j=1..v} D(U[i],V[j])$,

UPGMA [SS+73] : $D(U,V) = \sum_{i=1..u\ ,\ j=1..v} \frac{D(U[i],V[j])}{u\ v}$.

Nous avons retenu pour nos travaux la méthode UPGMA car c'est une des méthodes utilisées par Li et Vitányi [CV05]. Nous allons illustrer la classification ascendante hiérarchique par la méthode UPGMA à travers un exemple. Pour cela nous avons cinq éléments appelés a, b, c, d et e. Nous supposons une matrice de distance factice entre ces éléments. La méthode UPGMA est alors appliquée pour classifier ces cinq éléments. Si nous utilisons une autre méthode de classification hiérarchique, alors ce sera l'étape "mise à jour de la matrice" qui changera.

Étape 1

Matrice de distance :

$$\mathcal{D}_1 = \begin{array}{c} \begin{array}{ccccc} a & b & c & d & e \end{array} \\ \begin{pmatrix} 0 & 17 & 21 & 31 & 23 \\ 17 & 0 & 30 & 34 & 21 \\ 21 & 30 & 0 & 28 & 39 \\ 31 & 34 & 28 & 0 & 43 \\ 23 & 21 & 39 & 43 & 0 \end{pmatrix} \begin{array}{c} a \\ b \\ c \\ d \\ e \end{array} \end{array}$$

Regroupement : Les deux éléments les plus proches sont a et b. Ils sont regroupés dans une même classe u.

Distance au cluster : $\delta(a,u) = \delta(b,u) = \mathcal{D}_1(a,b)/2 = 8.5$.

Mise à jour de la matrice :

- $\mathcal{D}_2(u,c) = \mathcal{D}_2((a,b),c) = \frac{1\times\mathcal{D}_1(a,c)+1\times\mathcal{D}_1(b,c)}{1+1} = 25,5$
- $\mathcal{D}_2(u,d) = \mathcal{D}_2((a,b),d) = \frac{1\times\mathcal{D}_1(a,d)+1\times\mathcal{D}_1(b,d)}{1+1} = 32,5$
- $\mathcal{D}_2(u,e) = \mathcal{D}_2((a,b),e) = \frac{1\times\mathcal{D}_1(a,e)+1\times\mathcal{D}_1(b,e)}{1+1} = 22$

Étape 2

Matrice de distance :

$$\mathcal{D}_2 = \begin{array}{c} \begin{matrix} u & c & d & e \end{matrix} \\ \begin{pmatrix} 0 & 25,5 & 32,5 & 22 \\ 25,5 & 0 & 28 & 39 \\ 32,5 & 28 & 0 & 43 \\ 22 & 39 & 43 & 0 \end{pmatrix} \begin{matrix} u \\ c \\ d \\ e \end{matrix} \end{array}$$

Regroupement : Les deux éléments les plus proches sont $u = (a, b)$ et e. Ils sont regroupés dans une même classe v.

Distance au cluster : $\delta(a, v) = \delta(b, v) = \delta(e, v) = \mathcal{D}_2(u, e)/2 = 11.$

Mise à jour de la matrice :

- $\mathcal{D}_3(v, c) = \mathcal{D}_3(((a, b), e), c) = \frac{2\times\mathcal{D}_2(u,c)+1\times\mathcal{D}_2(e,c)}{2+1} = 30$
- $\mathcal{D}_3(v, d) = \mathcal{D}_3(((a, b), e), d) = \frac{2\times\mathcal{D}_2(u,d)+1\times\mathcal{D}_2(e,d)}{2+1} = 36$

Étape 3

Matrice de distance :

$$\mathcal{D}_3 = \begin{array}{c} \begin{matrix} v & c & d \end{matrix} \\ \begin{pmatrix} 0 & 30 & 36 \\ 30 & 0 & 28 \\ 36 & 28 & 0 \end{pmatrix} \begin{matrix} v \\ c \\ d \end{matrix} \end{array}$$

Regroupement : Les deux éléments les plus proches sont c et d. Ils sont regroupés dans une même classe w.

Distance au cluster : $\delta(c, w) = \delta(d, w) = \mathcal{D}_3(c, d)/2 = 14.$

Mise à jour de la matrice :

- $\mathcal{D}_4(v, w) = \mathcal{D}_4(((a, b), e), (c, d)) = \frac{1\times\mathcal{D}_3(c,v)+1\times\mathcal{D}_3(d,v)}{1+1} = 33$

Étape 4

Matrice de distance :

$$\mathcal{D}_4 = \begin{array}{c} \begin{matrix} v & w \end{matrix} \\ \begin{pmatrix} 0 & 33 \\ 33 & 0 \end{pmatrix} \begin{matrix} v \\ w \end{matrix} \end{array}$$

Regroupement : Les deux éléments les plus proches sont v et w. Ils sont regroupés dans une même classe r qui est la racine de la classification. Après cette étape nous aurons plus qu'un seul et unique classe.

Distance au cluster : $\delta(v, r) = \delta(r, w) = \mathcal{D}_4(w, v)/2 = 16,5.$

Un des avantages de la classification ascendante hiérarchique est qu'il est possible de la représenter graphiquement. Pour cela, nous utilisons un dendogramme qui est un graphique en forme d'arbre. La figure III.1 est la représentation de la classification des cinq éléments que nous venons de classifier dans l'exemple précédent. Tous les éléments de $\mathcal{X}$ sont appelés les feuilles et sont représentés ici à droite de la figure. Puis les branches se regroupent en nœuds qui forment les classes. Pour savoir quels éléments appartiennent à la classe, il suffit de redescendre toutes les branches vers la droite en direction des feuilles. Par exemple, considérons la classe u dans la figure III.1. En descendant les branches vers la droite jusqu'aux feuilles nous trouvons que la classe u est composée des deux éléments (a, b). La classe r est, quant à elle, composée des cinq éléments (a, b, c, d, e). Si deux classes ont une feuille en commun, c'est que l'une est incluse dans l'autre.

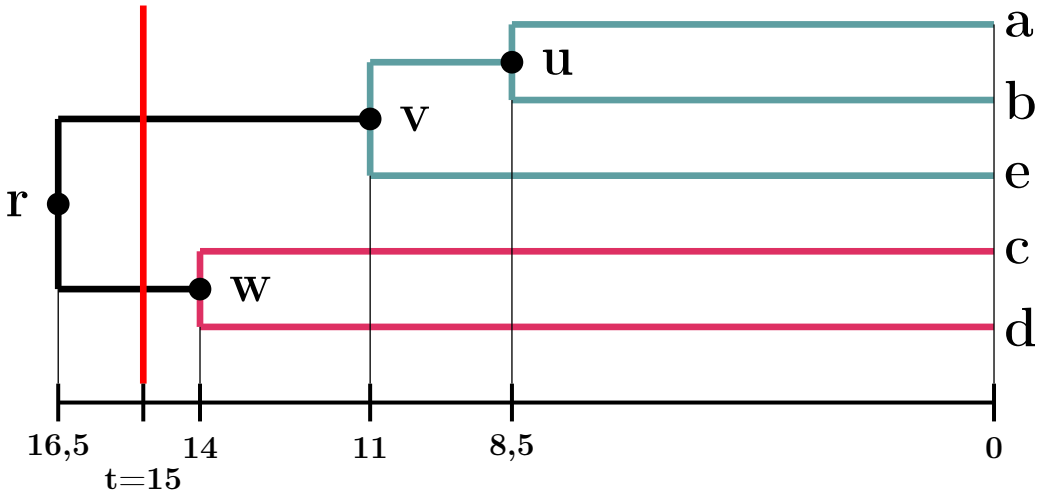


FIGURE III.1 – Dendogramme de la classification ascendante hiérarchique par la méthode UPGMA pour les cinq éléments a, b, c, d et e.

Une fois que la classification est obtenue, nous pouvons vouloir obtenir un nombre particulier de classes qui peut varier d'une seule et unique classe à autant de classes que d'éléments à classifier. Pour cela, il suffit d'utiliser un seuil t qui est la distance à laquelle nous coupons l'arbre. Les classes qui ont une distance à leurs éléments inférieure à t sont alors les classes retenues pour la classification. Si $t = 0$, alors chaque élément forme une classe à lui seul. Si le seuil est infini, alors tous les éléments forment une seule et unique classe. Par exemple, si nous posons un seuil $t = 15$ dans l'exemple précédent, on trouve deux classes : (a, b, e) et (c, d). Pour représenter ces deux classes, deux couleurs différentes sont utilisées dans la figure III.1.

Pour réaliser une classification et afficher le dendogramme associé, nous utilisons la bibliothèque python *scipy* et les fonctions de *Hierarchical clustering* [JOP+01]. Nous calculons les matrices de distances grâce à Complearn ([Cil+16]) pour la *NCD* et grâce à notre bibliothèque C pour la *NSD* et la divergence de Ziv-Merhav. Puis nous utilisons la fonction de *scipy* qui réalise la classification ascendante hiérarchique et affiche un dendogramme. Nous utilisons les paramètres par défauts de cette fonction hormis pour la méthode de classification où nous spécifions *'average'* qui correspond à la méthode UPGMA.

En résumé

Classification ascendante hiérarchique

- *Principe* : au début chaque élément forme une classe à lui seul, puis les deux éléments les plus proches sont rassemblés dans une nouvelle classe jusqu'à ce qu'il n'y ait plus qu'une seule et unique classe regroupant tous les éléments.
- *Représentation* : La classification ascendante hiérarchique est représentée par un dendogramme qui est un arbre où toutes les feuilles sont les éléments à classifier et chaque nœud est une classe.
- *Implémentation* : nous utilisons la bibliothèque python *scipy* et les fonctions de *Hierarchical clustering*.

III.3 Résultats et comparaisons

Nous allons maintenant appliquer la classification ascendante hiérarchique à différents jeux de données [1] en utilisant les différentes métriques que nous avons vues dans la section III.1. Pour chacune des données nous afficherons trois dendogrammes basés sur :

- La divergence de Ziv-Merhav.
- La *NCD* basée sur la compresseur `LZMA`. Comme le soulignent Cerbián et al. [CRAO05], nous ne pouvons pas utiliser n'importe quel compresseur selon les données que nous manipulons. Une des contraintes principales est la taille du buffer (voir section I.3.1) qui ne doit pas être inférieure à deux fois la taille du fichier. Nos données ayant une taille maximale de 2MO, `GZIP` ne fonctionnerait pas correctement, car son buffer a une taille de 32kO. C'est pourquoi nous utilisons `LZMA` qui a une taille de buffer de 4MO.
- La *NSD* utilisant la fonction admissible exponentielle $\mathbb{E}$. En effet comme nous l'avons vu dans la section II.4.1, la fonction admissible exponentielle est celle qui a le plus grand pouvoir discriminant.

La bibliothèque *scipy* requiert une matrice de dissimilarité symétrique, positive et avec des zéros sur la diagonale. Cependant, comme nous l'avons vu, les métriques basées sur la complexité ne sont pas des distances. Il faut donc apporter quelques modifications aux matrices pour qu'elles soient classifiées par la bibliothèque *scipy*.

- La divergence de Ziv-Merhav ne respecte ni la positivité, ni la symétrie, ni l'égalité des indiscernables. Il est donc nécessaire de rajouter une constante pour qu'elle soit positive et de forcer les zéros sur la diagonale. Pour la rendre symétrique, nous utilisons la valeur maximale.
- La *NCD* ne respecte pas la symétrie et l'égalité des indiscernables. La bibliothèque Complearn ([Cil+16]) fournit une implémentation de la *NCD* et rend symétrique la matrice en prenant le maximum et ajoute des zéros sur la diagonale.

1. Pour une description plus détaillée des données aller voir en annexe D

- La *NSD* respecte toutes les propriétés imposées par *scipy*, il n'est donc pas nécessaire de changer la matrice.

De part ses propriétés, la *NSD* est donc la seule matrice à pouvoir être utilisée par la bibliothèque *scipy* sans aucune modification.

III.3.1 Classification de chaînes de Markov

Nous allons maintenant classifier des chaînes de Markov sur un alphabet de taille 4, 64 ou 256. Pour chacune des tailles d'alphabet, nous générons aléatoirement six matrices de probabilités de transition différentes. Pour chacune de ces matrices, nous générons six chaînes de taille 15000 avec six initialisations différentes. Le nom d'une chaîne de Markov dans le jeu de données est de la forme *alphai_mj_ck*, où `i` est la taille de l'alphabet, `j` est le numéro de la matrice de transition qui a servi à générer la chaîne et `k` est le numéro de la réalisation. Le but de la classification est de réussir à regrouper les différentes chaînes issues d'une même matrice de transition.

Données : chaînes de Markov

Caractéristique : chaîne de Markov avec différentes matrices de transition.

Longueur des chaînes : 15 kilo-octets.

Taille de l'alphabet, α : 4, 64 ou 256.

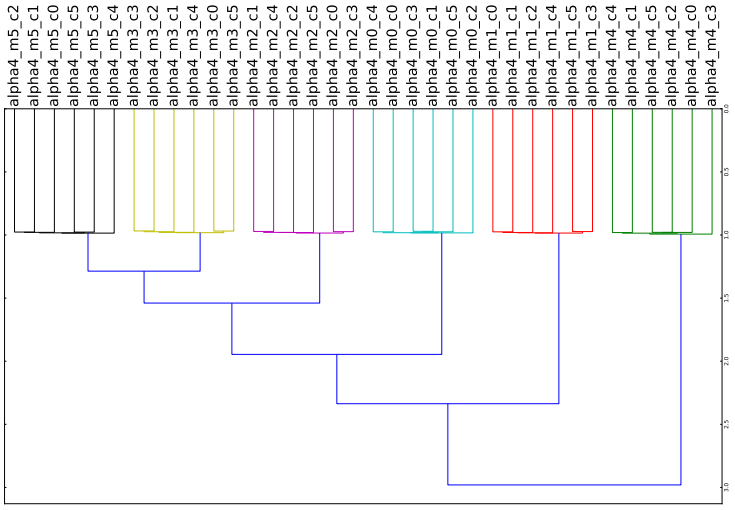
Nom de la chaîne : *alphai_mj_ck*, où `i` est la taille de l'alphabet, `j` est le numéro de la matrice de transition qui a servi à générer la chaîne et `k` est le numéro de la réalisation.

Objectif : regrouper les chaînes issues de la même matrice de transition.

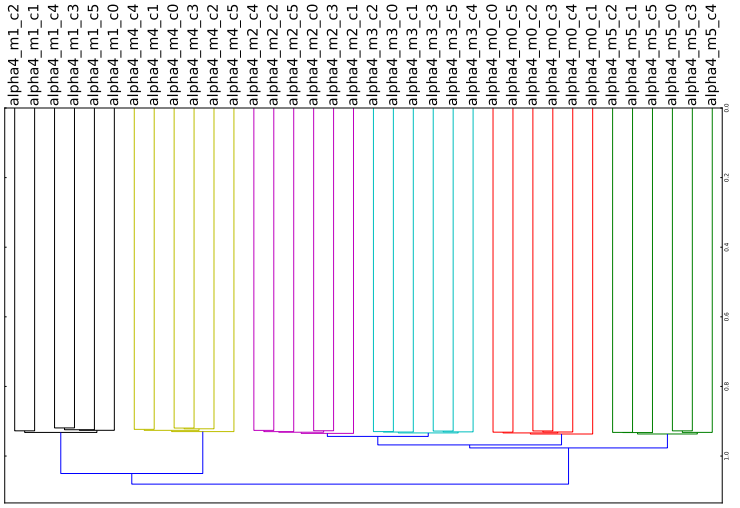
Sur les figures III.2, III.3 et III.4, nous pouvons observer les classifications ascendantes hiérarchiques des chaînes de Markov obtenues grâce aux différentes métriques basées sur la complexité : la divergence de Ziv-Merhav, la *NCD* basée sur le compresseur `LZMA` et la *NSD* basée sur la fonction admissible exponentielle.

Nous utilisons le seuil t comme un paramètre de contrainte et le plaçons manuellement pour chacune des classifications ascendantes hiérarchiques que nous avons obtenues. Dans cette section, nous plaçons t de manière à avoir six classes distinctes qui sont sensées représenter les six matrices de transitions. Dans la classification des chaînes sur un alphabet de taille 4, pour la divergence de Ziv-Merhav, le seuil peut être placé sous cette contrainte entre $1,05$ et $1,3$. Pour la *NCD*, le seuil doit être de $0,94$ si nous voulons respecter la contrainte des six classes. Enfin, de part son grand pouvoir discrimant, le seuil pour la *NSD* doit être en $0,8$ et $0,39$.

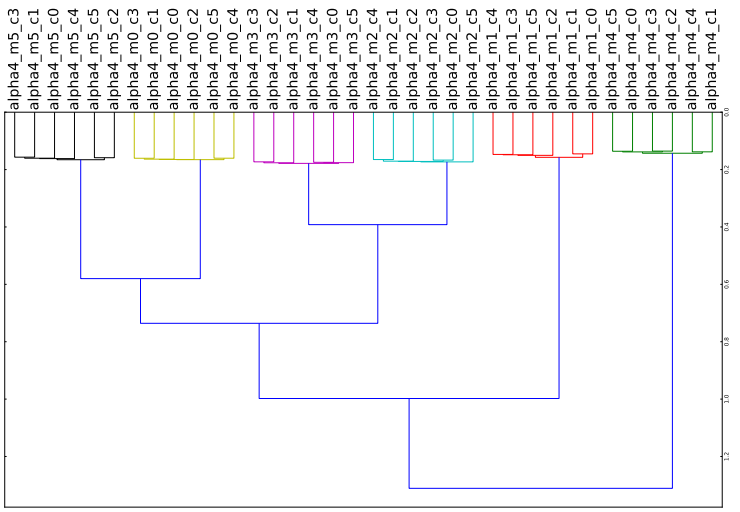
La *NSD* et la divergence de Ziv-Merhav classifient extrêmement bien les chaînes de Markov. Pour toutes les tailles d'alphabet, nous retrouvons les six classes correspondant aux six matrices de transition. De plus il est assez facile de placer empiriquement un seuil t pour déterminer les classes, par exemple entre $0,18$ et $0,39$ pour la *NSD* et $\alpha = 4$. Par contre la *NCD* arrive à regrouper difficilement les chaînes ayant les mêmes matrices



(a) Divergence de Ziv-Merhav, $t = 1,1$.

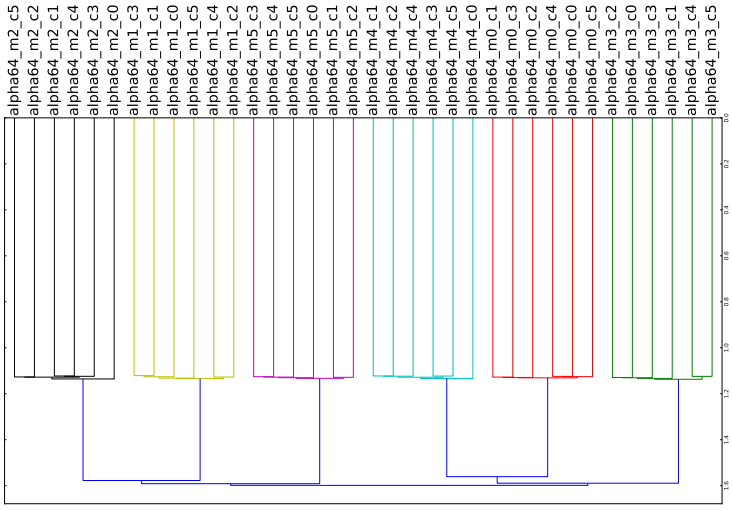


(b) $NCD_{\texttt{LZMA}}$, $t = 0,94$.

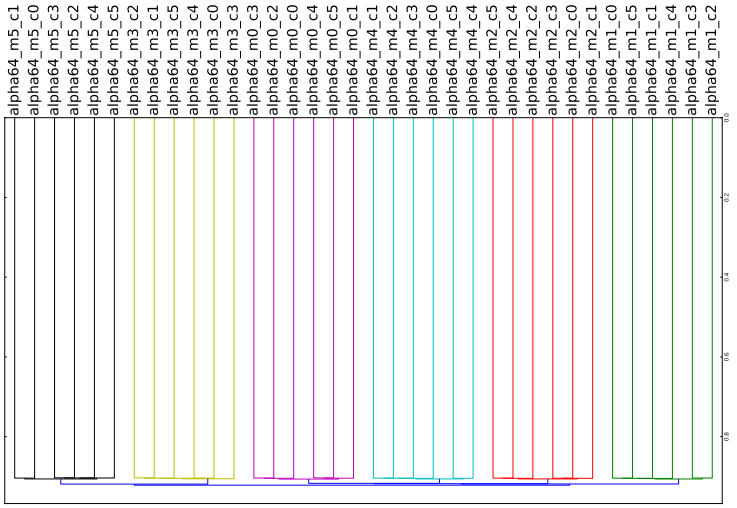


(c) $NSD_{\mathbb{E}}$, $t = 0,2$.

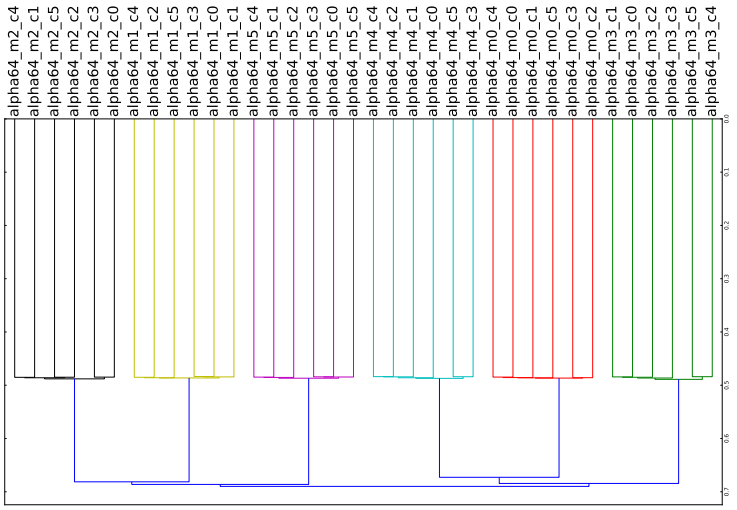
FIGURE III.2 – Classification de chaînes de Markov sur un alphabet de taille 4.



(a) Divergence de Ziv-Merhav, $t = 1,5$.

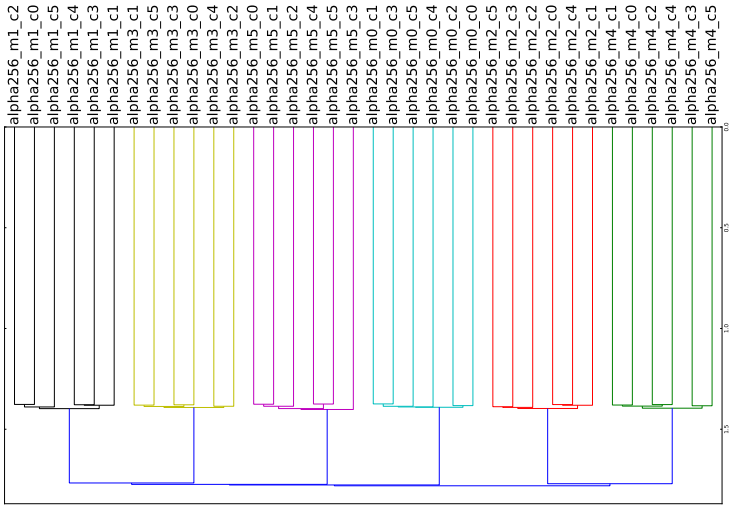


(b) $NCD_{\texttt{LZMA}}$, $t = 0,91$.

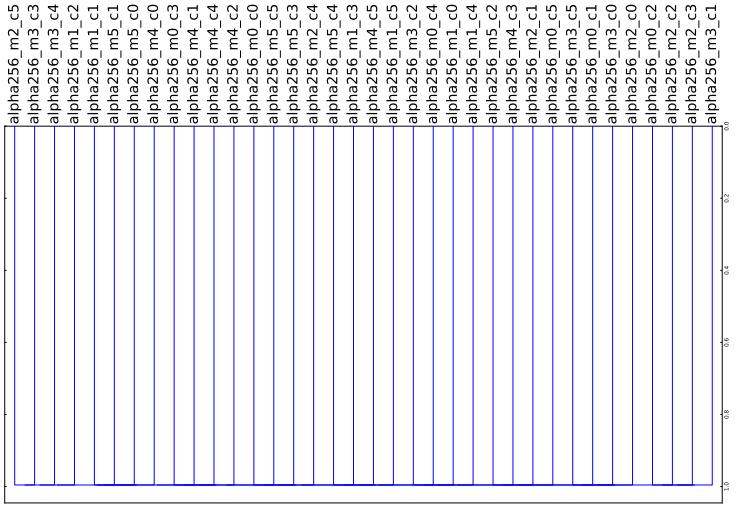


(c) $NSD_{\mathbb{E}}$, $t = 0,6$.

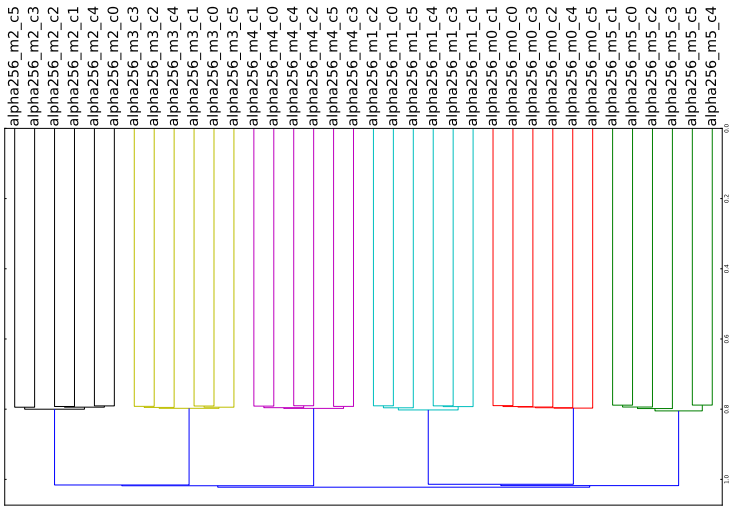
FIGURE III.3 – Classification de chaînes de Markov sur un alphabet de taille 64.



(a) Divergence de Ziv-Merhav, $t = 1,5$.



(b) $NCD_{\texttt{LZMA}}$.



(c) $NSD_{\mathbb{E}}$, $t = 0,9$.

FIGURE III.4 – Classification de chaînes de Markov sur un alphabet de taille 256.

de transition pour $\alpha = 4$ et $\alpha = 64$ mais n'y arrive plus du tout pour une grande taille d'alphabet. De plus, même pour les petites tailles d'alphabet, le seuil est très difficile à placer, il faut être précis au centième près pour le trouver. Il est donc extrêmement difficile de retrouver les classes avec la *NCD* pour toutes les tailles de l'alphabet.

Les différences entre les classifications proviennent de différents facteurs. Tout d'abord il est normal que la divergence de Ziv-Merhav produise de très bons résultats. En effet, Ziv et Merhav ont construit cette divergence pour classifier des chaînes de Markov [ZM93].

Pour comprendre pourquoi la *NCD* a du mal à classifier les chaînes de Markov, nous pouvons regarder la figure III.5. Les compresseurs utilisent l'approximation de la concaténation pour estimer la complexité de Kolmogorov conditionnelle. Ceci est équivalent à la décomposition $\_|^+$. L'histogramme d'une décomposition de deux chaînes provenant de la même matrice de transition avec le conditionnement $\_|^+$ est représenté par la courbe verte dans la figure III.5. La courbe bleue clair représente, quant à elle, celui de deux chaînes provenant de deux matrices de transition distinctes toujours avec le conditionnement $\_|^+$. Dans la *NSD*, nous utilisons le conditionnement $|^+$ pour notre décomposition. Nous avons représenté les deux histogrammes précédents mais avec le conditionnement $|^+$ par les courbes rouge et violette de la figure III.5.

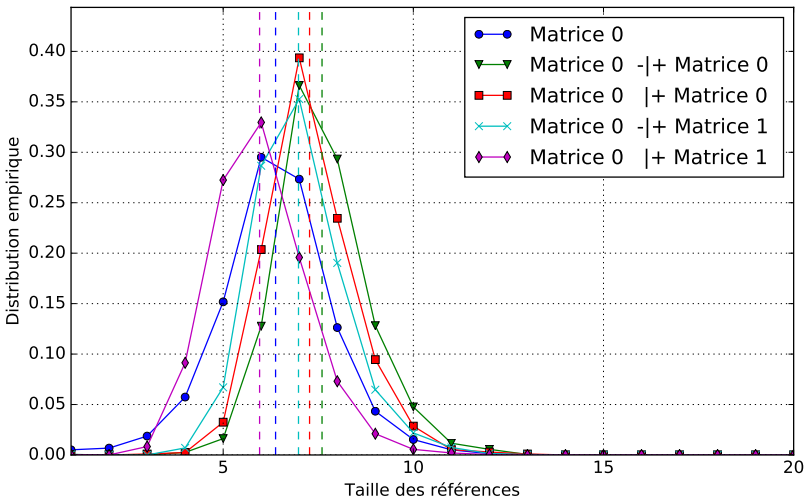


FIGURE III.5 – Histogramme des longueurs des opérations `copier` pour différentes décompositions de chaînes de Markov sur un alphabet de taille 4. La courbe bleue est la décomposition de la chaîne 0 de la matrice 0, *alpha4_m0_c0*. Les courbes verte et rouge sont les décompositions de la chaîne *alpha4_m0_c0* connaissant la chaîne 1 de la même matrice, *alpha4_m0_c1*, avec les conditionnements $\_|^+$ et $|^+$. Les courbes bleue clair et violette correspondent aux décompositions de la chaîne *alpha4_m0_c0* connaissant une chaîne issue d'une autre matrice de transition *alpha4_m1_c0*. Les barres verticales en pointillée représentent la longueur moyenne de chaque décomposition.

Avec la décomposition $\_|^+$, la connaissance d'une chaîne issue de la même matrice de transition n'influence pratiquement pas la distribution des longueurs par rapport à la connaissance d'une chaîne issue d'une matrice de transition différente. Les longueurs moyennes des deux décompositions sont presque identiques (7 pour deux matrices différentes et 7,6 pour deux matrices identiques). Il est donc difficile pour la *NCD* de faire la

différence entre deux chaînes qui appartiennent à la même matrice de transition et deux chaînes ayant des matrices de transition différentes. Dans le cas de la décompositon $|^+$ nous voyons clairement que les décompositions sont différentes. La longueur moyenne passe de 6 quand deux chaînes sont de la même matrice de transition à 7,3 quand les deux chaînes proviennent de matrices de transition différentes. Ainsi la discrimination entre les deux cas (deux chaînes générées à partir de la même matrice de transition et deux chaînes issues de matrices de transition différentes) est très forte pour la *NSD* et assez faible pour la *NCD*.

Un autre facteur, qui peut expliquer pourquoi la *NCD* échoue, est que la compression utilise les symboles dans leur intégralité, c'est-à-dire que les compresseurs prennent en compte les distances dans les symboles `Z` (voir section I.3.1.1). Dans cette exemple, les distances de rappels sont assez aléatoires ainsi le codage arithmétique (voir section I.3.1.1) qui encode les distances est moins efficace que si les distances étaient toujours les mêmes. Or dans la *NSD* nous ne prenons en compte que des longueurs. La compression et donc la *NCD* sont moins performantes.

Enfin, les performances de la *NCD* décroissent quand la taille de l'alphabet augmente à cause de l'apparition d'une quantité non négligeable d'opérations `insérer`. Les opérations `insérer` prennent beaucoup de place à la compression et plus il y en a, moins la compression est bonne. La *NSD* est bien moins sensible à cette augmentation d'opérations `insérer` grâce à la fonction admissible et la longueur seuil $l^\star$.

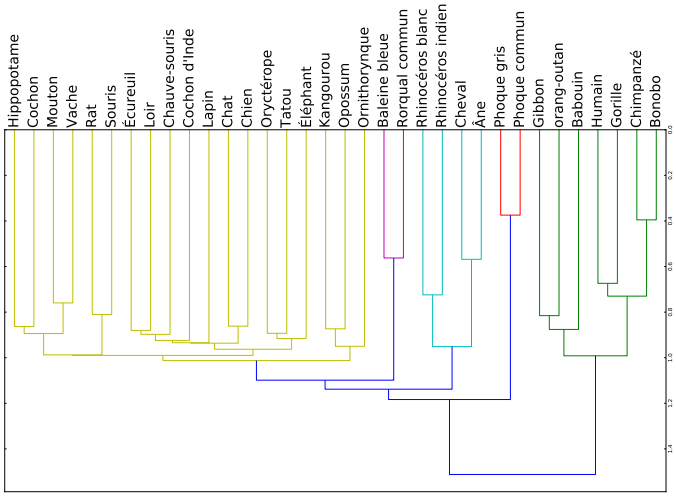
III.3.2 Classification d'ADN de différents mammifères

Nous allons maintenant tenter de regrouper des mammifères par famille génétique à l'aide d'une partie de leurs séquences ADN. Il a été introduit dans [Li+04] et possède trente-quatre fichiers textes différents. Chaque fichier texte est une suite des quatre caractères $\{A, C, G, T\}$ qui représente un ADN et décrit le patrimoine génétique d'une espèce. En moyenne, chaque ADN a une taille de 16000 octets. Le nom du fichier ADN correspond à celui du mammifère dont il est extrait.

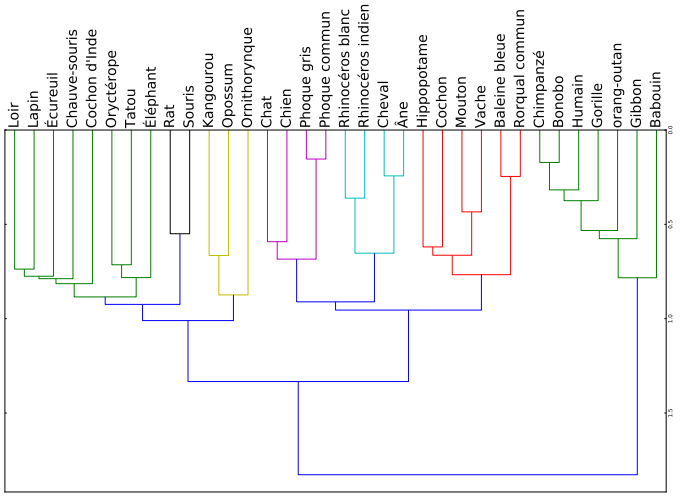
Données : ADN

Caractéristique : description génétique de différentes espèces de mammifères.
Longueur des chaînes : 16 kilo-octets à 17 kilo-octets.
Taille de l'alphabet, α : 4.
Nom de la chaîne : espèce du mammifère représentée par l'ADN.
Objectif : regrouper les mammifères par familles génétiques.

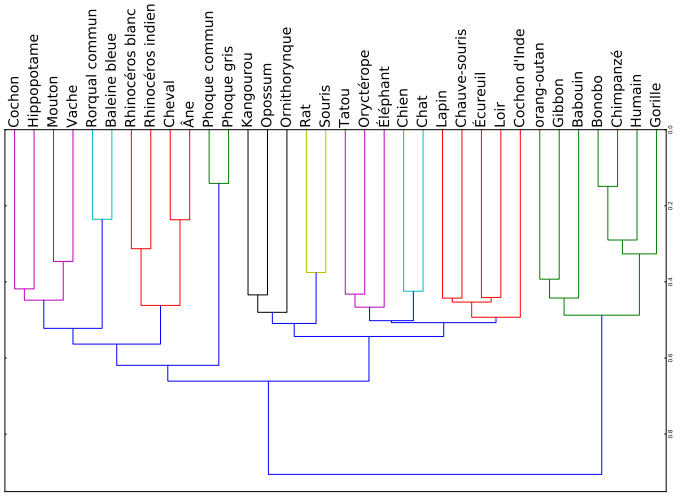
Dans la figure III.6, nous pouvons observer les classifications ascendantes hiérarchiques de l'ADN de mammifères obtenues grâce à la divergence de Ziv-Merhav, à la *NCD* basée sur le compresseur `LZMA` et à la *NSD* basée sur la fonction admissible exponentielle. La contrainte pour placer le seuil t est que tous les singes doivent être dans une seule et même classe.



(a) Divergence de Ziv-Merhav, $t = 1,2$.



(b) $NCD_{\texttt{LZMA}}$, $t = 0,1$.



(c) $NSD_{\mathbb{E}}$, $t = 0,5$.

FIGURE III.6 – Classification d'ADN de mammifères.

La divergence de Ziv-Merhav peine à établir des regroupements cohérents. La classe jaune regroupe beaucoup d'espèces qui n'ont rien à voir génétiquement les unes avec les autres. Par exemple, le chien qui est un carnivore est dans la même classe que le mouton qui est un herbivore. En particulier les espèces endémiques australiennes, les marsupiaux (opossum, kangourou et ornithorynque), ne forment pas une classe à part alors qu'elles sont génétiquement très différentes des autres espèces de mammifères.

La *NSD* fait apparaître dix classes. Parmi ces dix classes, nous retrouvons celle des grands singes en vert ou la classe des espèces endémiques de l'Australie en noir, mais encore une classe qui regroupe l'âne, le cheval et les deux rhinocéros. Ces espèces sont en effet assez similaires, car elles appartiennent toutes les quatre à l'ordre Perissodactyla. Il est aussi possible de noter la classe des baleines ou des grands herbivores. La *NCD* arrive à faire les mêmes regroupements mais arrive aussi à regrouper les phoques, le chien et le chat ensemble contrairement à la *NSD*. Ceci est plus correct, car ces quatre espèces appartiennent toutes les quatre à l'ordre des carnivores.

Pour comprendre le comportement des différentes métriques, nous pouvons regarder la figure III.7. Tout comme pour les chaînes de Markov (figure III.5), nous comparons les longueurs des décompositions pour le conditionnement de la *NCD* $\_|^+$, de la *NSD* $|^+$ et de la divergence de Ziv-Merhav. Nous comparons l'ADN d'humain et celui du chimpanzé qui sont deux espèces très proches génétiquement. Puis l'ADN d'humain et celui du chien qui sont très éloignées génétiquement.

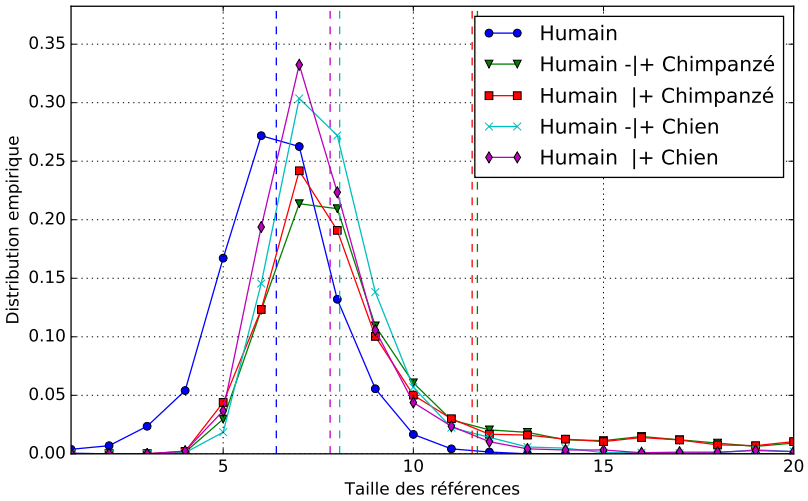


FIGURE III.7 – Histogramme des longueurs des opérations `copier` pour différentes décompositions d'ADN de mammifères. La courbe bleue est la décomposition de l'ADN d'humain. Les courbes verte et rouge sont les décompositions de l'ADN d'humain connaissant l'ADN de chimpanzé, avec les conditionnements $\_|^+$ et $|^+$. Les courbes bleue clair et violette correspondent aux décompositions de l'ADN d'humain connaissant l'ADN de chien, avec les conditionnements $\_|^+$ et $|^+$. Les barres verticales en pointillée représentent la longueur moyenne de chaque décomposition.

Nous avions observé, avec les chaînes de Markov, que le conditionnement $|^+$ faisait une nette différence entre le cas où deux chaînes étaient proches ou non, contrairement au conditionnement $\_|^+$. Dans le cas de l'ADN, les deux conditionnements produisent pratiquement

les mêmes décompositions. Ainsi quand nous décomposons l'ADN d'humain connaissant l'ADN de chien, la longueur moyenne des opérations `copier` est d'environ 8 pour les deux conditionnements. Par contre quand nous comparons deux ADN proches comme celui de l'homme et du chimpanzé alors la longueur moyenne est d'environ 12. Cet écart important et identique pour les deux types de conditionnement explique le fait que la *NSD* et la *NCD* donnent des résultats de classification corrects et très similaires.

Les difficultés rencontrées par la divergence de Ziv-Merhav peuvent être expliquées par la très forte redondance de l'ADN. En effet, la décomposition de l'ADN d'humain connaissant uniquement celui de chien a une longueur moyenne supérieure à celle de la décomposition de l'ADN d'humain connaissant uniquement son passé, et ceux alors que le chien et l'humain sont des espèces très éloignées génétiquement. Or la divergence de Ziv-Merhav compare si une chaîne est mieux expliqué par une autre ou par son propre passé. Dans le cas de l'ADN, on aura donc toujours un apport d'information, dû à sa grande redondance, ce qui fausse la mesure de la divergence de Ziv-Merhav.

III.3.3 Classification de langues indo-européennes

À travers un texte traduit en plusieurs langues indo-européennes, nous allons chercher à classifier des langues selon leur système d'écriture. Ce jeu de données a été introduit dans [Li+04]. La bibliothèque de l'ONU, [Onu], nous a permis de récupérer la Déclaration Universelle des Droits de l'Homme écrite dans cinquante langues indo-européennes différentes. Les langues indo-européennes utilisent un alphabet romain de 26 lettres plus des accents et des signes de ponctuation soit en moyenne un alphabet de taille 64. Le nom de la version représentée dans la classification de la Déclaration Universelle des Droits de l'Homme est la langue dans laquelle elle est rédigée.

Données : Déclaration Universelle des Droits de l'Homme

Caractéristique : Déclaration Universelle des Droits de l'Homme écrite dans différentes langues indo-européennes.

Longueur des chaînes : 10 kilo-octets à 13 kilo-octets.

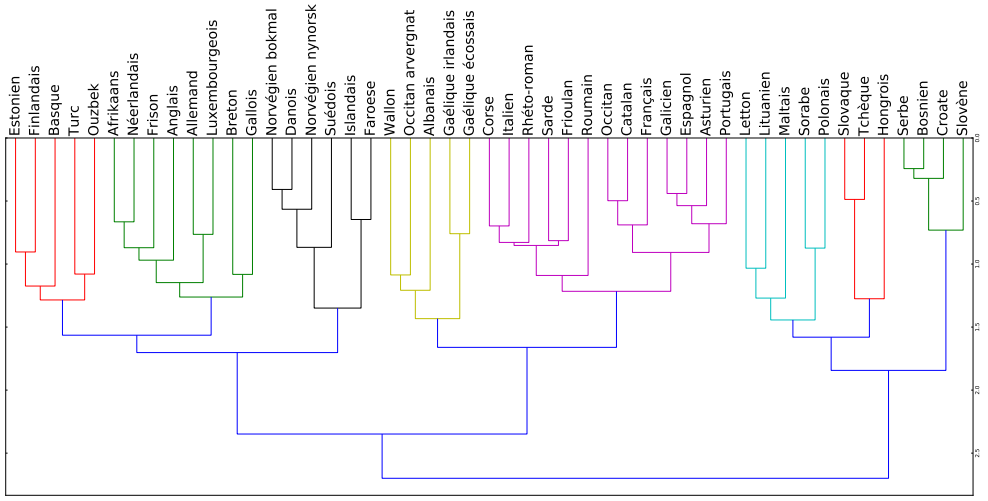
Taille de l'alphabet, α : environ 64.

Nom de la chaîne : langue dans laquelle la version a été rédigée.

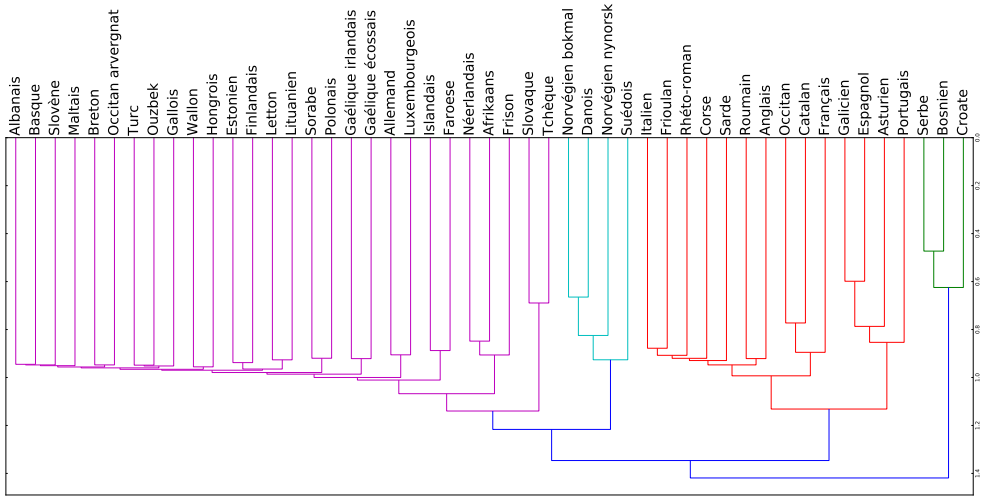
Objectif : regrouper les langues ayant la même origine dans leur système d'écriture.

La figure III.8 nous permet d'observer les classifications ascendantes hiérarchiques des langues indo-européennes écrites obtenues grâce à la divergence de Ziv-Merhav, à la *NCD* basée sur le compresseur `LZMA` et à la *NSD* basée sur la fonction admissible exponentielle. La contrainte pour placer le seuil t est d'avoir toujours les langues romanes dans une même classe.

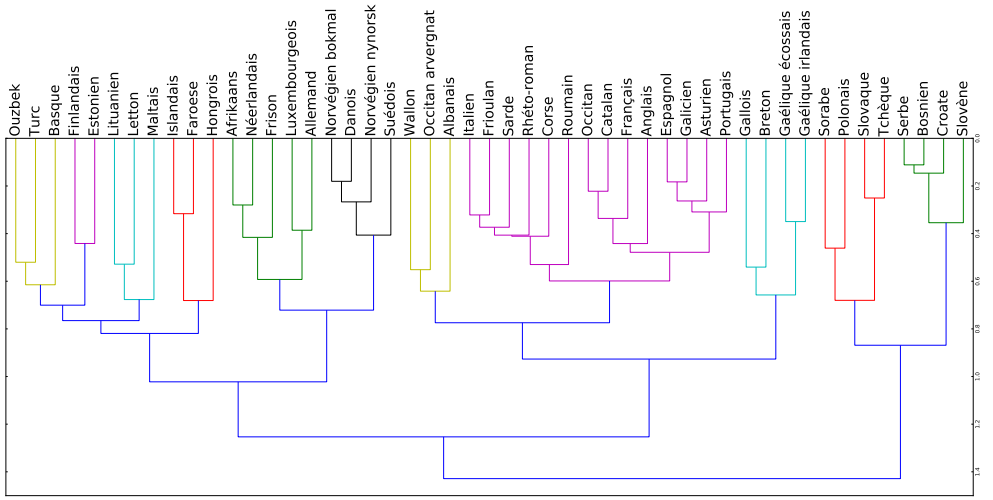
La *NSD* classifie les langues indo-européennes en 11 classes que nous allons décrire de la classe la plus en haut dans le graphe à la classe la plus basse dans le graphe.



(a) Divergence de Ziv-Merhav, $t = 1,5$.



(b) $NCD_{\texttt{LZMA}}$, $t = 1,15$.



(c) $NSD_{\mathbb{E}}$, $t = 0,7$.

FIGURE III.8 – Classification de langues.

- La première classe regroupe l'ouzbek et le turc qui sont toutes les deux des langues altaïques. Le basque est regroupé avec ces deux langues. Ce regroupement n'est pas du tout absurde, car c'est une des hypothèses émises sur l'origine du Basque [Mor92].
- La deuxième classe regroupe les langues finno-ougriennes. Cette classe est très proche de la classe précédente et elle a également été proposée comme origine du Basque [Mor92].
- La troisième classe est composée des langues baltiques.
- De la quatrième classe à la sixième, nous avons les langues germaniques. La classe quatre et la classe six sont les langues germaniques du nord. Et la classe cinq est le regroupement des langues germaniques de l'Ouest. L'afrikaans qui est une langue d'Afrique du Sud apparait aussi dans cette classe, ceci est dû à la colonisation de l'Afrique du Sud par les Néerlandais.
- Les deux classes suivantes sont les langues romanes. L'anglais est classé dans cette catégorie, car son influence est double : à la fois proche des langues germaniques par les Saxons, mais aussi du français. En effet, la cour anglaise a longtemps parlé français après la conquête de Guillaume le Conquérant. On peut aussi voir apparaitre des sous-classes dans cette grande famille comme celle des langues de la péninsule ibérique.
- La neuvième classe regroupe les langues celtiques.
- Les deux dernières classes sont les langues slaves.

La *NCD* ne permet pas du tout de retrouver toutes les subtilités de la classification réalisée par la *NSD*. La divergence de Ziv-Merhav obtient des résultats meilleurs que ceux de la *NCD* mais ne retrouve pas les langues celtiques par exemple.

La différence entre ces explications peut être interprétée grâce à la figure III.9. Nous représentons les longueurs des opérations `copier` pour les différents conditionnements des métriques. Puis nous comparons les décompositions quand deux langues sont proches comme le français et l'espagnol et quand deux langues sont différentes comme le polonais et le français.

L'explication des résultats est sensiblement la même que celle pour les chaînes de Markov. En effet, par exemple pour le français lorsqu'une voyelle est écrite, il est très peu probable qu'une deuxième voyelle la suive. Hormis pour les lettres 'l','m','n','p','s','r' et 't', il est très peu probable qu'une même lettre soit répétée. Ainsi nous pouvons observer le même phénomène dans la figure III.9 que dans la figure III.5. Grâce au conditionnement $|^+$, la différence des longueurs moyennes entre le français connaissant l'espagnol (langues très proches) et le français connaissant le polonais (langues très éloignées) est bien plus importante qu'avec le conditionnement $\_|^+$. La *NCD* a une discrimination bien moins bonne que la *NSD*, ce qui explique pourquoi la *NSD* obtient des résultats de classifications plus fins et pertinents.

III.3.4 Classification d'ouvrages littéraires français (1532-1890)

Tentons maintenant de classer des textes numérisés de la littérature française. Nous nous intéressons à différents romans français libres de droits. Tout comme les langues indo-européennes, ces romans sont écrits sur un alphabet de taille moyenne 64. La longueur des

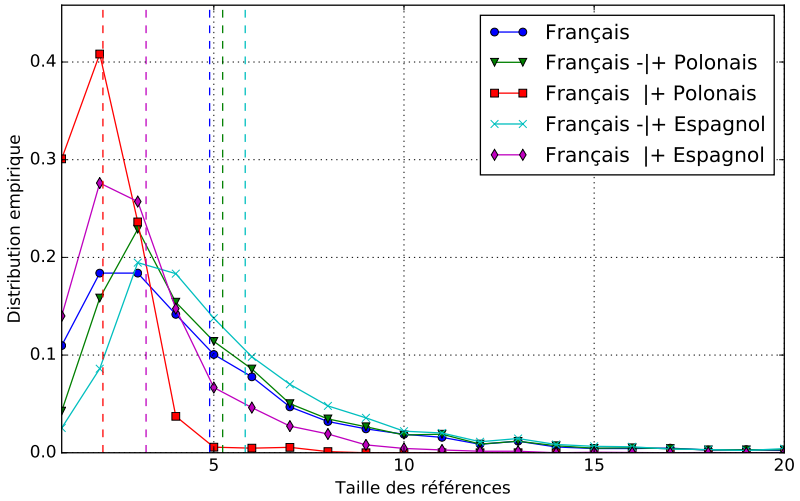


FIGURE III.9 – Histogramme des longueurs des opérations `copier` pour différentes décompositions de texte en différentes langues. La courbe bleue est la décomposition de la version française. Les courbes verte et rouge sont les décompositions de la version française connaissant la version polonaise, avec les conditionnements $\_|^+$ et $|^+$. Les courbes bleue clair et violette correspondent aux décompositions de la version française connaissant la version espagnole, avec les conditionnements $\_|^+$ et $|^+$. Les barres verticales en pointillée représentent la longueur moyenne de chaque décomposition.

textes varie énormément allant de 12 kilo-octets à 2 méga-octets. Chaque livre est présenté en utilisant le nom de son auteur et sa date de parution. Les titres des romans, sont répertoriés dans la table D.1 de l'annexe D.

Données : littérature française

Caractéristique : textes numérisés de la littérature française de différents auteurs et différentes époques.

Longueur des chaînes : de 12 kilo-octets à 2 méga-octets.

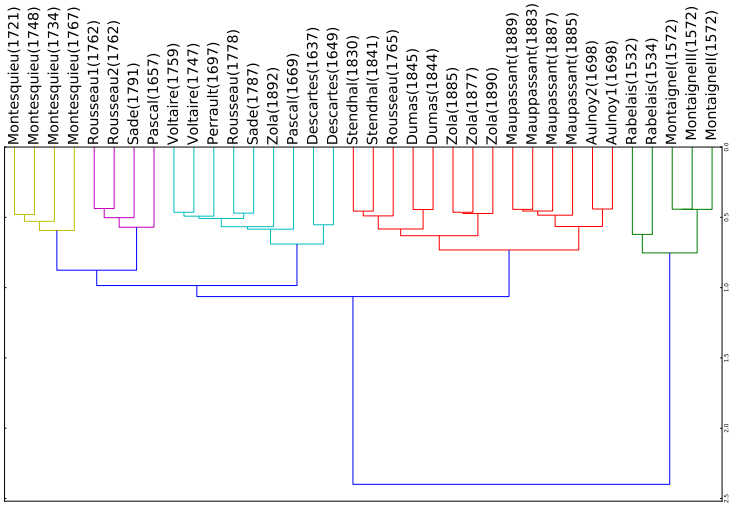
Taille de l'alphabet, α : environ 64.

Nom de la chaîne : auteur (date de parution).

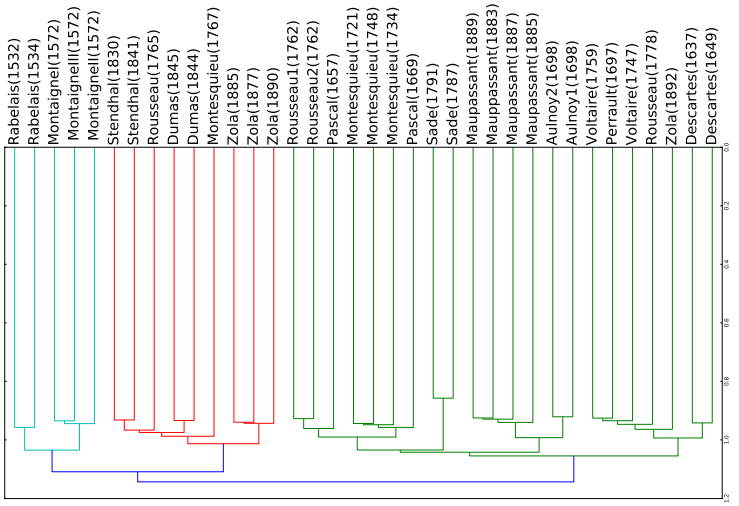
Objectif : regrouper des courants littéraires ou des œuvres d'un même auteur.

La figure III.10 nous permet d'observer les classifications ascendantes hiérarchiques de romans français obtenues grâce à la divergence de Ziv-Merhav, à la *NCD* basée sur le compresseur `LZMA` et à la *NSD* basée sur la fonction admissible exponentielle.

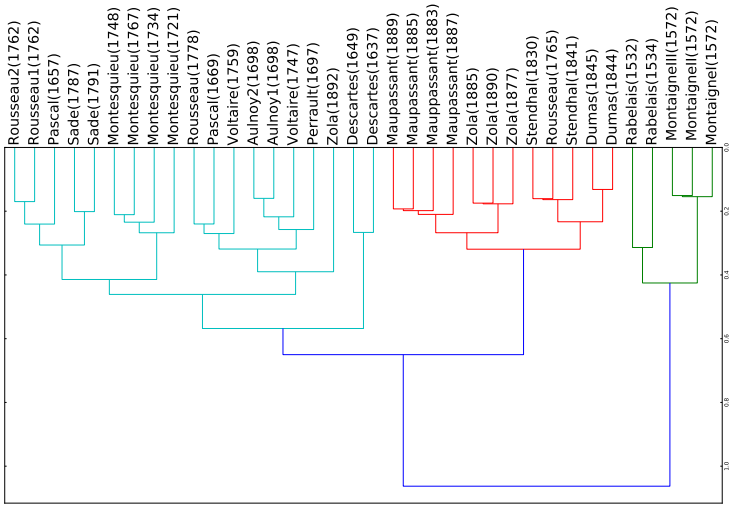
Les trois méthodes regroupent presque sans erreur les œuvres d'un même auteur. Une explication à cette classification réside sans doute dans le style littéraire et le vocabulaire propre à chaque auteur. Cependant toutes les méthodes classifient mal *Germinal* de Zola. Une autre erreur est faite sur *Les Confessions* de Rousseau qui se retrouve près de Stendhal. Mais cela n'est pas absurde, car Rousseau était une très grande source d'inspiration pour Stendhal.



(a) Divergence de Ziv-Merhav, $t = 0,8$.



(b) $NCD_{\texttt{LZMA}}$, $t = 1,1$.



(c) $NSD_{\mathbb{E}}$, $t = 0,6$.

FIGURE III.10 – Classification de livres de la littérature française.

Un autre critère de classification identifié par les différentes métriques semble être le siècle de parution. Ce critère fonctionne très bien pour la *NSD* et la divergence de Ziv-Merhav et un peu moins bien pour la *NCD*, qui classe Maupassant avec les auteurs du siècle précédent. Cela peut s'expliquer par le fait que chaque siècle avait ses sujets de prédilection ainsi que ses règles d'écriture. Les ouvrages d'une même époque ont donc un vocabulaire proche.

La *NSD* identifie une sous-classe qui regroupe les contes. Elle regroupe Aulnoy et Perrault qui ont écrit des contes pour enfants et les contes philosophiques de Voltaire. La *NSD* permet donc de retrouver, en plus des siècles et des auteurs, un sous-genre littéraire contrairement à la *NCD* et à la divergence de Ziv-Merhav.

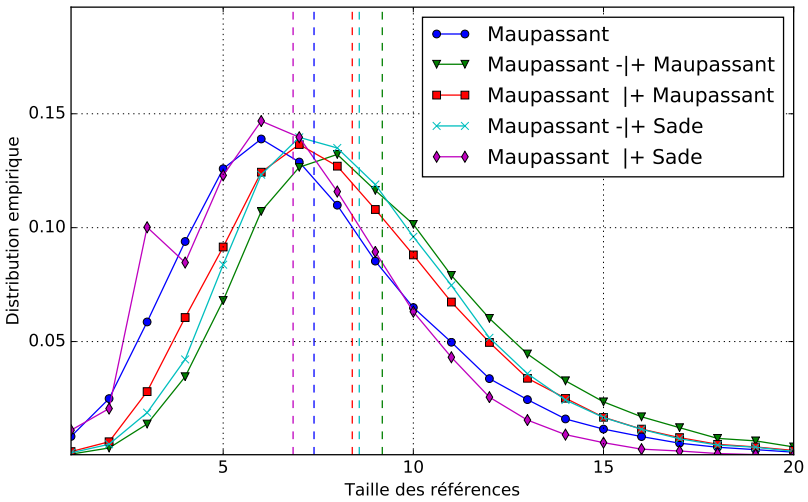


FIGURE III.11 – Histogramme des longueurs des opérations `copier` pour différentes décompositions de différents textes numérisés de la littérature française. La courbe bleue est la décomposition d'*Une vie* de Maupassant. Les courbes verte et rouge sont les décompositions d'*Une vie* de Maupassant connaissant *Justine* de Sade, avec les conditionnements $\_|^+$ et $|^+$. Les courbes bleue clair et violette correspondent aux décompositions d'*Une vie* de Maupassant connaissant *Bel ami* de Maupassant, avec les conditionnements $\_|^+$ et $|^+$. Les barres verticales en pointillée représentent la longueur moyenne de chaque décomposition.

L'explication des différences entre les métriques (figure III.11) est sensiblement la même que celle pour la Déclaration Universelle des Droits de l'Homme (figure III.9) car le type de données est sensiblement le même. La différence réside dans la taille des données. Pour la littérature française, les données sont bien plus longues, ainsi la *NCD* basée sur le compresseur `GZIP` ne peut pas fonctionner, car la taille de son buffer est trop petite. L'influence de la taille des données se voit par comparaison des figures III.11 et III.9. La longueur moyenne des opérations `copier` a beaucoup augmenté passant d'environ 3 à environ 8. Plus les opérations `copier` sont très grandes, plus la compression est efficace. L'augmentation de la longueur moyenne permet donc d'améliorer la *NCD* ce qui fait que la classification de la *NCD* est meilleure pour les livres que pour la Déclaration Universelle des Droits de l'Homme. Cependant elle reste toujours moins bonne que la *NSD* car l'amélioration faite par l'ajout d'un livre du même auteur est toujours moins grande avec la décomposition $\_|^+$ qu'avec la décomposition $|^+$.

En résumé

Résultat de la classification ascendante hiérarchique sur quatre jeux de données

Divergence de Ziv-Merhav :

- *Avantage* : elle produit de très bons résultats pour les chaînes de Markov, ce qui était attendu, car elle a été créée pour cela ([ZM93]).
- *Inconvénient* : il faut modifier la matrice pour qu'elle soit positive et pour avoir des 0 sur la diagonale. De plus si les données sont très redondantes, la méthode fonctionne mal.

NCD :

- *Inconvénient* : selon le compresseur utilisé, la taille des données peut être limitée. De plus, comme elle utilise le conditionnement $\_|^+$, la *NCD* fait moins la différence entre des données proches ou éloignées. Enfin la *NCD* n'étant pas une semi-distance, il faut adapter la matrice pour faire de la classification.

NSD :

- *Avantage* : les résultats obtenus sont meilleurs que ceux obtenus avec les autres méthodes grâce au conditionnement $|^+$ et à la prise en compte des longueurs des opérations `copier`. D'autre part, le pouvoir discriminant de la *NSD* est bien plus élevé que les autres méthodes ce qui permet de facilement ajuster le seuil permettant de séparer les classes. Enfin, par définition la matrice est déjà sous la forme requise par la bibliothèque *scipy*.

Conclusion

La complexité permet de définir des métriques entre chaînes indépendantes de toute hypothèse probabiliste ou du type de données étudiées. Ziv et Merhav sont les premiers à proposer une métrique basée sur la complexité de Lempel-Ziv : la divergence de Ziv-Merhav. Puis Benett et al. ont proposé une distance qui repose sur la complexité de Kolmogorov appelée *NID*. Cependant comme nous l'avons vu dans les chapitres précédents, la complexité de Kolmogorov n'est pas calculable en un temps fini. Li et Vitányi ont alors proposé d'utiliser l'approximation des compresseurs et ont ainsi créé la *NCD*. Cependant cette métrique ne garde que la propriété de positivité de la distance. Nous avons donc décidé d'utiliser `SALZA` comme estimateur de la complexité de Kolmogorov. Nous obtenons alors une semi-distance, c'est-à-dire que seule l'inégalité triangulaire n'est pas respectée, appelée *NSD*. Ces trois métriques, divergence de Ziv-Merhav, *NCD* et *NSD*, permettent de calculer des matrices de dissimilarité pour l'ensemble des chaînes de $\mathcal{X}$.

Nous avons combiné les trois métriques basées sur la complexité et la classification

ascendante hiérarchique pour classifier différents jeux de données. Dans un premier temps, par construction, la *NSD* est la seule métrique qui produise une matrice correcte pour les algorithmes de classification. En effet la *NCD* et la divergence de Ziv-Merhav ne respectent pas les propriétés de symétrie ou de positivité. Nous avons vu ensuite que la *NSD* permettait une meilleure classification sur tous les jeux de données que nous avons testé. Ceci est principalement dû au conditionnement $|^+$ que nous utilisons ainsi que la bonne prise en compte des longueurs grâce à la fonction admissible.

Chapitre IV

Inférence de causalité algorithmique

Sommaire

Face à un ensemble de chaînes, une question fondamentale est d'étudier sa structure de dépendance. Une des solutions est d'inférer un modèle graphique sous-jacent capable de rendre compte des relations causales entre les différentes chaînes. Nous posons donc dans un premier temps les notations et les concepts qui permettent de définir les modèles graphiques (section IV.1).

Une fois ces concepts définis nous présentons deux approches d'inférence de causalité. Classiquement, l'inférence de ces graphes est réalisée grâce aux outils probabilistes. Dans le cadre des chaînes de caractères, nous proposons des outils algorithmiques pour estimer les liens causaux.

La première approche repose sur la condition de Markov : un graphe est acceptable si pour tous les sommets, la chaîne de caractères est indépendante de ses non-descendants connaissant ses parents (section IV.2). Cette condition est classiquement définie via les probabilités et elle a été transposée par Janzing dans le domaine algorithmique [JS10]. Cependant, la complexité de Kolmogorov n'étant pas calculable en un temps fini, nous proposons d'utiliser `SALZA` pour pouvoir calculer la condition de Markov algorithmique.

La deuxième approche est la causalité de Granger : un processus X cause un autre processus Y si la prédiction de Y est améliorée en connaissant le passé de X [Gra69] (section IV.3). Michel et Amblard ont montré qu'il est possible d'estimer cette causalité grâce à l'information dirigée probabiliste [AM11]. Nous définissons la causalité de Granger algorithmique grâce à l'information dirigée basée sur `SALZA` (introduite dans la section II.3.2). Nous l'appliquons enfin à des chaînes de caractères simulées.

Ces travaux sur la causalité ont été motivés par une collaboration avec un chercheur travaillant au département de lettre de l'Université Grenoble Alpes. Ce dernier a mis à notre

disposition un corpus de textes regroupant différentes versions d'un même paragraphe. L'enjeu était de retrouver automatiquement l'influence entre les différentes versions. Les textes ne répondant pas à des modèles probabilistes, nous proposons de transposer les méthodes classiques d'inférence de causalité reposant sur les probabilités vers l'approche algorithmique pour répondre à cette problématique. Les applications, sur des données réelles et simulées que nous allons présenter, semblent montrer que l'approche algorithmique fonctionne correctement. Cependant, une validation sur d'autres jeux de données reste à faire, et permettrait de conforter les premiers résultats encourageants obtenus. Les développements présentés ci-après sont donc des résultats préliminaires à un travail sur le plus long terme portant sur l'inférence de causalité algorithmique.

IV.1 Notations et définitions pour la causalité

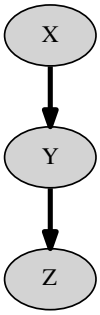
L'objectif de ce chapitre est d'inférer un modèle graphique qui représente les liens entre les différents processus à notre disposition. Ces processus peuvent être des variables aléatoires dans le cadre de la théorie probabiliste cependant ces processus peuvent aussi être des chaînes de caractères ce qui permet d'utiliser la théorie algorithmique. Un processus n'est donc pas forcément un processus aléatoire dans ce chapitre. Ce qui suit dans cette section est vrai qu'importe la nature du processus. Il faut dans un premier temps définir ce qu'est un graphe.

Définition IV.1.1 (Graphe)**.**
Un graphe $G = (V, E^u, E^d)$ est un couple formé de :

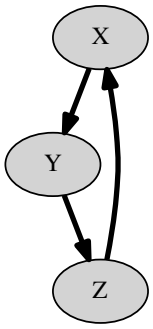
- *V : un ensemble de sommets, par exemple dans le cadre de la théorie algorithmique ce sont des chaînes de caractères.*
- *E^u : un ensemble d'arêtes non orientées, où chaque arête est un couple de sommets et représente le lien entre les deux sommets. Soient deux sommets X et Y, alors on note l'arête non orientée de X et Y par $(X, Y) \in V^2$.*
- *E^d : un ensemble d'arêtes orientées, où chaque arête est un couple de sommets et représente le lien orienté entre les deux sommets. Soient deux sommets X et Y, alors on note l'arête orientée de X vers Y par $(X, Y) \in V^2$.*

Il est intéressant de noter que si l'arête non-orientée $(X, Y) \in E^u$ existe alors automatiquement l'arête non-orientée inverse existe aussi $(Y, X) \in E^u$. Si le graphe représente une structure de causalité entre processus, alors une arête est appelé un lien de causalité.

Dans un graphe de causalité, nous cherchons quel processus a influencé un autre. Par exemple dans le domaine des probabilités, l'influence entre processus est liée à la densité de probabilité, mais l'influence peut aussi résulter d'une transformation linéaire ou encore d'un processus autoregressif. Dans les sections IV.2 et IV.3, nous proposons deux façons de quantifier cette influence grâce à la théorie algorithmique de l'information. Pour comprendre comment nous représentons un graphe, prenons un exemple : un processus X a influencé un processus Y qui a lui-même influencé un processus Z. Il existe donc un lien de causalité de X vers Y et un de Y vers Z. Le modèle graphique qui formalise la structure de causalité de cet ensemble de processus est donc $G = (V = \{X, Y, Z\}, E^u = \{\}, E^d = \{(X, Y), (Y, Z)\})$,



(a) $G = (V = \{X, Y, Z\}, E^d = \{(X, Y), (Y, Z)\})$.



(b) $G = (V = \{X, Y, Z\}, E^d = \{(X, Y), (Y, Z), (Z, X)\})$.

FIGURE IV.1 – Représentation des liens de causalité entre X, Y et Z.

ce graphe est représenté dans la figure IV.1a.

Nous introduisons maintenant quelques éléments de vocabulaire propre aux graphes.

Définition IV.1.2 (Parents et descendants)**.**
Soient un graphe $G = (V, E^d)$ et le sommet $X \in V$, on dit que :

- *$Y \in V$ est un parent de X s'il existe l'arête $(Y, X) \in E^d$,*
- *$Y \in V$ est descendant de X s'il existe une arête directe (X, Y) ou indirect entre X et Y, c'est-à-dire qu'il existe n tel que $\forall i = 1...n$, $X_i \in V$ les arêtes $\{(X, X_1), (X_1, X_2), ..., (X_n, Y)\} \in (E^d)^{n+1}$ existent,*
- *$Y \in V$ est non-descendant de X si Y n'est pas descendant de X et si Y n'est pas le parent de X.*

On note PA_X l'ensemble des parents de X et ND_X l'ensemble des non-descendants de X.

Reprenons l'exemple de la figure IV.1a. Dans ce cas, le parent de Z est Y et son non-descendant est X : $PA_Z = \{Y\}$ et $ND_Z = \{X\}$.

Nous travaillerons par la suite uniquement sur des graphes orientés acycliques ([Pea09]). Cela signifie qu'il ne peut pas y avoir de cycles dans le graphe, c'est-à-dire qu'un sommet ne peut pas être descendant de lui-même. Le graphe représenté dans la figure IV.1b ne sera donc pas considéré ici.

Reprenons l'exemple de la figure IV.1a pour comprendre un des enjeux de la causalité. Il est possible que X influence Z car X influence Y : Z n'est pas indépendant de X. Cependant, nous ne souhaitons pas de lien de causalité entre X et Z car X n'influence Z que par le biais de Y. En effet toute l'information apportée par X dans Z a forcément transité par Y. Si nous connaissons déjà Y, X n'influence pas Z : Z est indépendant de X en sachant Y. Il n'y a donc une arête que si les processus sont indépendants conditionnellement aux autres.

Définition IV.1.3 (Indépendance conditionnelle)**.**
Si X est indépendant de Z sachant Y alors on note :

$$X \perp\!\!\!\perp Z|Y.$$

La relation d'indépendance conditionnelle est symétrique, c'est-à-dire que si X est indépendant de Z sachant Y, alors Z est indépendant de X connaissant Y.

En complément de la condition d'indépendance, il est possible d'émettre au préalable des règles d'ordre au sein des processus. Ces règles donnent des directions possibles sur les liens de causalité. Dans le cas où les processus sont des séries temporelles, le temps fournit une base naturelle pour la définition du sens de la causalité. Une des causalités les plus connues qui repose sur cette hypothèse temporelle est la causalité de Granger comme nous le verrons dans la section IV.3. Si nous ne faisons aucune hypothèse sur l'ordre des processus, nous n'aurons qu'un ensemble de graphes. C'est ce que nous allons voir dans la section IV.2.

IV.2 Causalité de Markov

Dans cette section, nous étudions la condition de Markov qui pose qu'un sommet doit être indépendant de ses non-descendants connaissant ses parents. Nous montrons comment écrire cette condition avec `SALZA`. Dans ce cas, nous n'utilisons que des arc dirigés et les graphes s'écrivent donc sous la forme $G = \{V, E^d, E^u = \emptyset\} = \{V, E^d\}$.

IV.2.1 Définition

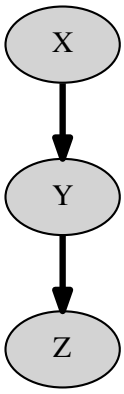
Différentes versions de la condition de Markov existent. Nous présentons ici la version locale. Les autres versions algorithmiques sont présentées dans [JS10]. Nous travaillons toujours sur des processus (c'est-à-dire qui peuvent être des variables aléatoires ou des chaînes de caractères) dans cette section.

Définition IV.2.1 (condition de Markov)**.**
Soit un graphe G qui formalise la structure de causalité sur les processus $X_1, X_2, ..., X_n$, alors G respecte la condition de Markov si :

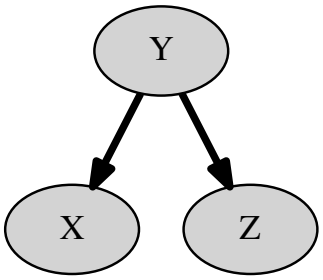
$$\forall i = 1...n \;\; X_i \perp\!\!\!\perp ND_{X_i}|PA_{X_i}. \tag{IV.1}$$

Un graphe est valide selon la condition de Markov uniquement si chaque sommet est indépendant de ses non-descendants connaissant ses parents. C'est-à-dire que l'information qu'il peut avoir en commun entre un sommet et ses non-descendants transite forcément par ses parents.

Cette condition ne permet d'obtenir qu'un seul et unique graphe dans très peu de cas ([JS10]). La plupart du temps, nous obtenons une classe de graphes équivalents. Prenons l'exemple du graphe dans la figure IV.2a. Voici les conditions de Markov auxquelles il répond :



(a) $G = (V = \{X, Y, Z\}, E^d = \{(X, Y), (Y, Z)\})$.



(b) $G = (V = \{X, Y, Z\}, E^d = \{(Y, X), (Y, Z)\})$.

FIGURE IV.2 – Deux graphes ayant les mêmes conditions de Markov.

- X n'a pas de parent et aucun non-descendant. La condition est donc $X \perp\!\!\!\perp \emptyset | \emptyset$. Or une chaîne est toujours indépendante de la chaîne vide. Cette condition est donc valable pour tous les graphes, il n'y a aucune condition spécifique à respecter pour X.
- Y a comme parent X et aucun non-descendant. Il n'y a donc aucune condition spécifique à respecter pour la même raison que précédemment.
- Z a comme parent Y et comme non-descendant X, la condition d'indépendance est $X \perp\!\!\!\perp Z | Y$.

La condition de Markov du graphe G de la figure IV.2a est donc que X est indépendant de Z connaissant Y : $X \perp\!\!\!\perp Z | Y$. Faisons maintenant la même opération avec la figure IV.2b. Les conditions de Markov du graphe sont les suivantes.

- X a comme parent Y et comme non-descendant Z, la condition d'indépendance est $X \perp\!\!\!\perp Z | Y$.
- Y n'a pas de parent et aucun non-descendant, il n'y a donc aucune condition spécifique à respecter.
- Z a comme parent Y et comme non-descendant X, la condition d'indépendance est $Z \perp\!\!\!\perp X | Y$.

Les deux graphes ont donc exactement les mêmes conditions d'indépendances (car $\perp\!\!\!\perp$ est symétrique). Ils sont donc équivalents au regard des conditions de Markov et on ne peut pas faire la différence entre les deux uniquement avec les conditions de Markov.

En résumé

Condition de Markov :

- *Principe* : un graphe G respecte la condition de Markov si tous ses sommets sont indépendants de leurs non-descendants sachant leurs parents.
- *Avantage* : cette condition est plutôt intuitive, les non-descendants n'apportent rien en plus de ce qu'ont déjà apporté les parents.
- *Inconvenient* : la plupart du temps nous n'obtenons qu'une classe de graphes équivalents. Pour cette raison, d'autres règles d'inférence doivent être definies pour enlever toute ambiguïté.

IV.2.2 Théorie de l'information et condition de Markov

Maintenant que la condition de Markov est défnie dans le cas générique, nous montrons comment la calculer dans un cas spécifique. Steudel et al. ([SJS10]) ont montré que pour toute mesure d'information R, la condition d'indépendance conditionnelle de X et Y sachant Z est :

$$X \perp\!\!\!\perp Y|Z \Leftrightarrow I_R(X,Y|Z) = R(X|Z) + R(Y|Z) - R(X,Y|Z) = 0.$$

Cette relation suggère que nous pouvons utiliser `SALZA` comme mesure d'information dans la condition de Markov. Comme nous utilisons `SALZA` et la théorie algorithmique de l'information, les processus sont dans ce cas spécifique des chaînes de caractèress.

Définition IV.2.2 (condition de Markov basée sur `SALZA`)**.**
Soit un graphe orienté acyclique G qui formalise la structure de causalité sur les chaînes de caracètres $X_1, X_2, ..., X_n$, alors G respecte la condition de Markov si :

$$\forall i = 1..n \;\; I_{S_f}(ND_{X_i}, X_i|PA_{X_i}) = 0 \Leftrightarrow S_f(X_{i\text{-}}|^+PA_{X_i}) = S_f(X_{i\text{-}}|^+PA_{X_i}, ND_{X_i}).$$

Un sommet respecte les conditions de Markov si la connaissance de ses non-descendants ne permet pas de faire de plus grandes opérations `copier` que lorsque nous connaissons uniquement les parents.

En résumé

La condition de Markov s'écrit avec `SALZA` de la façon suivante :

$$\forall i = 1..n \;\; I_{S_f}(ND_{X_i}, X_i|PA_{X_i}) = 0 \Leftrightarrow S_f(X_{i\text{-}}|^+PA_{X_i}) = S_f(X_{i\text{-}}|^+PA_{X_i}, ND_{X_i}). \quad \text{(IV.2)}$$

IV.2.3 Résultats

Nous allons maintenant essayer de retrouver les graphes de causalité grâce à la condition de Markov basée sur `SALZA` avec deux jeux de données. Le premier est la traduction automatique d'un même texte dans différentes langues. Le deuxième est le cheminement d'écriture d'un auteur.

IV.2.3.1 Traduction

Tout d'abord nous allons reprendre l'exemple présenté dans [SJS10]. Pour cela nous utilisons un texte dans une langue d'origine qui est la version 1 (notée $V1$). Nous le traduisons automatiquement en une langue de transit grâce à google traduction, puis nous retraduisons dans la langue d'origine. Cette version est alors la version 2 (notée $V2$). Cette étape est réitérée une dernière fois et on note $V3$ cette version. Nous avons donc au final trois versions d'un même texte dans une même langue.

Dans un premier temps, nous analysons tous les graphes orientés acycliques avec trois éléments. Les six classes de graphes équivalents et leurs conditions de Markov sont représentées dans le tableau IV.1.

classe	graphe	condition de Markov
1	$E^d = \{(V1, V2), (V2, V3)\}$ $E^d = \{(V2, V1), (V2, V3)\}$ $E^d = \{(V2, V1), (V3, V2)\}$	$V1 \perp\!\!\!\perp V3 \mid V2$
2	$E^d = \{(V1, V2), (V3, V2)\}$	$V1 \perp\!\!\!\perp V3$
3	$E^d = \{(V2, V1), (V1, V3)\}$ $E^d = \{(V1, V2), (V1, V3)\}$ $E^d = \{(V3, V1), (V1, V2)\}$	$V2 \perp\!\!\!\perp V3 \mid V1$
4	$E^d = \{(V2, V1), (V3, V1)\}$	$V2 \perp\!\!\!\perp V3$
5	$E^d = \{(V1, V3), (V3, V2)\}$ $E^d = \{(V3, V1), (V3, V2)\}$ $E^d = \{(V2, V3), (V3, V1)\}$	$V1 \perp\!\!\!\perp V2 \mid V3$
6	$E^d = \{(V3, V1), (V3, V2)\}$	$V1 \perp\!\!\!\perp V2$

TABLE IV.1 – Les différentes classes de graphe d'équivalence pour les conditions de Markov pour trois sommets

Nous appliquons ensuite les différentes traductions présentées dans le tableau IV.2 en utilisant pour chacune des langues d'origine, une langue de transit. Cela donne quatre graphes à inférer : le premier chapitre d'*"Une vie"* avec comme langue d'origine le français et comme langue de transit l'allemand ou l'espagnol et la Déclaration Universelle des Droits de l'Homme avec comme langue d'origine le français et comme langue de transit l'allemand ou l'espagnol.

Pour bien comprendre l'influence de la traduction, prenons le dernier paragraphe du chapitre 1 d'*"Une vie"* de Maupassant et observons les différences obtenues.

texte étudié	langue originale	langue de transit
Premier chapitre d'*"Une vie"* de Maupassant	français	allemand, espagnol
Déclaration universelle des droits de l'homme	français	allemand, espagnol

TABLE IV.2 – Les différents textes, leurs langues d'origine et leurs langues de transit

Texte original : Comme le gros poisson fatiguait Jeanne, elle lui passa dans les ouïes la canne de son père, dont chacun d'eux prit un bout ; et ils allaient gaiement en remontant la côte, bavardant comme deux enfants, le front au vent et les yeux brillants, tandis que la barbue, qui lassait peu à peu leurs bras, balayait l'herbe de sa queue grasse.

Après une traduction : Quand le gros poisson de Jeanne se fatiguait, elle laissait passer le bâton de son père dans leurs oreilles, dont chacune se terminait ; et ils allèrent gaiement sur la côte, babillant comme deux enfants, le front au vent et les yeux brillants, tandis que le poisson-chat, qui fatigua peu à peu leurs bras, balaya l'herbe de sa grosse queue.

Après une deuxième traduction : Quand le gros poisson de Jeanne s'est fatigué, elle a coincé son père à travers ses oreilles, dont chacune s'est terminée ; et ils sont allés joyeusement à la banque, babillant comme deux enfants, blessant leur front et leurs yeux brillants, tandis que le poisson-chat, qui s'étendait progressivement ses bras, a balayé l'herbe avec sa grosse queue.

Nous voyons bien que chaque étape de traduction apporte son lot d'erreurs [1]. Lorsqu'une erreur est introduite au cours d'une traduction, elle se propage tout au long des traductions suivantes comme nous pouvons le voir avec les mots en couleur. Il est très improbable que cette erreur s'inverse pour revenir à la version originale. La version 3 est donc bien modifiée depuis la version 2 et non pas depuis la version 1. Le graphe souhaité est donc celui de la figure IV.2a et appartient à la classe 1 du tableaux IV.1.

Pour trouver les classes d'équivalences, nous calculons les six conditions d'indépendances pour les quatre graphes. Les valeurs que nous avons trouvées pour la condition de Markov de chacune des classes avec `SALZA` sont représentées dans le tableau IV.3. Nous n'avons jamais une condition strictement égale à 0, donc nous choisissons la condition la plus faible. Tout comme [SJS10] avec la complexité de Lempel-Ziv, nous retrouvons la bonne classe de graphes dans tous les cas où nous avons l'avons testée. Ce jeu de données nous permet donc de valider l'approche algorithmique basée sur `SALZA` des conditions de Markov.

IV.2.3.2 Cheminement d'écriture

Nous proposons une nouvelle application basée sur la littérature aux conditions de Markov algorithmiques basée sur `SALZA`. Jean-Philippe Toussaint est un auteur belge et chacun

1. J'espère que Maupassant me pardonnera la destruction de son texte.

Type de données	Langue originale	Langue de transit	Condition de Markov pour chaque classe					
			1	2	3	4	5	6
Maupassant	français	allemand	766	6928	1922	8016	12885	19585
		espagnol	304	9096	1877	10627	14691	23744
Droit de l'homme		allemand	492	5053	673	5423	3278	8051
		allemand	116	6567	381	6824	3198	9648

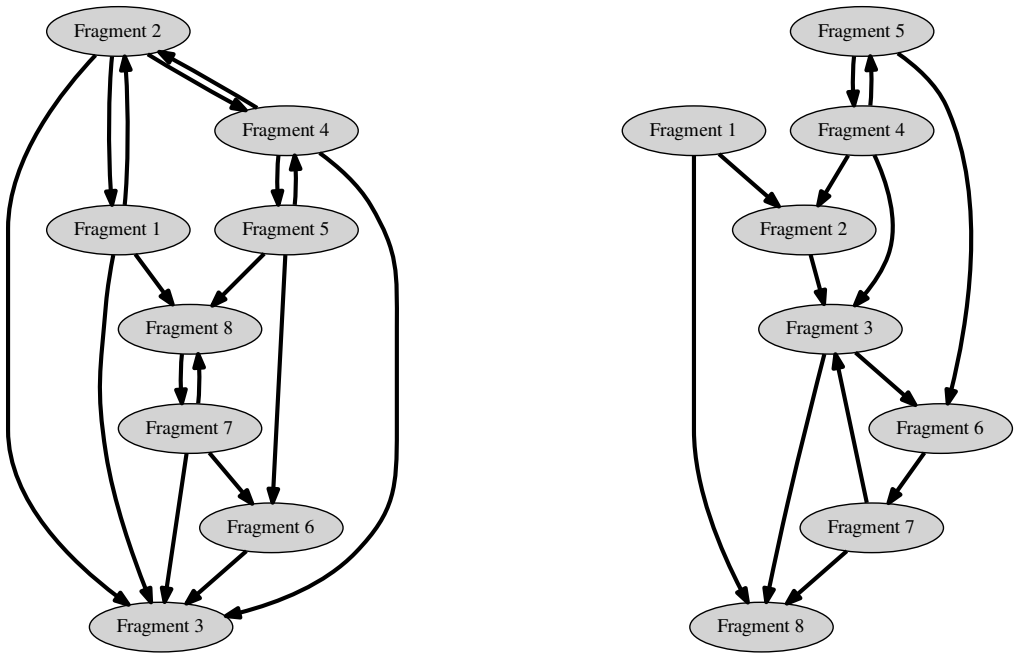
TABLE IV.3 – Condition de Markov calculée avec `SALZA` pour les différentes classes d'équivalence

des paragraphes publiés dans ses livres est le résultat d'itérations successives. D'abord, Jean-Philippe Toussaint écrit le paragraphe à la machine à écrire ou sur ordinateur et l'imprime. C'est l'étape de production de la première version. Ensuite, il annote manuellement ses corrections, créant ainsi une deuxième version. Puis il réécrit la version annotée en introduisant de nouveaux changements ou en supprimant des changements précédents, générant une troisième version. Ce processus est répété jusqu'à ce qu'il obtienne la version à publier. Chaque version est alors appelée un fragment. Nous nous proposons d'inférer les liens de causalité entre ces différents fragments d'un praragraphe issu de ce cheminement d'écriture itératif grâce aux conditions de Markov algorithmiques basée sur `SALZA`.

Notre jeu de données possède huit versions différentes d'un même paragraphe (pour plus de détails, voir l'annexe D). Or, il est assez difficile de trouver tous les graphes orientés acycliques avec huit sommets. Ainsi pour estimer un seul et unique graphe, Steudel et al. proposent d'utiliser *PCalg* en utilisant comme condition d'indépendance la condition de Markov. L'algorithme *PCalg* a été introduit par Pearl [Pea09]. Nous utilisons l'implémentation en python présentée dans [KB07].

Nous avons utilisé une bibliothèque déjà implémenté de *PCalg*. Les différents calculs de `SALZA` sont donc réalisés indépendemment les uns des autres. Nous ne pouvons donc pas définir de façon automatique $l^\star$ comme étant la longueur moyenne de toutes les décompositions. Nous avons donc calculé la longueur moyenne séparément, ici 14, et utilisé cette valeur comme longueur seuil. Le graphe obtenu grâce à *PCalg* et à la condition de Markov avec `SALZA` est présenté dans la figure IV.3b. Nous pouvons observer quelques liens causaux absurdes comme le fragment 4 qui aurait influencé l'écriture du fragment 2 et 3 alors qu'il a été écrit après ces derniers. Cependant, nous retrouvons principalement des liens causaux logiques. Le fragment 1 a influencé l'écriture du fragment 2 qui a lui-même influencé l'écriture du fragment 3... et ainsi de suite jusqu'au dernier fragment. Cette suite représente bien le cheminement d'écriture et est seulement interrompue entre le fragment 3 et 4 où le lien est inversé. Outre l'ordre logique d'écriture, nous retrouvons aussi le cheminement de la réflexion d'une version à l'autre. Par exemple, nous trouvons un lien direct entre le fragment 1 et le fragment final. Ceci est logique, Jean-Philippe Toussaint réintègre des éléments de la version initiale dans la version finale. Ce même phénomène est aussi observable pour le fragment 6 qui s'appuie sur la version antérieure, mais aussi reprend des éléments du fragment 3. La condition de Markov algorithmique permet donc de retrouver le cheminement d'écriture d'un auteur au fil des versions rédigées.

La figure IV.3a représente le graphe obtenu selon la méthode de Steudel et al [SJS10], c'est-à-dire grâce à la condition de Markov basée sur la complexité de Lempel-Ziv. Les



(a) En utilisant la complexité de Lempel-Ziv.

(b) En utilisant `SALZA` avec la fonction admissible exponentielle $\mathbb{E}$ et $l^\star$=14.

FIGURE IV.3 – Liens causaux entre les différentes version d'un paragraphe de Jean-Philippe Toussaint par *PCalg* ([KB07]) avec la condition de Markov basée sur `SALZA`.

arêtes qui représentent des liens absurdes (un fragment postérieur aurait influencé un fragment antérieur) sont beaucoup plus fréquents avec la complexité de Lempel-Ziv qu'avec `SALZA` (8 liens au lieu de 3). De plus, le cheminement d'écriture est interrompu entre les fragments 6 et 7 contrairement à `SALZA`. Ces différences peuvent être expliquées par le fait que les opérations `copier` sont très longues en moyenne, car souvent des phrases entières sont reprises entre deux itérations successives. Or, nous avons vu dans la section II.4.1 que la complexité de Lempel-Ziv avait un bon pouvoir discriminant pour des décompositions qui ont une faible longueur moyenne des opérations `copier`. Grâce à `SALZA`, nous sommes discriminants autour de la longueur moyenne, ce qui permet d'éviter les non-sens observés avec la complexité de Lempel-Ziv.

En résumé

SALZA et la causalité basée sur la condition de Markov :

- *Validation* : nous retrouvons la bonne classe de graphe avec trois sommets sur des données réelles.
- *Perspective* : la condition de Markov permet de retrouver le cheminement d'écriture entre différentes versions d'un même texte, d'autres applications peuvent être envisagées en littérature pour retrouver des influences entre auteurs par exemple.

IV.3 Causalité de Granger

La causalité de Granger permet d'estimer un graphe de causalité en utilisant le temps pour ordonner les variables. Ainsi nous n'aurons plus une classe de graphes équivalents mais un seul et unique graphe. Nous présenterons comment la causalité de Granger a été définie [Gra69]. Puis nous verrons qu'il est possible de l'estimer grâce à l'entropie de Shannon et SALZA. Nous présenterons enfin quelques résultats sur des données simulées.

IV.3.1 Définition

En 1969, l'économiste Clive Granger définit comment mesurer les relations causales [Gra69]. Cette mesure de causalité a été développée pour des processus stochastiques que nous noterons comme suit.

- X : un processus stochastique à temps discret.
- X_t : la valeur à l'instant t de X.
- $\overline{X_t}$: l'ensemble des valeurs passées de X_t ($\{X_0, ..., X_{t-1}\}$).
- $\overline{\overline{X_t}}$: l'ensemble des valeurs passées et présente de X_t ($\{X_0, ..., X_t\} = \overline{X_t} \cup X_t$).

Chaque élément X_t peut être expliqué par une combinaison linéaire de ses $t-1$ derniers éléments plus une erreur de prédiction. Par ailleurs, nous pouvons aussi écrire chaque élément X_t comme une combinaison linéaire des $t-1$ derniers éléments de X et d'un autre processus stochastique Y plus une erreur de prédiction. Cette prédiction est notée comme suit.

- $P(X|Y)$: le prédicteur optimal non biaisé de X en utilisant le processus stochastique Y.
- $\epsilon_t(X|Y)$: la série d'erreurs prédictives à moyenne nulle.
- $\sigma^2(X|Y)$: la variance de $\epsilon_t(X|Y)$.

Soit U l'ensemble des processus stochastiques à notre disposition. On note $U \backslash Y$ cet ensemble privé du processus stochastique Y. La causalité de Granger s'exprime donc selon la définition suivante.

Définition IV.3.1 (Causalité de Granger)**.**
Y cause X si et seulement si :

$$\sigma^2(X|\overline{U}) < \sigma^2(X|\overline{U\backslash Y}).$$

Y cause X signifie que Y permet de mieux prédire X. En effet, la prédiction de X_t connaissant toutes les valeurs passées exceptées celles de Y est améliorée lorsque nous connaissons en plus celles de Y.

Définition IV.3.2 (Rétroaction)**.**
Il y a rétroaction entre X et Y, si et seulement si

$$\sigma^2(X|\overline{U}) < \sigma^2(X|\overline{U\backslash Y})$$

et

$$\sigma^2(Y|\overline{U}) < \sigma^2(Y|\overline{U\backslash X}).$$

La rétroaction entre Y et X signifie que Y permet de mieux prédire X et que X permet de mieux prédire Y.

Définition IV.3.3 (Causalité de Granger instantanée)**.**
Y cause instantanément X si et seulement si

$$\sigma^2(X|\overline{U},\overline{\overline{Y}}) < \sigma^2(X|\overline{U}).$$

Si Y cause instantanément X cela veut dire que l'instant présent de Y apporte de l'information à X. C'est-à-dire que Y_t permet de mieux prédire X_y.

Illustrons ces définitions avec l'exemple simple de la météorologie. S'il y a des nuages dans le ciel depuis plusieurs heures alors il y a de fortes chances qu'il pleuve. Regarder le ciel nous permet donc de mieux prédire l'arrivée de la pluie. Les nuages causent donc la pluie. Par contre le fait qu'il y ait eu de la pluie dans le passé ne permet pas de dire si le temps est nuageux ou pas. Mais qu'il pleuve à l'instant présent nous permet de prédire la présence de nuages dans le ciel. Il y a donc un lien de causalité instantanné entre la pluie et la présence de nuages.

Maintenant que ces notions sont établies, nous pouvons définir un graphe de causalité basé sur la causalité de Granger [Eic00].

Définition IV.3.4 (Graphe de causalité de Granger)**.**
Soit $\mathcal{X} = \{X_1, ..., X_n\}$ un ensemble de n processus stochastiques, alors le graphe de causalité de Granger $G = (\mathcal{X}, E^u, E^d)$ est défini $\forall i = 1...n, j = 1...n$:

$$(X_i, X_j) \notin E^d \Leftrightarrow X_j \perp\!\!\!\perp \overline{X_i}|\overline{\mathcal{X}}\backslash\overline{X_i}$$

et

$$(X_i, X_j) \notin E^u \Leftrightarrow X_j \perp\!\!\!\perp X_i|\overline{\overline{\mathcal{X}}}\backslash\{X_i, X_j\}.$$

Il existe un arête orienté de X vers Y si X cause Y au sens de la causalité de Granger.

Les arcs non-orientés représentent quant à elles la causalité instantanée de Granger entre deux processus.

En résumé

Causalité de Granger :

- *Principe* : si Y permet de mieux prédire X alors Y cause X.
- *Avantage* : la causalité de Granger permet d'obtenir un seul et unique graphe.
- *Inconvénient* : il faut travailler sur des processus dont le temps est une variable latente.

IV.3.2 Théorie de l'information et causalité de Granger

La définition de la causalité de Granger est générique et rien n'est spécifié sur les conditions d'indépendance utilisées. Dans cette section, nous proposons d'interpréter l'indépendance grâce à l'information dirigée. Dans un premier temps, nous verrons comment Amblard et Michel ont réussi à relier la causalité de Granger et l'information dirigée probabiliste [AM11]. Puis nous montrerons qu'il est possible de la même manière d'utiliser l'information dirigée algorithmique basée sur `SALZA`.

IV.3.2.1 Théorie probabiliste de l'information

Pour rappel (voir section II.3.2.2), on note l'entropie de Shannon conditionnelle causale comme suit

$$\overrightarrow{H}(Y|X) = -\sum_{k=0}^{n} P(y_i|x_{1:i}, y_{1:i-1}) log_2 P(y_i|x_{1:i}, y_{1:i-1}).$$

L'opérateur `retard` retarde une chaîne de caractères d'un symbole et se note `R`. La chaîne `R`X est la chaîne X retardée de 1, c'est-à-dire que le $i^{\text{ième}}$ symbole de `R`X est le $i+1^{\text{ième}}$ symbole de X (voir section II.3.2.2).

Amblard et Michel définissent dans [AM11] l'information dirigée de X vers Y conditionnelle par rapport à Z comme étant :

$$I(X \to Y||Z) = \overrightarrow{H}(Y|Z) - \overrightarrow{H}(Y|Z, X).$$

Ils ont montré qu'il est possible de trouver une équivalence entre les conditions de causalité de Granger et l'information dirigée. Il est alors possible de définir un graphe de causalité basé sur la causalité de Granger et l'information dirigée.

Définition IV.3.5 (Graphe de causalité de Granger basé sur l'information dirigée probabiliste)**.**
Soit $\mathcal{X} = \{X_1, ..., X_n\}$ *un ensemble de processus stochastiques, alors le graphe de causalité de Granger* $G = (\mathcal{X}, E^u, E^d)$ *basé sur l'information dirigée probabiliste est défini* $\forall i = 1...n, j = 1...n$ *comme :*

$$(X_i, X_j) \notin E^d \Leftrightarrow I(\mathtt{R}X_i \to X_j || \mathcal{X} \backslash \{X_i, X_j\}) = 0$$

et

$$(X_i, X_j) \notin E^u \Leftrightarrow I(X_i \to X_j || \mathtt{R}X_i, \mathcal{X} \backslash \{X_i, X_j\}) = 0$$

Si X_i cause X_j cela veut dire que le passé de X_i permet de mieux prédir l'instant présent de X_j. Il y a donc un arête orientée (X_i, X_j). Si X_i cause instantanément X_j cela veut dire que l'instant présent de X_i permet de mieux prédire l'instant présent de X_j. Il y a donc un arête non orientée (X_i, X_j).

IV.3.2.2 SALZA

Nous pouvons maintenant relier SALZA et la causalité de Granger pour deux raisons. La première est que LZ77 est considéré comme un prédicteur optimal [FMG92]. Nous pouvons donc repartir de la définition donnée par Granger en 1969. La deuxième est que grâce à SALZA, nous avons défini l'information dirigée algorithmique. Il est donc possible de transposer les définitions basées sur la théorie probabiliste de l'information à la théorie algorithmique de l'information. Cela permet, de la même façon qu'Amblard et Michel dans [AM11], de définir la causalité de Granger avec l'information dirigée algorithmique.

Il est possible de transposer la définition de l'information dirigée conditionnelle comme suit :

$$I(X \to Y || Z) = S_f(Y_- |^{\mathrm{p}} X) - S_f(Y_- |^{\mathrm{p}} X, Z).$$

C'est à dire que l'ajout de la connaissance du passé de Z n'améliore pas la prédiction de Y par rapport à la connaisance d'uniquement le passé de X.

Le graphe de causalité basé sur la causalité de Granger et l'information dirigée algorithmique est définie comme suit.

Définition IV.3.6 (Graphe de causalité de Granger basé sur l'information dirigée algorithmique)**.**
Soit $\mathcal{X} = \{X_1, ..., X_n\}$ *un ensemble de chaînes de caractères, alors le graphe de causalité de Granger* $G = (\mathcal{X}, E^u, E^d)$ *basé sur l'information dirigée algorithmique est défini* $\forall i = 1...n, j = 1...n$ *comme :*

$$(X_i, X_j) \notin E^d \Leftrightarrow S_f(X_{j\_} |^p \mathcal{X} \backslash \{X_i, X_j\}) = S_f(X_{j\_} |^p \mathcal{X} \backslash \{X_i, X_j\}, \mathtt{R}X_i)$$

et

$$(X_i, X_j) \notin E^u \Leftrightarrow S_f(X_{j\_} |^p \mathcal{X} \backslash \{X_i, X_j\}, \mathtt{R}X_i) = S_f(X_{j\_} |^p \mathcal{X} \backslash \{X_j\})$$

Voilà comment il est possible d'interpréter la causalité de Granger algorithmique. Si X_i

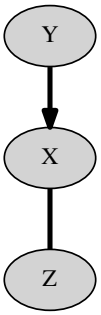


FIGURE IV.4 – Représentation des liens de causalité entre X, Y et Z.

cause X_j cela signifie que si le passé de X_i est connu alors il est possible de faire de plus longues références pour décomposer X_j. Si X_i cause instantanément X_j cela veut dire que si l'octet présent de X_i est connu alors il est possible de faire de plus longues références pour décomposer X_j.

Pour bien comprendre la causalité, prenons une exemple où $\mathcal{X}$ est composé de trois chaînes, X, Y et Z. X est une chaîne aléatoire, $Y = \mathtt{R}X$ est X décalée de 1 et Z alterne 5 caractères de X puis 5 caractères aléatoires et ainsi de suite.

Pour décomposer Y la connaissance de X ou de Z n'apporte rien. Ce sont toutes les deux des chaînes aléatoires qui n'aident en rien à mieux prédire l'instant présent de Y. Donc Y n'est pas causé par Z ou par X. Le même raisonnement peut être appliqué à Z qui n'est causé ni par Y, ni par X. Par contre X est causé par Y : le passé de Y permet d'expliquer entièrement X. Enfin nous pouvons voir que X et Z se causent de façon instantanée car l'instant présent de l'un permet de prédire l'instant présent de l'autre une fois sur deux. Nous pouvons observer la représentation de ces liens de causalité dans la figure IV.4. La flèche représente la causalité entre Y et X et le trait représente la causalité instantanée entre X et Z.

En résumé

La séquence X_i cause la séquence X_j si et seulement si

causalité de Granger probabiliste :

$$I(\mathtt{R}X_i \to X_j || \mathcal{X} \backslash \{X_i, X_j\}) > 0.$$

causalité de Granger algorithmique :

$$S_f(X_{j\_}|^{\mathrm{p}}\mathcal{X}\backslash\{X_i, X_j\}) - S_f(X_{j\_}|^{\mathrm{p}}\mathcal{X}\backslash\{X_i, X_j\}, \mathtt{R}X_i) > 0.$$

IV.3.3 Résultats

Pour tester la causalité de Granger basée sur `SALZA`, nous utilisons des données simulées. Il faut dans un premier temps définir une matrice de causalité G pour les K processus. Cela signifie que pour la chaîne i, il y a $p_{1,i}$ chances que la chaîne 1 génère la chaîne i, $p_{2,i}$ chances que la chaîne 2 génère la chaîne i ... et $p_{K+1,i}$ chances que la chaîne soit générée aléatoirement. Une fois cette matrice spécifiée, nous utilisons l'algorithme 4 pour générer les données.

Algorithme 4 Simulation des données pour la causalité de Granger

Entrée : G : la matrice de causalité désirée des K processus
Entrée : mc : la longueur maximale des copies
Entrée : L : la longueur désirée des fichiers

générer K séquences aléatoires de taille mc
$l \leftarrow mc$
tant que $l < L$ **faire**
  c : entier aléatoire entre 1 et mc
  p : entier aléatoire entre 1 and $l - c$
  pour $i = 1..K$ **faire**
    g : choisir une séquence génératrice selon la matrice G
    ajouter à la séquence i :
    si g==bruit **alors**
      une séquence aléatoire de longueur c
    sinon
      `copier` depuis la séquence g de la position p à la position $p + c$
    fin si
  fin pour
  $l \leftarrow l + c$
fin tant que

Nous avons testé la causalité de Granger basée sur `SALZA` sur plusieurs matrices mais nous ne présentons ici que deux matrices de génération affichées dans les tableaux IV.4a et IV.5a. Leurs graphes de Granger associés sont affichés sur les figures IV.5a et IV.6a.

Nous contraignons la longueur maximale des opérations `copier` à 300, $mc = 300$ et les fichiers ont une longueur de 32000 octets, $L = 32000$. Les résultats numériques normalisés (en divisant par L) obtenus avec `SALZA` sont affichés dans les tableaux IV.4b et IV.5b. Leurs graphes de Granger associés sont affichés dans les figures IV.5b et IV.6b.

L'épaisseur des flèches ainsi que leur opacité est une fonction linéaire de la condition de causalité $S_f(X_{j\_}|^{\mathrm{p}}\mathcal{X}\backslash\{X_i, X_j\}) - S_f(X_{j\_}|^{\mathrm{p}}\mathcal{X}\backslash\{X_i, X_j\}, \mathtt{R}X_i)$. La condition de causalité n'est jamais égale strictement à zéro. Comme l'ont suggéré Amblard et Michel ([AM11]), il faut définir un seuil en pratique. Nous pouvons voir que dans tous les cas, il est très facile de poser un seuil qui permet de retrouver la représentation initiale. En effet, il y a un facteur 10 entre les liens véritables et les fausses alarmes. `SALZA` permet donc bien d'estimer la causalité de Granger sur nos données simulées.

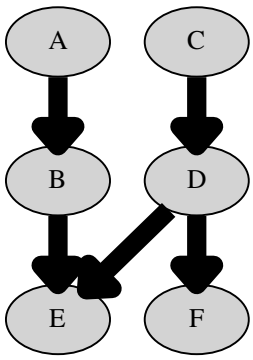
	A	B	C	D	E	F
A	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
B	5,00e-01	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
C	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
D	0,00e+00	0,00e+00	5,00e-01	0,00e+00	0,00e+00	0,00e+00
E	0,00e+00	5,00e-01	0,00e+00	5,00e-01	0,00e+00	0,00e+00
F	0,00e+00	0,00e+00	0,00e+00	5,00e-01	0,00e+00	0,00e+00

(a) Pour la génération.

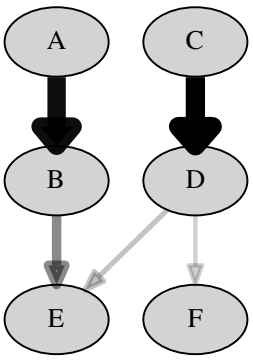
	A	B	C	D	E	F
A	0,00e+00	1,20e-03	2,70e-03	1,12e-03	2,43e-05	2,33e-03
B	3,05e-01	0,00e+00	1,27e-03	5,86e-04	6,62e-06	1,05e-03
C	2,12e-03	1,20e-03	0,00e+00	1,09e-03	7,35e-05	1,83e-03
D	1,11e-03	5,49e-04	3,21e-01	0,00e+00	8,83e-06	7,65e-04
E	0,00e+00	1,49e-01	0,00e+00	7,46e-02	0,00e+00	0,00e+00
F	1,11e-03	2,93e-04	1,45e-03	5,98e-02	1,10e-05	0,00e+00

(b) Estimée grâce à `SALZA` avec $\alpha = 256$.

Table IV.4 – Matrice de causalité.



(a) Pour la génération.



(b) Estimée grâce à `SALZA` avec $\alpha = 256$.

Figure IV.5 – Représentation graphique de la causalité de Granger.

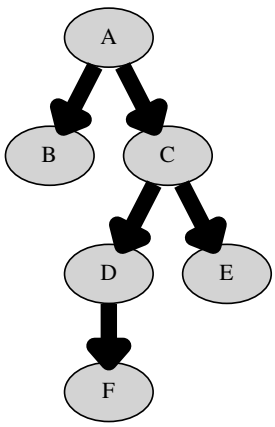
	A	B	C	D	E	F
A	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
B	8,00e-01	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
C	8,00e-01	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
D	0,00e+00	0,00e+00	8,00e-01	0,00e+00	0,00e+00	0,00e+00
E	0,00e+00	0,00e+00	8,00e-01	0,00e+00	0,00e+00	0,00e+00
F	0,00e+00	0,00e+00	0,00e+00	8,00e-01	0,00e+00	0,00e+00

(a) Pour la génération.

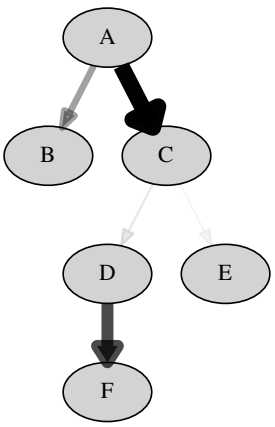
	A	B	C	D	E	F
A	0,00e+00	1,62e-04	4,41e-05	7,40e-05	1,84e-04	1,67e-04
B	4,60e-02	0,00e+00	-1,12e-05	-1,02e-05	3,44e-05	-1,86e-05
C	1,24e-01	-1,33e-05	0,00e+00	7,52e-06	-3,27e-06	2,14e-05
D	-7,01e-06	8,72e-05	1,22e-02	0,00e+00	4,22e-05	-2,20e-06
E	-2,68e-05	9,08e-06	6,74e-03	1,26e-05	0,00e+00	4,25e-06
F	5,66e-05	-3,95e-05	4,03e-06	8,82e-02	-3,08e-05	0,00e+00

(b) Estimée grâce à `SALZA` avec $\alpha = 256$.

Table IV.5 – Matrice de causalité.



(a) Pour la génération.



(b) Estimée grâce à `SALZA` avec $\alpha = 256$.

Figure IV.6 – Représentation de la causalité.

Une application réelle de la causalité de Granger algorithmique pourrait être envisagée sur des signaux électroencéphalographiques (EEG).

Premièrement, la complexité de Lempel-Ziv a été utilisée sur des signaux électroencéphalographiques (EEG) dans la détection des yeux ouverts ou fermés ou encore dans la détection de la fatigue musculaire ([IM+15], [Li+08], [Abo+06]). Le principal enjeu de ces méthodes est la quantification des signaux. En effet, les signaux EEG sont des signaux analogiques qui sont trasnformés en signaux numériques à l'acquisition. Ces signaux numériques sont, en général, quantifiés sur 16 bits et sont très bruités. C'est pourquoi, la plupart du temps les signaux EEG sont quantifiés sur 1 bit. Beaucoup d'information est alors perdue, il ne reste que l'information basse fréquence des signaux. Il serait donc très intéresant d'étudier plus en profondeur les problématiques de quantifications sur de telle signaux et d'utiliser `SALZA` qui permet, comme nous l'avons vu, une meilleure adaptation aux données.

Deuxièmement, la causalité de Granger basée sur l'entropie de Shannon et l'information dirigée probabiliste a aussi été utilisée sur des signaux EEG pour estimer le réseau neuronal ([Qui+11], [Hes+03]). C'est-à-dire qu'ils cherchent comment les neurones s'influencent entre eux.

Il serait donc intéressant d'utiliser la causalité de Granger algorithmique basée sur la complexité pour estimer le réseau neuronal une fois que la problématique de la quantification sera correctement étudiée. En effet, `SALZA` peut s'adapter aux données grâce à sa fonction admissible contrairement à la complexité de Lempel-Ziv (voir section II.4.1). La quantification sur de tels signaux pourrait contenir plus de 1 bit ce qui permettrait de garder plus d'information.

En résumé

`SALZA` et la causalité de Granger :

- *Validation* : nous retrouvons le bon graphe pour des données simulées.
- *Perspectives* : la complexité et la causalité de Granger ont été utilisées indépendamment sur des signaux électroencéphalographiques (EEG), une application réelle envisageable serait d'utiliser la causalité de Granger basée sur `SALZA` sur des signaux EEG.

Conclusion

L'estimation des relations causales entre différents processus a suscité un intérêt croissant au cours de ces dernières années. Les mesures d'indépendance classiques permettent uniquement de montrer l'existence d'une relation entre les différentes variables aléatoires. Les méthodes de causalité sont, quant à elles, conçues pour générer automatiquement des hypothèses sur la direction des liens entre les différents processus. La plupart de ces méthodes reposent sur des modèles probabilistes.

Nous avons vu comment définir la condition de Markov et la causalité de Granger grâce à `SALZA` rendant ainsi ces mesures indépendantes du type de données ou encore de modèles probabilistes. La condition de Markov a été validée sur un jeu de données réelles avec trois processus. Nous avons montré qu'elle peut aussi servir à retrouver le cheminement d'écriture d'un auteur. La causalité de Granger n'a été validée que sur un jeu de données simulées. Une des perspectives de ce travail pourrait être de l'appliquer à des signaux EEG sur lesquels la causalité de Granger et la complexité de Lempel-Ziv ont été utilisées indépendamment.

Conclusion et perspectives

Conclusion

Les données sous forme de chaînes de caractères sont de plus en plus nombreuses et sont extrêmement variées, allant des textes littéraires numérisés à des fichiers MIDI de musique en passant par des signaux électroencéphalographiques quantifiés. Une mesure de description universelle, indépendante de tout modèle ou hypothèse probabiliste, est donc primordiale. Cette mesure s'appelle la complexité de Kolmogorov et elle repose sur une idée très simple : une chaîne de caractères est complexe quand il n'en existe pas une description courte. Au cours de nos travaux, nous avons présenté la théorie algorithmique de l'information reposant sur la complexité de Kolmogorov (chapitre I) et un nouvel estimateur `SALZA` de cette complexité (chapitre II). Puis nous avons montré comment appliquer la complexité pour une classification universelle des chaînes de caractères (chapitre III) et pour l'inférence de causalité (chapitre IV).

Au cours du chapitre I, nous avons introduit la complexité de Kolmogorov et ses estimateurs. Pour bien comprendre son fonctionnement, nous avons rappelé qu'une manière d'évaluer l'information contenue dans une chaîne de caractères de taille finie sur un alphabet de taille finie est l'entropie de Shannon (section I.1.1). Cette mesure repose sur les probabilités contrairement à la complexité de Kolmogorov qui ne s'appuie que sur l'algorithmique : la complexité d'une chaîne est la taille du plus petit programme capable de la générer (section I.1.2). Tout comme pour l'entropie de Shannon, il est possible de définir le pendant algorithmique de la théorie probabiliste de l'information qui est alors appelée théorie algorithmique de l'information (section I.1.3). Cependant, la complexité de Kolmogorov n'est pas calculable en un temps fini ce qui la rend inutilisable en pratique. Les premiers à rendre la complexité de Kolmogorov opérationnelle sont Lempel et Ziv qui proposent de restreindre les opérations possibles à `copier` et `insérer` (section I.2.1). La complexité est alors le nombre minimal, de ces deux opérations élémentaires, nécessaires pour générer la chaîne. Nous avons vu qu'il n'existe pas une unique définition de la complexité de Lempel-Ziv jointe et conditionnelle, ce qui empêche une estimation efficace des grandeurs de la théorie algorithmique de l'information (section I.2.3). Une autre façon d'estimer la complexité de Kolmogorov est d'utiliser un compresseur sans perte (section I.3.2). En effet, le compresseur parfait serait celui qui créerait le plus petit fichier à partir du fichier d'entrée. Cette définition n'est pas sans rappeler celle de la complexité de Kolmogorov et il est donc possible d'estimer cette dernière par la taille du fichier compressé. Nous avons vu que la famille des compresseurs basés sur la complexité de Lempel-Ziv (`GZIP`, `LZMA`, section I.3.1) respecte les propriétés des compresseurs normaux et qu'il est donc possible de les utiliser comme estimateur. Cependant, il est impossible de compresser un fichier connaissant un autre fichier et il faut avoir recours à des approximations pour calculer la taille d'un fichier compressé connaissant un autre fichier. Cela empêche, tout comme la complexité de Lempel-Ziv, de définir correctement la théorie algorithmique de l'information basée sur les compresseurs.

Partant de ce constat, nous avons proposé un nouvel estimateur de complexité de Kol-

mogorov basé sur la complexité de Lempel-Ziv dans le chapitre II. Cette estimateur appelé `SALZA` est implémenté sur les bases de l'algorithme `GZIP` basé lui-même sur la complexité de Lempel-Ziv (section II.1.1). Différents types de conditionnements ont été définis, qui ont permis de pallier le problème de la mauvaise définition de la complexité de Lempel-Ziv conditionnelle (section II.1.2). Puis nous avons mis en évidence que le seul nombre des opérations ne retient pas toute l'information. C'est pourquoi nous avons introduit la *quantité significative* qui repose sur la longueur des opérations `copier` et mesure le nombre de caractères de la chaîne bien expliqués (section II.1.3). Ces trois points ont permis de proposer l'estimateur appelé `SALZA` basé sur la complexité de Lempel-Ziv et la *quantité significative* (section II.2.1). `SALZA` est une mesure d'information pour le cas particulier de la fonction indicatrice. Nous avons cependant observé qu'avec les autres fonctions admissibles étudiées cette propriété est très souvent respectée (section II.2.3). Il peut exister plusieurs définitions de `SALZA` jointe selon la manière dont nous interprétons la meilleure décomposition de X et Y (section II.3.1). Grâce à ces définitions, il a alors été facile de définir l'information mutuelle algorithmique qui mesure la dépendance entre les deux variables X et Y ainsi que, pour la première fois, l'information dirigée algorithmique qui mesure la dépendance directionnelle entre ces deux chaînes de caractères (section II.3.2). L'implémentation et la bonne définition de notre mesure permettent alors un calcul efficace des grandeurs de la théorie algorithmique de l'information. Enfin, le comportement de `SALZA` a été comparé à celui de la complexité de Lempel-Ziv et des compresseurs. Dans un premier temps, nous avons réalisé une étude théorique en supposant une distribution sur la longueur des symboles pour comparer `SALZA` et la complexité de Lempel-Ziv (section II.4.1). Dans un deuxième temps, une simulation a été réalisée pour comparer les compresseurs et `SALZA` (section II.4.2). Nous avons observé que la prise en compte des longueurs permet une discrimination plus fine autour de la longueur seuil choisie ce qui permet de s'adapter au type de la chaîne (par exemple la taille de son alphabet ou sa longueur).

La complexité, qui est une mesure de description universelle, permet de réaliser une classification universelle indépendante de toute hypothèse probabiliste ou du type de données étudiées comme nous l'avons illustré dans le chapitre III. Nous avons présenté trois métriques qui reposent sur la complexité (section III.1). La première est la divergence de Ziv-Merhav qui repose sur la complexité de Lempel-Ziv, la seconde est la *NCD* qui utilise la taille des fichiers compressés et enfin la troisième est la *NSD* qui utilise `SALZA`. Ces trois métriques basées sur la complexité ont ensuite été utilisées pour réaliser une classification ascendante hiérarchique sur différents jeux de données : des chaînes de Markov, des ADN de mammifères, la Déclaration des Droits de l'Homme en cinquante langues indo-européennes et des romans de la littérature française (section III.3). Tout d'abord, par construction, la *NSD* est la seule métrique qui produise une matrice correcte pour les algorithmes de classification. En effet la *NCD* et la divergence de Ziv-Merhav ne respectent pas les propriétés de symétrie, d'égalité des indiscernables ou de positivité. Nous avons vu ensuite que la *NSD* permettait une meilleure classification sur toutes les données que nous avons testées. Ceci est principalement dû au conditionnement approprié que nous utilisons ainsi que la bonne prise en compte des longueurs grâce à la *quantité significative*.

Enfin nous nous sommes intéressés à l'inférence de la causalité par la complexité dans le chapitre IV. La causalité permet d'étudier la structure de causalité des différentes chaînes d'un jeu de données et d'inférer un modèle graphique sous-jacent (section IV.1). Cependant, la plupart des méthodes d'inférence reposent sur des modèles probabilistes. Nous avons

donc proposé deux méthodes pour inférer la causalité algorithmiquement grâce à `SALZA`, rendant cette estimation universelle et indépendante du type de données. Tout d'abord, nous avons rendu opérationnelle la condition de Markov algorithmique qui dit qu'une chaîne doit être indépendante de ses non-descendants connaissant ses parents (section IV.2). La condition de Markov a été validée sur un jeu de données réelles avec trois chaînes ainsi que sur un jeu de données contenant huit chaînes ce qui permet de retrouver le cheminement d'écriture d'un auteur. Ensuite, nous avons défini la causalité de Granger grâce à l'information dirigée algorithmique définie pour la première fois grâce à notre bon conditionnement (section IV.3). La causalité de Granger n'a été validée que sur un jeu de données simulées.

Perspectives

Ainsi au cours de ces travaux, nous avons dans un premier temps clarifié et proposé de nouveaux conditionnements pour la complexité de Lempel-Ziv. Ceci nous a permis d'établir un nouvel estimateur de la complexité de Kolmogorov qui se base sur le nombre d'opérations `copier` et `insérer` mais aussi sur leur longueur. La clarification sur le conditionnement que nous avons réalisée a rendu possible l'élaboration d'un nouvel encodage de la complexité jointe et ainsi de l'information dirigée algorithmique. L'implémentation et la bonne définition de notre mesure permettent alors un calcul efficace des grandeurs de la théorie algorithmique de l'information. De plus, en utilisant notre estimateur `SALZA` au lieu des compresseurs pour estimer une distance, nous avons pu observer que nous obtenons une meilleure classification. Cette amélioration est due au bon conditionnement que nous avons utilisé et à la prise en compte des longueurs des symboles. Enfin, nous avons proposé une approche algorithmique de la causalité en utilisant la condition de Markov algorithmique grâce à `SALZA` et en introduisant la causalité de Granger algorithmique.

Quatre perspectives plus ou moins complexes pourraient être explorées pour approfondir cette recherche.

Causalité

La première serait dans la continuité directe des travaux que nous avons présentés. En effet, les développements que nous avons présentés sont des résultats préliminaires à un travail à plus long terme portant sur l'inférence de causalité algorithmique. Les applications sur des données réelles et simulées semblent montrer que l'approche algorithmique fonctionne correctement. Cependant, une validation sur d'autres jeux de données reste à faire, et permettrait de conforter les premiers résultats encourageants obtenus.

Rajouter des opérations

Comme nous l'avons vu (section I.2.1.3 et section III.1.4), d'autres opérations que les simples `copier` et `insérer` pourraient être prises en compte. Une autre opération pourrait être `copier retourner` qui réalise une opération copier mais en partant du dernier

caractère au premier. Cela permettrait d'observer d'autres phénomènes que de simples répétitions et ferait tendre notre estimateur vers la complexité de Kolmogorov (étant donné que la restriction des opérations serait moins forte). Il serait donc intéressant de répertorier les opérations qui se révèlent pertinentes.

Une fois ce premier travail réalisé, il faudra analyser théoriquement l'influence de ces nouvelles opérations. Il faudrait montrer qu'avec elles `SALZA` reste bien une mesure d'information. La deuxième étape serait alors de rajouter ces opérations à l'implémentation de `SALZA`. Il faudrait alors comparer l'amélioration des résultats et la perte de temps que cela engendre. En effet, comme plus d'opérations sont disponibles alors forcément le temps de calcul pour la décomposition sera plus long. Enfin, en rajoutant énormément d'opérations, il faudra vérifier que nous ne tombons pas dans l'écueil du surapprentissage. Les opérations correspondraient trop étroitement à un ensemble de chaînes de caractères particulier ce qui fait que `SALZA` ne pourrait pas s'appliquer de manières fiables à d'autres chaînes de caractères.

Quantification

Au cours de cette recherche, nous n'avons utilisé que des données sur un alphabet fini. Cependant aujourd'hui, dans le monde numérique, tous les signaux analogiques stockés sur un ordinateur sont échantillonnés en temps et quantifiés en amplitude. Ces signaux deviennent alors des chaînes de taille finie sur un alphabet fini. C'est par exemple le cas des signaux électroencéphalographiques en biomédical. Sur ces signaux, la complexité de Lempel-Ziv a été utilisée à maintes reprises par exemple pour détecter la fatigue musculaire [TCM11], la maladie de Parkinson [FB11] ou encore la schizophrénie et la dépression [Li+08]. L'utilisation pour les signaux biomédicaux de la complexité de Lempel-Ziv a été synthétisée par Aboy et al. [Abo+06]. Cependant dans toutes ces applications, le signal est quantifié sur un bit : si le caractère est supérieur à un certain seuil T alors il est associé à 1 sinon à 0. Le signal perd alors forcément et irrémédiablement de l'information. Cependant, `SALZA` peut s'adapter aux données grâce à sa fonction admissible contrairement à la complexité de Lempel-Ziv (voir section II.4.1). La quantification sur de tels signaux pourrait être moins forte ce qui permettrait de garder plus d'information haute fréquence. Il serait donc intéressant d'étudier l'influence de la quantification de signaux analogiques sur le fonctionnement de `SALZA`.

Transposer des applications probabilistes au domaine algorithmique

Une dernière perspective à plus long terme serait de trouver d'autres applications probabilistes à transposer en algorithmique. Une des applications possibles serait une analyse *complexité-fréquence* comme l'ont proposée Ibanez et al. [IM+15]. En effet, la quantification sur un bit, évoquée précédemment, ne permet que de retrouver les changements basse fréquence. En faisant varier une fenêtre, ils proposent ainsi une analyse basée sur la complexité de Lempel-Ziv selon différentes fréquences. Grâce à cela, ils arrivent à trouver la différence entre des patients avec les yeux ouverts et les yeux fermés. De plus, cette approche a aussi été développée de manière plus approfondie grâce à la mesure de l'entropie

[CGP05]. Il semble donc tout à fait logique et possible de transposer ce concept dans le domaine algorithmique grâce à la complexité et en particulier à `SALZA`. La transposition d'applications probabilistes au domaine algorithmique permettrait ainsi d'utiliser la propriété d'universalité de la complexité et donc de s'affranchir des contraintes de modèles sur les données.

Annexe A

Démonstrations du chapitre II

Dans ce chapitre nous présentons les preuves des propriétés et des théroèmes abordés dans le chapitre II.

Propriété II.1.2 (Bornes de la *quantité significative*)**.**
Soient deux chaînes Y et $X \in \mathcal{A}^*$ et une fonction admissible f centrée en $l^\star$, alors

$$0 \leq W_f(X \wr Y) \leq l(X).$$

Démonstration. :

Borne inférieure Une fonction admissible est définie entre 0 et 1, $f : \mathbb{N}^* \to [0,1]$. Donc $W_f(X \wr Y) = \sum_{l \in \mathcal{L}_{X \wr Y}} lf(l) \geq \sum_{l \in \mathcal{L}_{X \wr Y}} l \times 0 = 0$. Cette borne est atteinte par exemple s'il existe l_{inf} telle que $\forall l \leq l_{inf}\ f(l) = 0$. Si toutes les longeurs dans $\mathcal{L}_{X \wr Y}$ sont inférieures à l_{inf}, alors $W_f(X \wr Y) = 0$.

Borne supérieure Une fonction admissible est définie entre 0 et 1, $f : \mathbb{N}^* \to [0,1]$. Donc $W_f(X \wr Y) = \sum_{l \in \mathcal{L}_{X \wr Y}} lf(l) \leq \sum_{l \in \mathcal{L}_{X \wr Y}} l \times 1 = l(X)$. Cette borne est atteinte quand par exemple toutes les longueurs de $\mathcal{L}_{X \wr Y}$ sont supérieures à T. Alors dans ce cas là, $\forall l \in \mathcal{L}_{X \wr Y}\ ,\ f(l) = 1$, et donc $W_f = l(X)$.

□

Propriété II.2.1 (Bornes de `SALZA`)**.**
Soient deux chaînes Y et $X \in \mathcal{A}^*$ et une fonction admissible f centrée en $l^\star$, alors l'estimateur de complexité `SALZA`, S_f, satisfait :

$$0 \leq S_f(X \wr Y) \leq l(X) - 1$$

Démonstration. :

Borne supérieure Les longueurs sont comprises entre 0 et $l(X)$ et $f(l)$ est comprise entre 0 et 1. Donc $\forall l \in \mathcal{L}_{X \wr Y}\ ,\ (l-1)f(l) \geq 0$. Donc $S_f(X \wr Y) = l(X) - \sum_{l \in \mathcal{L}_{X \wr Y}} (l-1)f(l) \leq l(X)$. Cette borne est atteinte par exemple quand tous les symboles émis sont des littéraux.

Borne inférieure Une fonction admissible est définie entre 0 et 1, $f : \mathbb{N}^* \to [0,1]$. Donc $\sum_{l \in \mathcal{L}_{X \wr Y}} (l-1)f(l) \leq \sum_{l \in \mathcal{L}_{X \wr Y}} lf(l) \leq \sum_{l \in \mathcal{L}_{X \wr Y}} l \times 1 = l(X)$. On a donc bien $S_f(X \wr Y) \geq 0$.

FIGURE A.1 – Représentation de la décomposition de $X \wr Y$ (en haut) et $X \wr Y, Z$ (en bas).

Cette borne est atteinte quand on ne fait qu'une seule et unique référence pour expliquer X depuis Y.

□

Théorème II.2.1 (Ajout de connaissance)**.**
Soient X, Y et $Z \in \mathcal{A}^*$ et l'estimateur de complexité ***SALZA*** *avec la fonction admissible indicatrice $\mathbb{1}$. L'estimateur $S_{\mathbb{1}}$ satisfait alors :*

$$S_{\mathbb{1}}(X \wr Y, Z) \leq S_{\mathbb{1}}(X \wr Y).$$

Démonstration. La preuve si dessous est une preuve par récurrence.

Il est nécessaire pour la preuve de définir les notations suivantes. Soient X, Y et Z trois chaînes dans $\mathcal{A}^*$, alors $\mathcal{L}_{X \wr Y}$ est l'ensemble des longueurs de la décomposition $X \wr Y$. On note $\mathcal{L}^i_{X \wr Y}$ les i premières longueurs de la décomposition $X \wr Y$. De même $\mathcal{L}_{X \wr Y, Z}$ est l'ensemble des longueurs de la décomposition $X \wr Y, Z$ et on note $\mathcal{L}^i_{X \wr Y, Z}$ les i premières longueurs de la décomposition $X \wr Y, Z$.

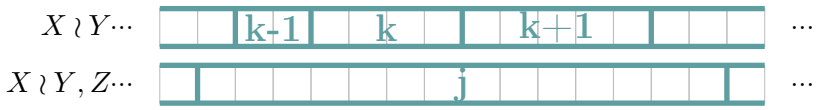
Hypothèse de récurrence :

$$H_k : \text{soit } j \text{ le plus petit entier tel que } \sum_{l \in \mathcal{L}^k_{X \wr Y}} l \leq \sum_{l \in \mathcal{L}^j_{X \wr Y, Z}} l, \text{ alors } j \leq k$$

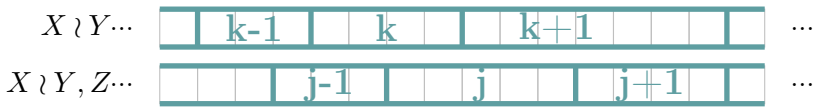
Pour bien comprendre l'hypothèse de récurrence, regardons la figure A.1 où la première ligne représente la décomposition de $X \wr Y$ et la deuxième ligne la décomposition $X \wr Y, Z$. Les carrés gris sont les octets de X. Les rectangles de couleur représentent les références avec leur numéro, par exemple la référence numéro 1 de $X \wr Y$ fait 4 octets et la référence numéro j de $X \wr Y, Z$ fait 6 octets. Revenons à l'hypothèse de récurrence, et expliquons ce qu'elle signifie : trouver le plus petit entier j tel que $\sum_{l \in \mathcal{L}^k_{X \wr Y}} l \leq \sum_{l \in \mathcal{L}^j_{X \wr Y, Z}} l$. Pour le moment avec la décomposition $X \wr Y$, nous avons décomposé $\sum_{l \in \mathcal{L}^k_{X \wr Y}} l$ octets de X. Nous cherchons le plus petit nombre de références j qui permet d'expliquer au moins autant d'octets de X mais avec la décomposition $X \wr Y, Z$. Nous montrons alors que pour expliquer le même nombre d'octets de X, nous faisons moins de références en ajoutant la connaissance de Z.

Pour la preuve par récurrence, nous démontrons que H_0 est vrai puis que $H_k \Rightarrow H_{k+1}$.

(a) Cas 1.



(b) Cas 21.



(c) Cas 22 .

FIGURE A.2 – Représentation de la décomposition de $X \wr Y$ (en haut) et$X \wr Y, Z$(en bas) dans différents cas.

- H_0
  Lors de la décomposition de X connaissant Y, la première référence est de taille $l^1_{X \wr Y}$.

L'ajout de la connaissance de Z permet au mieux de faire une référence de taille plus grande que $l^1_{X \wr Y}$ dans Z ou au pire de faire exactement la même référence dans Y. Donc $l^1_{X \wr Y} \leq l^1_{X \wr Y, Z}$. Ainsi le plus petit nombre de références j qui permet d'expliquer au moins autant d'octets de X mais avec la décomposition $X \wr Y, Z$ est 1.

On a donc $j = 1$ et $k = 1$.

- **En conclusion, H_0 est vrai .**

- $H_k \Rightarrow H_{k+1}$
  On sait que j est le plus petit entier tel que $\sum_{l \in \mathcal{L}^k_{X \wr Y}} l \leq \sum_{l \in \mathcal{L}^j_{X \wr Y, Z}} l$ et que $j \leq k$. Nous voulons donc montrer que sous cette hypothèse H_k , l'hypothèse H_{k+1} est vraie. Pour cela il y a trois cas possibles. Ces trois cas sont représenté dans la figure A.2 et sont détaillés ci-après.

Cas 1 : [figure A.2a]

$$\sum_{l \in \mathcal{L}^j_{X \wr Y, Z}} l = \sum_{l \in \mathcal{L}^k_{X \wr Y}} l$$

Dans ce cas là, nous avons expliqué exactement le même nombre d'octets en connaissant seulement Y qu'en connaissant Z et Y avec respectivement k symboles et j symboles. Nous

nous retrouvons donc dans le même cas que H_0. En connaissant Y, nous faisons une nouvelle référence $l^{k+1}_{X\wr Y}$. Si on rajoute Z comme connaissance, alors au mieux on fait une référence de taille plus grande que $l^{k+1}_{X\wr Y}$ ou au pire, on fait exactement la même référence dans Y. Donc $l^{k+1}_{X\wr Y} \leq l^{j+1}_{X\wr Y,Z}$.

Ainsi le plus petit entier j tel que $\sum_{l\in\mathcal{L}^{k+1}_{X\wr Y}} l \leq \sum_{l\in\mathcal{L}^{J}_{X\wr Y,Z}} l$ est $j+1$. Or, d'après H_k on a $j \leq k$, donc $J = j+1 \leq k+1$.

Cas 1 : l'hypothèse de récurrence $H_k \Rightarrow H_{k+1}$ est vérifiée.

Cas 2 :

$$\sum_{l\in\mathcal{L}^{j}_{X\wr Y,Z}} l > \sum_{l\in\mathcal{L}^{k}_{X\wr Y}} l$$

Ici, nous avons expliqué plus d'octets de X avec j symboles en connaissant Z et Y qu'avec k symboles en connaissant Y. Ce cas peut être découpé en deux sous-cas.

Cas 21 : figure A.2b

$$\sum_{l\in\mathcal{L}^{j}_{X\wr Y,Z}} l \geq \sum_{l\in\mathcal{L}^{k+1}_{X\wr Y}} l$$

Dans cette configuration, la référence j de la décomposition $X\wr Y, Z$ est tellement grande qu'elle englobe même la référence $k+1$ de la décomposition $X\wr Y$. Ainsi le plus petit entier J tel que $\sum_{l\in\mathcal{L}^{k+1}_{X\wr Y}} l \leq \sum_{l\in\mathcal{L}^{J}_{X\wr Y,Z}} l$ est j. Or, d'après H_k on a $j \leq k$, donc $J = j \leq k+1$.

Cas 21 : l'hypothèse de récurrence $H_k \Rightarrow H_{k+1}$ est vérifiée.

Cas 22 : figure A.2c

$$\sum_{l\in\mathcal{L}^{j}_{X\wr Y,Z}} l < \sum_{l\in\mathcal{L}^{k+1}_{X\wr Y}} l$$

Dans ce cas là la référence, j de la décomposition $X\wr Y, Z$ n'est pas assez grande pour englober la référence $k+1$ de la décomposition $X\wr Y$. Il se peut que la référence $l^{j+1}_{X\wr Y}$ soit plus petite que la la référence $l^{k+1}_{X\wr Y}$ (c'est le cas dans la figure A.2c). Cependant dans le pire des cas avec $j+1$ références avec la décomposition $X\wr Y, Z$ nous expliquons toujours au moins autant d'octets qu'avec $k+1$ références avec la décomposition $X\wr Y$. En effet,

la référence $k+1$ est dans Y donc au pire, il est possible de faire référence dans Y pour compléter la référence j et arriver au même nombre d'octets expliqués.

Ainsi le plus petit entier J tel que $\sum_{l \in \mathcal{L}^{k+1}_{X \wr Y}} l \leq \sum_{l \in \mathcal{L}^{J}_{X \wr Y, Z}} l$ est $j+1$. Or, d'après H_k on a $j \leq k$, donc $J = j+1 \leq k+1$.

Cas 22 : l'hypothèse de récurrence $H_k \Rightarrow H_{k+1}$ est vérifiée.

Dans tous les cas : l'hypothèse de récurrence $H_k \Rightarrow H_{k+1}$ est vérifiée.

Par le principe de récurrence on peut donc conlcure que l'hypothèse est vraie pour tout k. □

Théorème II.2.2 (Mesure d'information)**.**
Soit $\mathcal{A}$ un alphabet, $\mathcal{A}^*$ l'ensemble des chaînes définies sur $\mathcal{A}^*$ et f une fonction admissible. Alors ***SALZA*** *est une mesure d'information pour la fonction admissible indicatrice $\mathbb{1}$ et vérifie les trois propriétés suivantes :*

1. *normalisation : $S_f(\Lambda) = 0$,*
2. *monotonie : $\forall X, Y \in \mathcal{A}^*, X \leq Y \Rightarrow S_{\mathbb{1}}(X) \leq S_{\mathbb{1}}(Y)$,*
3. *sous-modularité : $\forall X, Y \in \mathcal{A}^*, S_{\mathbb{1}}(X) + S_{\mathbb{1}}(Y) \geq S_{\mathbb{1}}(X) + S_{\mathbb{1}}(Y_{\cdot}|^{+}X)$.*

Démonstration. Il est important de noter que la preuve de la normalisation est valable pour toutes les fonctions admissibles.

Normalisation . La chaîne Λ est la chaîne vide donc aucun symbole n'est émis lors de sa décomposition : $\mathcal{L}_\Lambda = \emptyset$. Donc la somme est nulle, $\sum_{l \in \mathcal{L}_\Lambda} (l-1) f(l) = 0$. De plus par définition, nous avons $l(\Lambda) = 0$. On a donc bien pour toute fonction admissible f, $S_f(\Lambda) = 0$.

Monotonie . Soit $Y = XZ$. Alors forcément la décomposition de Y comptera au moins le même nombre de symbole que la décompositions de X.

Sous-modularité . La preuve est immédiate par le théorème II.2.1.

□

Propriété II.3.1 (SALZA jointe)**.**
L'encodage de deux chaînes X et $Y \in \mathcal{A}^*$, (X, Y), permet de calculer une complexité jointe notée $S_f(X, Y)$ avec f une fonction admissible, si ***SALZA*** *respecte les propriétés ci-après :*

1. *Positivité : $S_f(X, Y) \geq 0$,*
2. *Symétrie : $S_f(X, Y) = S_f(Y, X)$,*
3. *Invariance : $S_f(X, X) = S_f(X)$,*
4. *Borne inférieure : $S_f(X, Y) \geq max\{S_f(X), S_f(Y)\}$,*
5. *Borne supérieure : $S_f(X, Y) \leq S_f(X) + S_f(Y)$.*

Démonstration. Pour SALZA jointe basée sur l'encodage concaténé

$$S_f(X,Y)^{|u} = S_f(X) + S_f(Y_{\text{-}}|^{+}X)$$

1. Positivité : $S_f(X,Y)^{|u} = S_f(X) + S_f(Y_{\text{-}}|^{+}X) \geq 0 + 0 = 0$,
2. Symétrie : fausse, voir contre exemple présenté dans la section II.3.1,
3. Invariance : $S_f(X,X)^{|u} = S_f(X) + S_f(X_{\text{-}}|^{+}X) = S_f(X) + 1$,
4. Broen inférieure : $S_f(X,Y)^{|u} = S_f(X) + S_f(Y_{\text{-}}|^{+}X) \geq S_f(X)$,
5. Borne supérieure : $S_f(X,Y)^{|u} \leq S_f(X) + S_f(Y)$ par le théorème II.2.1.

□

Démonstration. Pour SALZA jointe basée sur l'encodage simultané

$$S_f(X,Y)^{|s} = S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y_{\text{-}}|^{\text{p}}X)$$

1. Positivité : $S_f(X,Y)^{|s} = S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y_{\text{-}}|^{\text{p}}X) \geq 1 + 1 > 0$,
2. Symétrie : fausse, voir contre exemple dans présenté dans la section II.3.1,
3. Invariance : $S_f(X,X)^{|s} = S_f(X_{\text{-}}|^{\text{-}}X) + S_f(X_{\text{-}}|^{\text{p}}X) = S_f(X) + 1$,
4. Borne supérieure $S_f(X,Y)^{|s} = S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y_{\text{-}}|^{\text{p}}X) \leq S_f(X) + S_f(Y)$ par le théorème II.2.1.

□

Propriété II.3.3 (Information dirigée basée sur l'entropie de Shannon)**.**
Toute information dirigée I doit respecter ces propriétés :

1. *Positivité : $I(X \to Y) \geq 0$,*
2. *Borne supérieure : $I(X \to Y) \leq I(X:Y)$, l'égalité est vérifiée quand il n'y a pas de rétroaction entre X et Y,*
3. *Décomposition : $I(X \to Y) + I(\mathtt{R}Y \to X) = I(X:Y)$.*

Démonstration. Pour l'information dirigée.

1. Poitivité : $I_S(X \to Y) = S_f(Y) - S_f(Y_{\text{-}}|^{\text{p}}X) \geq 0$ par le théorème II.2.1.
2. Brone supérierue : pour cela nous utilisons l'information mutuelle algorithmique basée sur l'encodage simultané : $I(X:Y)^{|s} - I_S(X \to Y) = S_f(X) - S_f(X_{\text{-}}|^{\text{-}}Y) + S_f(Y) - S_f(Y_{\text{-}}|^{\text{p}}X) - (S_f(Y) - S_f(Y_{\text{-}}|^{\text{p}}X)) = S_f(X) - S_f(X_{\text{-}}|^{\text{-}}Y) \geq 0$ par le théoreme II.2.1. S'il n'y a pas de rétroaction alors Y n'apporte aucune information à X : $S_f(X_{\text{-}}|^{\text{-}}Y) = S_f(X)$. Donc nous avons bien l'égalité entre les deux informations : $I(X:Y)^{|s} - I_S(X \to Y) = S_f(X) - S_f(X) = 0$.
3. Décomposition : notons que $S_f(X_{\text{-}}|^{\text{p}}\mathtt{R}Y) = S_f(X_{\text{-}}|^{\text{-}}Y)$. On a donc $I_S(X \to Y) + I_S(\mathtt{R}Y \to X) = S_f(Y) - S_f(Y_{\text{-}}|^{\text{p}}X) + S_f(X) - S_f(X_{\text{-}}|^{\text{p}}\mathtt{R}Y) = I_{S_f}(X:Y)^{|s}$.

□

Théorème II.4.1 (Pouvoir discriminant pour une loi de Poisson tronquée en zéro)**.**
Si $\mathcal{L}_{X\wr Y}$ suit une loi de Poisson tronquée en zéro de paramètre λ, alors

$$P_{S_f}(\lambda) = -l(X)\frac{(1-e^{-\lambda})^2 e^{-\lambda}}{1-e^{-\lambda}(1+\lambda)}\sum_{k=1}^{\infty}\frac{\lambda^{k-2}}{k!}(k-1-\lambda)(k-1)f(k).$$

En particulier si f est la fonction indicatrice $\mathbb{1}$ alors nous obtenons le pouvoir discriminant de la complexité de Lempel-Ziv étendue :

$$P_{S_{\mathbb{1}}}(\lambda) = -l(X)\frac{(1-e^{-\lambda})^2}{\lambda^2}$$

Démonstration. Soit une loi de Poisson tronquée en zéro de paramètre λ, alors la probabilité que $X = k$ est :

$$P(X=k) = \frac{\lambda^k}{(e^\lambda - 1)k!}.$$

Sa moyenne est :

$$m(\lambda) = \frac{\lambda e^\lambda}{e^\lambda - 1}.$$

On a

$$\frac{d\, S_f(X\wr Y)}{d\,\lambda} = \frac{d\, S_f(X\wr Y)}{d\, m(\lambda)}\,\frac{d\, m(\lambda)}{d\,\lambda}.$$

Donc

$$P_{S_f}(m) = \frac{1}{\frac{d\, m(\lambda)}{d\,\lambda}}\,\frac{d\, S_f(X\wr Y)}{d\,\lambda}.$$

On a $\frac{d\, m(\lambda)}{d\,\lambda} = \frac{1-e^{-\lambda}(1+\lambda)}{(1-e^{-\lambda})^2}$.

On pose $L = l(X)$ la longueur de X et n_k le nombre moyen de symbols de longueur k : $n_k = P(X=k)\frac{L}{m(\lambda)}$. Alors, S s'écrit en fonction de λ comme suit :

$$\begin{aligned}
S_f(X\wr Y)_{|\lambda} &= L - \sum_{k=1}^{L} n_k(k-1)f(k) \\
&= L\left(1 - \sum_{k=1}^{L}\frac{P(X=k)}{m(\lambda)}(k-1)f(k)\right) \\
&= L\left(1 - \sum_{k=1}^{L} e^{-\lambda}\frac{\lambda^{k-1}}{k!}(k-1)f(k)\right) \\
&= L\left(1 - \sum_{k=1}^{L} h_k(\lambda)\frac{(k-1)f(k)}{k!}\right),
\end{aligned}$$

où $h_k(\lambda) = e^{-\lambda}\lambda^{k-1}$.

Donc la dérivé de S en fonction de λ est donnée par :

$$\begin{aligned}\frac{d\, S_f(X \wr Y)_{|\lambda}}{d\, \lambda} &= -L\sum_{k=1}^{L}\frac{d\, h_k(\lambda)}{d\, \lambda}\frac{(k-1)f(k)}{k!} \\ &= -L\sum_{k=1}^{L}e^{-\lambda}\lambda^{k-2}(k-1-\lambda)\frac{(k-1)f(k)}{k!}.\end{aligned}$$

Ce qui permet d'obtenir le pouvoir discriminant pour une distribution de Poisson tronquée en zéro comme étant :

$$\begin{aligned}P_{S_f}(m) &= \frac{1}{\frac{d\, m(\lambda)}{d\, \lambda}}\,\frac{d\, S_f(X \wr Y)}{d\, \lambda} \\ &= -\frac{(1-e^{-\lambda})^2e^{-\lambda}}{1-e^{-\lambda}(1+\lambda)}L\sum_{k=1}^{\infty}\frac{\lambda^{k-2}}{k!}(k-1-\lambda)(k-1)f(k).\end{aligned}$$

Pour le cas particulier de $f = \mathbb{1}$, $S_{\mathbb{1}}$ est le nombre de symboles dans la décomposition. Ce nombre de symboles est simplement $S_{\mathbb{1}} = \frac{L}{m(\lambda)}$. Le pouvoir discriminant pour $f = \mathbb{1}$ est alors directement donné par $P_{S_{\mathbb{1}}}(m) = -\frac{1}{m(\lambda)^2} = -\frac{(1-e^{-\lambda})^2L}{\lambda^2}$.

□

Annexe B

Tableaux de valeurs

Dans cette annexe nous présentons plus en détails les valeurs des matrices présentées sous forme d'image dans la chapitre I et II. Les calculs sont dans tous les cas réalisés sur des chaînes i.i.d. d'une loi uniforme discrète avec différentes tailles d'alphabet, des chaînes de Markov avec différentes taille d'alphabet, de l'ADN et la déclaration des droits de l'homme dans différentes langues. Les détails sur les données sont consultables dans l'annexe D. Pour l'analyse des résultats, il faut se reporter à l'analyse des figures associées à chaque tableau.

• **Asymétrie des compresseurs**

Pour deux chaînes de ce jeu de données, nous allons observer l'asymétrie d'un compresseur normal. Pour deux chaînes quelconques X et Y de notre jeu de données, les mesures que nous effectuons sont :

Asymétrie : $C(XY)-C(YX)$. Cette quantité, qui doit être le plus proche de zéro possible, permet d'évaluer la différence entre compresser d'abord X puis Y ou Y puis X.

Normalisation : $\frac{C(XY)-C(YX)}{C(X)+C(Y)}$. Afin de pouvoir comparer la complexité de chaînes de taille différente, il faut normaliser en utilisant la borne supérieure $C(XY) \leq C(X) + C(Y)$.

Les résultats sont synthétisés sur la figure I.4 et commentés dans la section I.3.2. La variance, la moyenne et la maximum pour chaque jeu de données sont présentés dans les tableaux B.1, B.2 et B.3.

• **Influence de l'ajout d'information**

Pour chacune des chaînes de ces jeux de données nous regardons l'influence de l'ajout de la connaissance des autres chaînes du jeu de données. Voici les calculs que nous effectuons pour chaque chaîne X et Y du jeu de données :

Complexité simple : $S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda) = S_{\mathbb{E}}(X)$.

Complexité conditionnelle : $S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda, Y) = S_{\mathbb{E}}(X_{\text{-}}|^{+}Y)$.

Ajout de connaissance : $S_{\mathbb{E}}(X) - S_{\mathbb{E}}(X_{\text{-}}|^{+}Y)$. Cette différence, qui doit être positive, permet d'évaluer ce qu'apporte la connaissance de Y à la connaissance du passé de X.

Normalisation : $\frac{S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda)-S_{\mathbb{E}}(X_{\text{-}}|^{+}Y)}{S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda)}$, pour pouvoir comparer la complexité de chaînes de taille différente, il faut normaliser l'ajout de connaissance par la borne supérieure.

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	1,13e-03	1,10e-04	8,01e-05	1,62e-02	2,27e-02	7,85e-05	2,43e-05	2,62e-05
	1,01e-06	1,19e-08	7,42e-09	8,92e-06	2,88e-06	8,40e-09	1,90e-09	1,93e-09
	8,27e-03	6,51e-04	5,69e-04	2,56e-02	2,79e-02	4,30e-04	3,36e-04	3,50e-04
ADN	1,10e-04	2,34e-03	9,31e-05	2,54e-02	3,37e-02	9,88e-05	4,56e-05	4,24e-05
	1,19e-08	3,45e-06	7,41e-09	1,67e-06	2,86e-07	8,76e-09	3,50e-09	2,25e-09
	6,51e-04	8,75e-03	4,44e-04	2,88e-02	3,50e-02	4,87e-04	3,96e-04	2,66e-04
markov4	8,01e-05	9,31e-05	3,63e-03	1,67e-02	2,69e-02	1,05e-04	6,19e-05	6,29e-05
	7,42e-09	7,41e-09	8,08e-06	1,80e-05	1,15e-05	1,04e-08	3,71e-09	4,61e-09
	5,69e-04	4,44e-04	1,37e-02	2,29e-02	3,13e-02	5,54e-04	3,45e-04	3,47e-04
markov64	1,62e-02	2,54e-02	1,67e-02	1,24e-04	1,60e-02	8,00e-05	3,41e-02	1,65e-02
	8,92e-06	1,67e-06	1,80e-05	1,05e-08	4,04e-07	3,48e-08	5,52e-07	5,76e-07
	2,56e-02	2,88e-02	2,29e-02	5,70e-04	1,81e-02	2,73e-03	3,64e-02	1,90e-02
markov256	2,27e-02	3,37e-02	2,69e-02	1,60e-02	0,00e+00	1,64e-02	4,04e-02	0,00e+00
	2,88e-06	2,86e-07	1,15e-05	4,04e-07	0,00e+00	2,00e-07	2,32e-07	0,00e+00
	2,79e-02	3,50e-02	3,13e-02	1,81e-02	0,00e+00	1,77e-02	4,19e-02	0,00e+00
aléa4	7,85e-05	9,88e-05	1,05e-04	8,00e-05	1,64e-02	2,13e-03	7,09e-05	4,94e-05
	8,40e-09	8,76e-09	1,04e-08	3,48e-08	2,00e-07	2,58e-06	5,33e-09	3,62e-09
	4,30e-04	4,87e-04	5,54e-04	2,73e-03	1,77e-02	9,81e-03	5,56e-04	3,78e-04
aléa64	2,43e-05	4,56e-05	6,19e-05	3,41e-02	4,04e-02	7,09e-05	1,88e-04	8,93e-03
	1,90e-09	3,50e-09	3,71e-09	5,52e-07	2,32e-07	5,33e-09	2,08e-08	2,28e-07
	3,36e-04	3,96e-04	3,45e-04	3,64e-02	4,19e-02	5,56e-04	9,87e-04	1,04e-02
aléa256	2,62e-05	4,24e-05	6,29e-05	1,65e-02	0,00e+00	4,94e-05	8,93e-03	0,00e+00
	1,93e-09	2,25e-09	4,61e-09	5,76e-07	0,00e+00	3,62e-09	2,28e-07	0,00e+00
	3,50e-04	2,66e-04	3,47e-04	1,90e-02	0,00e+00	3,78e-04	1,04e-02	0,00e+00

TABLE B.1 – Symétrie du compresseur `GZIP`, pour chaque catégorie nous affichons la moyenne, la variance et le maximum d'asymétrie.

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	1,07e-03	6,05e-03	6,14e-03	7,20e-04	1,34e-03	8,27e-03	4,05e-04	1,69e-03
	7,88e-07	4,44e-06	4,91e-06	3,21e-07	1,61e-07	4,68e-06	1,13e-07	2,78e-07
	8,11e-03	1,36e-02	1,31e-02	3,07e-03	2,59e-03	1,52e-02	1,98e-03	3,30e-03
ADN	6,05e-03	6,77e-03	1,95e-03	7,41e-04	3,42e-04	1,72e-03	3,52e-04	4,07e-04
	4,44e-06	2,94e-05	2,31e-06	3,31e-07	7,46e-08	1,71e-06	9,89e-08	1,05e-07
	1,36e-02	2,89e-02	9,28e-03	2,68e-03	1,81e-03	6,67e-03	2,23e-03	1,90e-03
markov4	6,14e-03	1,95e-03	1,66e-03	7,37e-04	2,62e-04	2,17e-03	3,93e-04	3,36e-04
	4,91e-06	2,31e-06	1,85e-06	3,59e-07	5,28e-08	3,31e-06	1,30e-07	8,85e-08
	1,31e-02	9,28e-03	8,84e-03	3,19e-03	1,27e-03	9,72e-03	2,30e-03	2,01e-03
markov64	7,20e-04	7,41e-04	7,37e-04	7,48e-04	1,61e-02	1,08e-03	1,62e-03	1,72e-02
	3,21e-07	3,31e-07	3,59e-07	3,12e-07	2,48e-07	1,39e-06	5,58e-07	2,83e-07
	3,07e-03	2,68e-03	3,19e-03	2,57e-03	1,73e-02	1,62e-02	4,19e-03	1,87e-02
markov256	1,34e-03	3,42e-04	2,62e-04	1,61e-02	0,00e+00	2,64e-04	5,54e-04	0,00e+00
	1,61e-07	7,46e-08	5,28e-08	2,48e-07	0,00e+00	7,08e-08	4,98e-08	0,00e+00
	2,59e-03	1,81e-03	1,27e-03	1,73e-02	0,00e+00	3,96e-03	1,40e-03	0,00e+00
aléa4	8,27e-03	1,72e-03	2,17e-03	1,08e-03	2,64e-04	1,98e-03	1,23e-03	3,12e-04
	4,68e-06	1,71e-06	3,31e-06	1,39e-06	7,08e-08	2,38e-06	2,47e-06	9,87e-08
	1,52e-02	6,67e-03	9,72e-03	1,62e-02	3,96e-03	1,03e-02	2,21e-02	5,16e-03
aléa64	4,05e-04	3,52e-04	3,93e-04	1,62e-03	5,54e-04	1,23e-03	1,38e-04	6,68e-04
	1,13e-07	9,89e-08	1,30e-07	5,58e-07	4,98e-08	2,47e-06	2,19e-08	4,65e-08
	1,98e-03	2,23e-03	2,30e-03	4,19e-03	1,40e-03	2,21e-02	7,73e-04	1,57e-03
aléa256	1,69e-03	4,07e-04	3,36e-04	1,72e-02	0,00e+00	3,12e-04	6,68e-04	0,00e+00
	2,78e-07	1,05e-07	8,85e-08	2,83e-07	0,00e+00	9,87e-08	4,65e-08	0,00e+00
	3,30e-03	1,90e-03	2,01e-03	1,87e-02	0,00e+00	5,16e-03	1,57e-03	0,00e+00

TABLE B.2 – Symétrie du compresseur `LZMA`, pour chaque catégorie nous affichons la moyenne, la variance et le maximum d'asymétrie.

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	8,05e-04	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	5,49e-08	9,72e-06
	6,77e-07	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	8,21e-12	1,45e-08
	7,42e-03	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	1,79e-04	2,51e-03
ADN	0,00e+00	2,06e-05	0,00e+00	1,69e-06	0,00e+00	0,00e+00	0,00e+00	0,00e+00
	0,00e+00	2,02e-08	0,00e+00	2,09e-09	0,00e+00	0,00e+00	0,00e+00	0,00e+00
	0,00e+00	1,75e-03	0,00e+00	1,56e-03	0,00e+00	0,00e+00	0,00e+00	0,00e+00
markov4	0,00e+00	0,00e+00	1,06e-04	0,00e+00	0,00e+00	0,00e+00	2,45e-07	1,11e-06
	0,00e+00	0,00e+00	2,30e-07	0,00e+00	0,00e+00	0,00e+00	1,17e-10	1,81e-09
	0,00e+00	0,00e+00	5,35e-03	0,00e+00	0,00e+00	0,00e+00	6,20e-04	2,34e-03
markov64	0,00e+00	1,69e-06	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
	0,00e+00	2,09e-09	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
	0,00e+00	1,56e-03	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00
markov256	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	5,18e-07	0,00e+00	0,00e+00
	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	3,39e-10	0,00e+00	0,00e+00
	0,00e+00	0,00e+00	0,00e+00	0,00e+00	0,00e+00	8,23e-04	0,00e+00	0,00e+00
aléa4	0,00e+00	0,00e+00	0,00e+00	0,00e+00	5,18e-07	2,75e-05	2,34e-06	2,79e-07
	0,00e+00	0,00e+00	0,00e+00	0,00e+00	3,39e-10	2,28e-08	2,34e-09	2,56e-10
	0,00e+00	0,00e+00	0,00e+00	0,00e+00	8,23e-04	2,18e-03	2,49e-03	1,21e-03
aléa64	5,49e-08	0,00e+00	2,45e-07	0,00e+00	0,00e+00	2,34e-06	0,00e+00	0,00e+00
	8,21e-12	0,00e+00	1,17e-10	0,00e+00	0,00e+00	2,34e-09	0,00e+00	0,00e+00
	1,79e-04	0,00e+00	6,20e-04	0,00e+00	0,00e+00	2,49e-03	0,00e+00	0,00e+00
aléa256	9,72e-06	0,00e+00	1,11e-06	0,00e+00	0,00e+00	2,79e-07	0,00e+00	0,00e+00
	1,45e-08	0,00e+00	1,81e-09	0,00e+00	0,00e+00	2,56e-10	0,00e+00	0,00e+00
	2,51e-03	0,00e+00	2,34e-03	0,00e+00	0,00e+00	1,21e-03	0,00e+00	0,00e+00

TABLE B.3 – Symétrie du compresseur `BZIP2`, pour chaque catégorie nous affichons la moyenne, la variance et le maximum d'asymétrie.

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	1,28e-01	9,28e-04	-9,45e-05	-2,35e-04	-2,44e-03	0,00e+00	-2,92e-03	-1,73e-03
	1,88e-02	1,28e-06	7,96e-08	1,67e-06	4,36e-06	0,00e+00	1,50e-06	2,96e-06
	4,13e-02	-1,26e-03	-6,57e-04	-4,12e-03	-9,70e-03	0,00e+00	-7,09e-03	-8,90e-03
ADN	-1,63e-05	4,88e-01	0,00e+00	-3,85e-06	-4,74e-06	0,00e+00	0,00e+00	-2,90e-06
	6,22e-09	9,53e-03	0,00e+00	3,62e-10	4,00e-10	0,00e+00	0,00e+00	2,51e-10
	-6,64e-04	3,83e-01	0,00e+00	-1,28e-04	-2,01e-04	0,00e+00	0,00e+00	-2,06e-04
markov4	-1,44e-05	0,00e+00	2,72e-01	7,11e-06	-5,88e-06	0,00e+00	-6,95e-07	-3,97e-06
	4,85e-09	0,00e+00	2,23e-02	2,74e-09	1,47e-08	0,00e+00	1,49e-08	1,07e-08
	-4,95e-04	0,00e+00	8,00e-02	-1,11e-04	-8,09e-04	0,00e+00	-4,28e-04	-9,70e-04
markov64	3,21e-04	6,06e-05	1,02e-05	8,67e-02	2,69e-03	9,44e-05	1,13e-03	1,95e-03
	8,09e-08	1,89e-08	1,52e-09	2,40e-02	9,29e-07	1,97e-08	2,80e-07	6,41e-07
	-4,14e-04	-1,31e-04	-1,30e-04	4,45e-02	-8,73e-05	-1,14e-04	-8,78e-04	-8,86e-04
markov256	5,90e-03	2,31e-04	2,12e-04	3,60e-02	1,61e-01	1,91e-04	3,77e-02	9,88e-02
	9,78e-07	1,01e-08	1,32e-08	1,61e-06	2,02e-02	7,12e-09	2,13e-06	4,38e-06
	3,62e-03	0,00e+00	0,00e+00	3,14e-02	1,25e-01	0,00e+00	3,37e-02	9,15e-02
aléa4	0,00e+00	0,00e+00	0,00e+00	-8,39e-06	-3,06e-06	3,40e-01	-6,09e-06	-1,32e-06
	0,00e+00	0,00e+00	0,00e+00	5,18e-09	1,31e-09	4,51e-03	8,70e-09	6,56e-10
	0,00e+00	0,00e+00	0,00e+00	-9,64e-04	-5,88e-04	2,94e-01	-1,33e-03	-4,71e-04
aléa64	7,56e-04	0,00e+00	7,05e-05	2,47e-03	6,83e-03	3,90e-04	6,44e-02	4,95e-03
	1,21e-07	0,00e+00	1,22e-08	3,53e-07	8,87e-07	6,64e-08	8,85e-03	6,30e-07
	-9,05e-05	0,00e+00	-1,63e-04	4,54e-04	3,52e-03	-1,08e-04	4,48e-02	2,05e-03
aléa256	6,67e-03	2,56e-04	2,27e-04	3,98e-02	1,47e-01	2,27e-04	4,16e-02	1,19e-01
	1,43e-06	2,35e-08	1,63e-08	3,08e-06	8,37e-06	2,12e-08	3,42e-06	7,85e-03
	3,30e-03	0,00e+00	0,00e+00	3,48e-02	1,35e-01	0,00e+00	3,45e-02	9,87e-02

TABLE B.4 – Influence de l'ajout d'information, pour chaque catégorie la moyenne, la variance et le minimum de la différence $\frac{S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda)-S_{\mathbb{E}}(X_{\text{-}}|^{+}Y)}{S_{\mathbb{E}}(X_{\text{-}}|^{+}\Lambda)}$.

Les résultats sont synthétisés sur la figure II.5 et commentés dans la section II.2.2. La variance, la moyenne et le minimum pour chaque jeu de données sont présentés dans le tableau B.4.

• **Asymétrie de `SALZA` jointe basée sur l'encodage concaténé**

Pour chacune des chaînes des jeux de données, nous observons l'asymétrie de `SALZA` jointe concaténée selon les différentes catégories du jeu de données. Pour cela, nous effectuons pour chaque chaîne X et Y du jeu de données les calculs suivants :

Asymétrie : $S_{\mathbb{E}}(X,Y)^{|u} - S_{\mathbb{E}}(Y,X)^{|u}$, cette différence permet d'évaluer l'asymétrie de `SALZA` jointe concaténée pour X et Y.

Normalisation : $\frac{S_{\mathbb{E}}(X,Y)^{|u}-S_{\mathbb{E}}(Y,X)^{|u}}{S_{\mathbb{E}}(X)+S_{\mathbb{E}}(Y)}$, pour pouvoir comparer la complexité de chaînes de tailles différentes, il faut normaliser l'asymétrie. Pour cela nous divisons par la borne supérieure de la complexité jointe.

Les résultats sont synthétisés dans la figure II.6a et sont commentés dans la section II.3.1. La variance, la moyenne et le maximum pour chaque catégorie sont présentés dans le tableau B.5.

• **Asymétrie de `SALZA` jointe basée sur l'encodage simultané**

Pour réaliser le tableau B.6, nous réalisons exactement le même procédé que pour le

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	3,64e-03	5,64e-04	1,68e-04	3,56e-04	3,12e-03	0,00e+00	1,25e-03	3,44e-03
	8,27e-06	1,89e-07	3,24e-08	6,55e-08	3,85e-07	0,00e+00	1,55e-07	5,08e-07
	2,19e-02	2,55e-03	1,21e-03	1,51e-03	5,16e-03	0,00e+00	2,76e-03	6,26e-03
ADN	5,64e-04	5,59e-03	0,00e+00	1,13e-04	1,04e-04	0,00e+00	0,00e+00	1,01e-04
	1,89e-07	1,74e-05	0,00e+00	1,19e-08	4,21e-09	0,00e+00	0,00e+00	7,11e-09
	2,55e-03	2,64e-02	0,00e+00	5,80e-04	3,16e-04	0,00e+00	0,00e+00	5,05e-04
markov4	1,68e-04	0,00e+00	1,09e-02	2,26e-05	1,03e-04	0,00e+00	1,01e-04	1,06e-04
	3,24e-08	0,00e+00	6,89e-05	1,95e-09	6,34e-09	0,00e+00	8,03e-09	7,77e-09
	1,21e-03	0,00e+00	3,91e-02	3,59e-04	4,58e-04	0,00e+00	5,76e-04	4,42e-04
markov64	3,56e-04	1,13e-04	2,26e-05	1,38e-03	7,72e-03	1,34e-04	2,91e-04	8,20e-03
	6,55e-08	1,19e-08	1,95e-09	1,11e-06	5,30e-07	1,26e-08	5,69e-08	6,06e-07
	1,51e-03	5,80e-04	3,59e-04	6,28e-03	1,02e-02	8,03e-04	1,51e-03	1,12e-02
markov256	3,12e-03	1,04e-04	1,03e-04	7,72e-03	7,55e-04	8,08e-05	7,26e-03	2,79e-03
	3,85e-07	4,21e-09	6,34e-09	5,30e-07	3,82e-07	3,63e-09	6,59e-07	9,34e-07
	5,16e-03	3,16e-04	4,58e-04	1,02e-02	3,03e-03	3,34e-04	1,02e-02	6,43e-03
aléa4	0,00e+00	0,00e+00	0,00e+00	1,34e-04	8,08e-05	6,76e-03	2,86e-04	9,60e-05
	0,00e+00	0,00e+00	0,00e+00	1,26e-08	3,63e-09	2,70e-05	4,62e-08	7,79e-09
	0,00e+00	0,00e+00	0,00e+00	8,03e-04	3,34e-04	3,08e-02	1,50e-03	5,42e-04
aléa64	1,25e-03	0,00e+00	1,01e-04	2,91e-04	7,26e-03	2,86e-04	1,25e-03	8,13e-03
	1,55e-07	0,00e+00	8,03e-09	5,69e-08	6,59e-07	4,62e-08	8,64e-07	7,59e-07
	2,76e-03	0,00e+00	5,76e-04	1,51e-03	1,02e-02	1,50e-03	5,68e-03	1,22e-02
aléa256	3,44e-03	1,01e-04	1,06e-04	8,20e-03	2,79e-03	9,60e-05	8,13e-03	7,86e-04
	5,08e-07	7,11e-09	7,77e-09	6,06e-07	9,34e-07	7,79e-09	7,59e-07	3,62e-07
	6,26e-03	5,05e-04	4,42e-04	1,12e-02	6,43e-03	5,42e-04	1,22e-02	3,49e-03

TABLE B.5 – Disymétrie de `SALZA` jointe basée sur l'encodage concaténé, pour chaque catégorie la moyenne, la variance et le minimum.

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	4,24e-05	8,06e-07	1,54e-08	1,62e-07	4,33e-07	0,00e+00	1,72e-07	3,30e-07
	1,37e-08	1,47e-10	2,08e-13	1,58e-11	2,74e-11	0,00e+00	1,53e-11	2,79e-11
	8,00e-04	3,37e-04	1,36e-05	1,12e-04	6,45e-05	0,00e+00	1,50e-04	8,66e-05
ADN	8,06e-07	2,62e-04	0,00e+00	0,00e+00	1,03e-07	0,00e+00	0,00e+00	0,00e+00
	1,47e-10	2,98e-07	0,00e+00	0,00e+00	6,49e-12	0,00e+00	0,00e+00	0,00e+00
	3,37e-04	9,03e-03	0,00e+00	0,00e+00	6,32e-05	0,00e+00	0,00e+00	0,00e+00
markov4	1,54e-08	0,00e+00	4,03e-05	0,00e+00	0,00e+00	0,00e+00	6,95e-08	0,00e+00
	2,08e-13	0,00e+00	1,34e-08	0,00e+00	0,00e+00	0,00e+00	9,48e-12	0,00e+00
	1,36e-05	0,00e+00	1,01e-03	0,00e+00	0,00e+00	0,00e+00	1,63e-04	0,00e+00
markov64	1,62e-07	0,00e+00	0,00e+00	5,18e-06	9,23e-07	2,11e-08	1,29e-07	1,28e-06
	1,58e-11	0,00e+00	0,00e+00	4,35e-10	4,16e-11	1,14e-12	3,99e-12	7,34e-11
	1,12e-04	0,00e+00	0,00e+00	1,38e-04	4,61e-05	6,38e-05	8,17e-05	1,12e-04
markov256	4,33e-07	1,03e-07	0,00e+00	9,23e-07	1,92e-06	1,85e-08	1,39e-06	1,83e-06
	2,74e-11	6,49e-12	0,00e+00	4,16e-11	6,88e-11	1,24e-12	7,24e-11	7,78e-11
	6,45e-05	6,32e-05	0,00e+00	4,61e-05	3,79e-05	6,67e-05	1,03e-04	4,45e-05
aléa4	0,00e+00	0,00e+00	0,00e+00	2,11e-08	1,85e-08	7,19e-05	2,14e-07	9,00e-09
	0,00e+00	0,00e+00	0,00e+00	1,14e-12	1,24e-12	3,36e-08	2,95e-11	8,09e-13
	0,00e+00	0,00e+00	0,00e+00	6,38e-05	6,67e-05	1,75e-03	1,67e-04	9,00e-05
aléa64	1,72e-07	0,00e+00	6,95e-08	1,29e-07	1,39e-06	2,14e-07	7,09e-06	1,39e-06
	1,53e-11	0,00e+00	9,48e-12	3,99e-12	7,24e-11	2,95e-11	6,45e-10	9,11e-11
	1,50e-04	0,00e+00	1,63e-04	8,17e-05	1,03e-04	1,67e-04	2,79e-04	1,28e-04
aléa256	3,30e-07	0,00e+00	0,00e+00	1,28e-06	1,83e-06	9,00e-09	1,39e-06	2,10e-06
	2,79e-11	0,00e+00	0,00e+00	7,34e-11	7,78e-11	8,09e-13	9,11e-11	1,09e-10
	8,66e-05	0,00e+00	0,00e+00	1,12e-04	4,45e-05	9,00e-05	1,28e-04	1,07e-04

TABLE B.6 – Disymétrie de `SALZA` jointe basée sur l'encodage simultané, pour chaque catégorie la moyenne, la variance et le minimum.

tableau B.5 sauf que nous utilisons `SALZA` jointe basée sur l'encodage simultané. La figure associée est la figure II.6b.

• **Différence entre les différents encodages pour `SALZA` jointe**

Pour chacune des chaînes des jeux de données nous calculons `SALZA` jointe basée sur un encodage k et `SALZA` jointe basée sur un autre encodage l et nous observons la différence. Pour cela nous nous basons sur la méthode décrite ci dessous.

Comparaison : $S_{\mathbb{E}}(X,Y)^{|k} - S_{\mathbb{E}}(X,Y)^{|l}$, cette différence permet évaluer la différence de `SALZA` jointe entre l'encodage k et l'encodage l.

Normalisation : $\frac{S_{\mathbb{E}}(X,Y)^{|k} - S_{\mathbb{E}}(X,Y)^{|l}}{S_{\mathbb{E}}(X)+S_{\mathbb{E}}(Y)}$, pour pouvoir comparer la complexité de chaînes de taille différente, il faut normaliser la comparaison. Pour cela nous divisons par la borne supérieur de la complexité jointe.

Les résultats obtenus sont synthétisés dans la figure II.7 et sont commentés dans la section II.3.1. La variance, la moyenne et le maximum pour chaque jeu de données sont présentés dans les tableaux B.7, B.8 et B.9.

	DDHC	ADN	markov4	markov64	markov256	aléa4	aléa64	aléa256
DDHC	6,08e-03	3,59e-05	1,67e-06	2,40e-04	2,30e-03	0,00e+00	4,24e-04	2,29e-03
	1,11e-05	1,24e-08	5,80e-10	3,67e-08	2,49e-07	0,00e+00	5,85e-08	2,91e-07
	1,91e-02	7,26e-04	4,18e-04	1,26e-03	4,23e-03	0,00e+00	1,35e-03	4,48e-03
ADN	5,53e-04	8,60e-03	0,00e+00	1,09e-04	9,34e-05	0,00e+00	0,00e+00	8,94e-05
	1,87e-07	3,26e-05	0,00e+00	1,15e-08	3,52e-09	0,00e+00	0,00e+00	6,05e-09
	2,55e-03	2,99e-02	0,00e+00	5,80e-04	3,16e-04	0,00e+00	0,00e+00	4,21e-04
markov4	1,67e-04	0,00e+00	7,41e-03	1,99e-05	9,22e-05	0,00e+00	9,30e-05	9,27e-05
	3,20e-08	0,00e+00	3,28e-05	1,62e-09	5,37e-09	0,00e+00	7,05e-09	6,83e-09
	1,21e-03	0,00e+00	2,78e-02	2,39e-04	3,92e-04	0,00e+00	5,76e-04	4,40e-04
markov64	1,95e-04	9,18e-06	3,30e-06	1,27e-03	4,48e-03	7,07e-07	1,77e-04	4,24e-03
	2,15e-08	9,97e-10	3,35e-10	9,72e-07	4,08e-07	6,58e-11	1,83e-08	3,66e-07
	8,81e-04	2,36e-04	1,22e-04	6,45e-03	6,63e-03	1,24e-04	8,53e-04	6,50e-03
markov256	8,20e-04	1,14e-05	1,20e-05	3,24e-03	1,06e-03	1,67e-07	2,79e-03	6,46e-04
	6,60e-08	5,91e-10	6,31e-10	1,84e-07	4,65e-07	1,11e-11	2,13e-07	2,13e-07
	1,85e-03	6,35e-05	6,59e-05	4,50e-03	3,47e-03	6,68e-05	4,66e-03	3,01e-03
aléa4	0,00e+00	0,00e+00	0,00e+00	1,34e-04	8,07e-05	5,95e-03	2,83e-04	9,56e-05
	0,00e+00	0,00e+00	0,00e+00	1,26e-08	3,63e-09	2,07e-05	4,47e-08	7,76e-09
	0,00e+00	0,00e+00	0,00e+00	8,03e-04	3,34e-04	2,99e-02	1,48e-03	5,42e-04
aléa64	8,29e-04	0,00e+00	1,30e-05	3,04e-04	4,47e-03	6,73e-06	1,03e-03	4,52e-03
	1,04e-07	0,00e+00	1,39e-09	5,19e-08	4,99e-07	9,92e-10	6,38e-07	4,97e-07
	2,29e-03	0,00e+00	3,24e-04	1,38e-03	6,71e-03	3,34e-04	5,17e-03	7,71e-03
aléa256	1,15e-03	1,19e-05	1,36e-05	3,96e-03	2,29e-03	3,87e-07	3,62e-03	5,75e-04
	1,34e-07	8,55e-10	9,91e-10	2,88e-07	7,04e-07	3,46e-11	3,28e-07	1,95e-07
	2,70e-03	8,45e-05	8,88e-05	5,99e-03	5,27e-03	9,04e-05	5,82e-03	2,94e-03

TABLE B.7 – Différence entre `SALZA` jointe basée sur l'encodage concaténé ou sur l'encodage simultané, pour chaque catégorie la moyenne, la variance et le minimum.

	ADN	markov4	aléa4
ADN	8,42e-02	9,12e-02	6,97e-02
	5,73e-04	2,43e-03	5,59e-05
	2,44e-01	1,72e-01	9,35e-02
markov4	9,12e-02	1,11e-01	5,24e-02
	2,43e-03	1,19e-02	8,71e-04
	1,72e-01	6,00e-01	1,02e-01
aléa4	6,97e-02	5,24e-02	9,43e-02
	5,59e-05	8,71e-04	1,31e-04
	9,35e-02	1,02e-01	1,17e-01

TABLE B.8 – Différence entre `SALZA` jointe basée sur l'encodage concaténé ou sur l'encodage cardinal, pour chaque catégorie la moyenne, la variance et le minimum.

	ADN	markov4	aléa4
ADN	9,21e-02	9,12e-02	6,97e-02
	6,65e-04	2,43e-03	5,59e-05
	2,50e-01	1,72e-01	9,35e-02
markov4	9,12e-02	1,09e-01	5,24e-02
	2,43e-03	1,19e-02	8,71e-04
	1,72e-01	5,94e-01	1,02e-01
aléa4	6,97e-02	5,24e-02	9,48e-02
	5,59e-05	8,71e-04	1,34e-04
	9,35e-02	1,02e-01	1,18e-01

TABLE B.9 – Différence entre `SALZA` jointe basée sur l'encodage cardinal ou sur l'encodage simultané, pour chaque catégorie la moyenne, la variance et le minimum.

Annexe C

Démonstrations du chapitre III

Dans cette annexe, nous présentons les preuves sur les différentes métriques présentées dans le chapitre III.

Propriété III.1.1 (Distance)**.**
Soit X, Y et Z trois chaînes définies sur $\mathcal{A}^*$, alors la métrique $D : \mathcal{A}^* \times \mathcal{A}^* \to \mathbb{R}^+$ est une distance si et seulement si elle respecte les propriétés suivantes :

1. *Positivité : $D(X,Y) \geq 0$,*
2. *Symétrie : $D(X,Y) = D(Y,X)$,*
3. *Égalité des indiscernables : $D(X,Y) = 0$ si et seulement si $X = Y$,*
4. *Inégalité triangulaire : $D(X,Z) \leq D(X,Y) + D(Y,Z)$.*

• La divergence de Ziv-Merhav est une divergence.

Démonstration. Nous présentons ici des contre-exemples simples permettant de vérifier que les différentes propriétés ne sont pas vérifiées.

1. La divergence de Ziv-Merhav n'est pas positive. Prenons l'exemple où $X = Y$ deux chaînes aléatoires de taille $n = 10000$ sur un alphabet binaire $\alpha = 2$. On a alors $L(X) = L(Y) \approx \frac{n}{log_{\alpha^2} n}$ (voir section I.2.1.3). De plus $L_Z(X|Y) = 1$ car on ne fait qu'une opération `copier` depuis Y pour expliquer X. En réalisant l'application numérique, nous trouvons : $\Delta(X||Y) = -0.84$.
2. La divergence de Ziv-Merhav n'est pas symétrique. Prenons $X = 1\ 2\ ...$ la chaîne des 256 premiers chiffre et $Y = (0\ 1\ 2\ 3)^{64}$ la chaîne où l'on répète '0 1 2 3' 64 fois. On a alors $L(X) = 256$ et $L(Y) = 5$; $L(X|Y) = 1 + 252 = 253$ et $L(Y|X) = 64$. En réalisant l'application numérique on trouve : $\Delta(X||Y) = -0.0156 \neq 0.244 = \Delta(Y||X)$
3. La divergence de Ziv-Merhav ne respecte pas l'égalité des indisernables. Pour cela il suffit de reprendre l'exemple de la non positivité.
4. La divergence de Ziv-Merhav ne respecte pas l'inégalité triangulaire. Le contre exemple de la *NSD* qui se trouve à la page suivante est aussi valable pour la divergence de Ziv-Merhav.

□

• La *NCD* est une pseudo-quasi-semi-distance.

Démonstration. La *NCD* ne respecte que la propriété de positivité d'une distance.

1. La *NCD* est positive. Si nous rallongeons la chaîne X alors forcément la taille du fichier compressé sera plus grande. Cependant on ne pas compresser un fichier vide avec les compresseurs.
2. La *NCD* n'est pas symétrique. Voir section I.3.2.
3. La *NCD* ne respecte pas l'égalité des indisernables. Ceci est du faite que $C(XX) \neq C(X)$
4. La *NCD* ne respecte pas l'inégalité triangulaire. Le contre exemple de la *NSD* qui se trouve dans la démonstration suivante est aussi valable pour la *NCD*.

□

• **La *NSD* est une semi-distance.**

Démonstration. La *NSD* respecte toutes les propriétés d'une distance hormis l'inégalité triangulaire.

1. La *NSD* est positive par définition car $\forall X, Y \in \mathcal{A}\ S_f(X \wr Y) \geq 0$.
2. La *NSD* est symétrique par définition.
3. La *NSD* respecte l'égalité des indisernables.
   - $\Rightarrow$: Si $X = Y$ alors $S_f(X|^+Y) = S_f(Y|^+X) = 0$ car on ne réalise qu'une référence pour expliquer X avec Y et inversement. Donc $X = Y \Rightarrow NSD(X, Y) = 0$
   - $\Leftarrow$: si $NSD(X, Y) = 0$ alors cela singifie que $S_f(X|^+Y) = S_f(Y|^+X) = 0$. Si $S_f(X|^+Y) = 0$ alors cela signifie qe l'on a pu expliquer X avec une seule référence dans Y. De même si $S_f(Y|^+X) = 0$ alors cela signifie que l'on a pu expliquer Y avec une seule référence dans X. Donc X=Y.
4. La *NSD* ne respecte pas l'inégalité triangulaire. Prenons pour cela un contre exemple. Pour rappel $(X)^n$ signifie que nous répétons n fois la chaîne X. Pour inverser les caractères de la chaîne X, nous notons $\overline{X} = x_n x_{n-1} ... x_2 x_1$. Soit trois alphabet $\mathcal{A}_A$, $\mathcal{A}_B$ et $\mathcal{A}_C$ de taille α distincts, i.e. $\mathcal{A}_A \cap \mathcal{A}_B = \mathcal{A}_A \cap \mathcal{A}_C = \mathcal{A}_B \cap \mathcal{A}_C = \emptyset$. Prenons trois chaînes A, B et C constituées des α éléments de $\mathcal{A}_A$, respectivement $\mathcal{A}_B$ et $\mathcal{A}_C$. On pose alors
   - $X = (A)^n B\overline{C}$
   - $Y = B(C)^n\overline{A}$
   - $Z = AC(B)^n$

   On a alors pour la fonction admissible indicatrice :
   - $S_{\mathbb{1}}(X|^+Y) = S_{\mathbb{1}}(Y|^+X) = (n+1)\alpha$
   - $S_{\mathbb{1}}(X|^+Z) = S_{\mathbb{1}}(Y|^+Z) = S_{\mathbb{1}}(Z|^+Y) = S_{\mathbb{1}}(Z|^+X) = n + \alpha$

   Ce qui donne :
   - $NSD_{\mathbb{1}}(X, Y) = \frac{n+1}{n+2}$
   - $NSD_{\mathbb{1}}(X, Z) = \frac{n+\alpha}{(n+2)\alpha}$

- $NSD_{\mathbb{1}}(Z,Y) = \frac{n+\alpha}{(n+2)\alpha}$

Pour les valeurs $n = 1000$ et $\alpha = 10$, le calcul nous donne

$$NSD_{\mathbb{1}}(X,Z) + NSD_{\mathbb{1}}(Z,Y) - NSD_{\mathbb{1}}(X,Y) = -0,78 < 0.$$

L'inégalité triangulaire n'est donc pas respectée.

□

Annexe D

Données utilisées

Dans cette annexe nous présentons toutes les données que nous avons utilisées au cours de cette thèse.

D.1 Chaînes i.i.d.

Ces chaînes sont des chaînes idd sur un alphabet de taille $\alpha = 4$, 64 ou 256. Pour cela nous tirons 10000 valeurs selon une loi uniforme discrète $\mathcal{U}_{[0,\alpha-1]}$.

Chaînes idd

Caractéristique : chaîne aléatoire i.i.d. selon une loi uniforme discrète.

Longueur $= 10kO$.

Alphabet : $\alpha = 4$,64 ou 256.

Nom : aléa4, aléa64 ou aléa256.

D.2 Chaînes de Markov

Ces chaînes sont des chaînes de Markov sur un alphabet de taille 4 , 64 ou 256. Pour chacune des tailles d'alphabet, nous générons aléatoirement six matrices de probabilités de transition différentes. Pour chacune de ces matrices, nous générons six chaînes de taille 15000 avec six initialisations différentes. Le nom d'un chaîne de Markov est de la forme *alpha*`i`*_mj_c*`k`, où `i` est la taille de l'alphabet, `j` est le numéro de la matrice de transition qui a servi à générer la chaîne et `k` est le numéro de la réalisation.

Chaînes de Markov

Caractéristique : chaîne de Markov avec différentes matrice de transition.

Longueur $= 15kO$.

Alphabet : $\alpha = 4$,64 ou 256

Nom : markov4, markov64 et markov256.

D.3 ADN

Ces chaînes de caractères sont des ADN de mammifères provenants de la base de données de Complean ([Cil+16]) qui est le site dédié à la *NCD*. Ce jeu de données a été introduit dans [Li+04] et possède trente-quatre fichiers textes différents. Chaque fichier texte est une suite des quatre caractères $\{A, C, G, T\}$ qui représente un ADN et décrit le patrimoine génétique d'une espèce. En moyenne, l'ADN a une taille de 16000 octets. Le nom du fichier ADN correspond à celui du mammifère donc il est extrait.

ADN

Caractéristique : description génétique de différentes espèces de mammifères.

Longueur de $16kO$ à $17k0$.

Alphabet : α= 4.

Nom : ADN.

D.4 Déclaration Universelle des Droits de l'Homme

Nous avons extrait du site de l'INU [Onu] la Déclaration des Droits Universelle de l'Homme écrite dans cinquante langues indo-européennes différentes. Ce jeu de données a été introduit dans [Li+04]. Les langues indo-européennes sont écrites sur l'alphabet romain de 26 lettres plus des accents et les signes de ponctuations, ce qui fait que l'alphabet est en moyenne de taille 64. Le nom de la version de la déclaration des droits de l'homme est la langue dans laquelle elle est rédigée

Déclaration universelle des droits de l'homme

Caractéristique : déclaration universelle des droits de l'homme écrite dans différentes langues indo-européennes.

Longueur de $10kO$ à$13kO$.

Alphabet : α est environ 64.

Nom : DDHC.

D.5 Littérature française

Nous avons récupérer différents romans français libre de droit. Ces romans sont écrits sur l'alphabet romain de 26 lettres plus des accents et les signes de ponctuations ce qui fait que l'alphabet est en moyenne de taille 64. Leur taille varie énormément allant de 12 kO à 2 MO. Pour voir à quoi correspond les noms dans les dendogrammes voir la table D.1.

Fichier	Titre	Date	Auteur	Nom
aulnoy_contes_1.txt	Les Contes des Fées	1698	Aulnoy	Aulnoy1(1698)
aulnoy_contes_2.txt	Les Contes nouveaux	1698	Aulnoy	Aulnoy2(1698)
descartes_discours_methode.txt	Discours de la méthode	1637	Descartes	Descartes(1637)
descartes_les_passions_de_l_ame.txt	Les Passions de l'âme	1649	Descartes	Descartes(1649)
dumas_les_trois_mousquetaires.txt	Les trois Mousquetaires	1844	Dumas	Dumas(1844)
dumas_vingt_ans_apres	Vingt ans après	1845	Dumas	Dumas(1845)
maupassant_bel_ami.txt	Bel-Ami	1885	Maupassant	Maupassant(1885)
maupassant_fort_comme_la_mort.txt	Fort comme la mort	1889	Maupassant	Maupassant(1889)
maupassant_mont_oriol.txt	Mont Oriol	1887	Maupassant	Maupassant(1887)
maupassant_une_vie.txt	Une vie	1883	Maupassant	Mauppassant(1883)
montaigne_essais-I.txt	Essais I	1572	Montaigne	MontaigneI(1572)
montaigne_essais-II.txt	Essais II	1572	Montaigne	MontaigneII(1572)
montaigne_essais-III.txt	Essais III	1572	Montaigne	MontaigneIII(1572)
montesquieu_considerations.txt	Considérations	1734	Montesquieu	Montesquieu(1734)
montesquieu_lettres_familieres.txt	Lettres famillières	1767	Montesquieu	Montesquieu(1767)
montesquieu_lettres_persanes.txt	Lettres persannes	1721	Montesquieu	Montesquieu(1721)
montesquieu_esprit_des_lois.txt	Esprit des Lois	1748	Montesquieu	Montesquieu(1748)
pascal_les_provinciales.txt	Les provinciales	1657	Pascal	Pascal(1657)
pascal_pensee.txt	Les pensées	1669	Pascal	Pascal(1669)
perrault_contes.txt	Histoires ou Contes du temps passé	1697	Perrault	Perrault(1697)
rabelais_pantagruel.txt	Pantagruel	1532	Rabelais	Rabelais(1532)
rabelais_gargantua.txt	Gargantua	1534	Rabelais	Rabelais(1534)
rousseau_emile_de_education_1_3.txt	Émile ou De l'éducation (vol1-3)	1762	Rousseau	Rousseau1(1762)
rousseau_emile_de_education_4.txt	Émile ou De l'éducation (vol4)	1762	Rousseau	Rousseau2(1762)
rousseau_les_confessions.txt	Les confessions	1765	Rousseau	Rousseau(1765)
rousseau_reveries_promeneur_solitaire.txt	Les Rêveries du promeneur solitaire	1778	Rousseau	Rousseau(1778)
sade_justine.txt	Justine	1791	Sade	Sade(1791)
sade_les_infortunes_de_la_vertu.txt	Les infortunés de la vertu	1787	Sade	Sade(1787)
stendhal_la_chartreuse_de_parme.txt	La Chartreuse de Parme	1841	Stendhal	Stendhal(1841)
stendhal_le_rouge_et_le_noir.txt	Le Rouge et le Noir	1830	Stendhal	Stendhal(1830)
voltaire_candide.txt	Candide	1759	Voltaire	Voltaire(1759)
voltaire_romans_et_contes.txt	Romans et contes	1747	Voltaire	Voltaire(1747)
zola_assomoir.txt	L'Assomoir	1877	Zola	Zola(1877)
zola_bete_humaine.txt	La Bête humaine	1890	Zola	Zola(1890)
zola_deblacle.txt	La Débacle	1892	Zola	Zola(1892)
zola_germinal.txt	Germinal	1885	Zola	Zola(1885)

TABLE D.1 – Romans français numérisés, de gauche à droite le nom du fichier dans la base de données, le noms de l'oeuvre, sa date de parution, son auteur et le nom donné au roman au cours de cette thèse.

Littérature française

Caractéristique : romans de la littérature française de différents auteurs et à différentes époques.

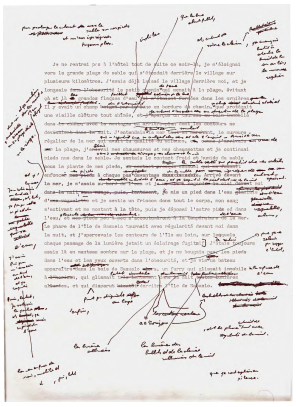
Longueur de $12kO$ à $2MO$.

Alphabet : $\alpha]$ est environ 64.

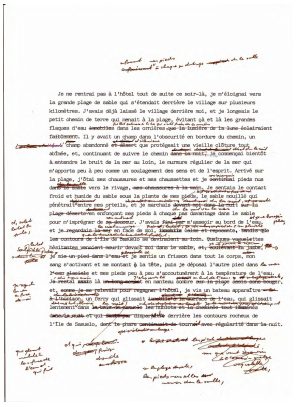
Nom : livres.

D.6 Toussaint

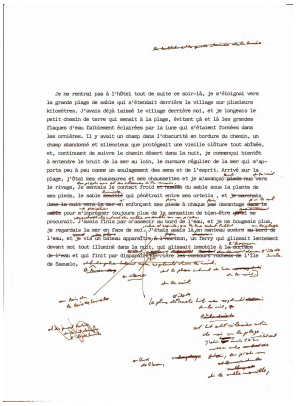
Jean-Philippe Toussaint est un auteur belge connu pour sa technique de travail particulière.Chaque paragraphe publié dans ses livres est le résultat d'itérations successives.



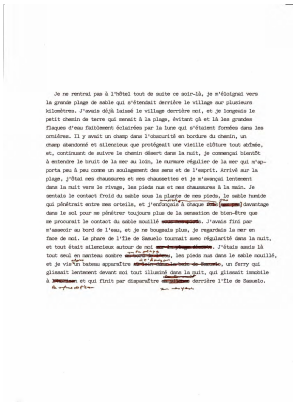
(a) Versions 1 et 2.



(b) Versions 3 et 4.



(c) Versions 5 et 6.



(d) Versions 7 et 8.

FIGURE D.1 – Scan des huit versions successives d'un paragraphe dans *La Réticence* de Toussaint.

D'abord, il écrit le paragraphe à la machine à écrire ou sur ordinateur et l'imprime, ce qui fait une première version. Ensuite, il annote manuellement les changements, ce qui crée une deuxième version. Puis il réécrit la version annoté en introduisant de nouveaux changements ou en supprimant, ce qui crée une troisième version. Ce processus est répété jusqu'à ce que la version publiée soit atteinte. La figure D.1 montre comment les huit versions successives d'un paragraphe de *La Réticence* convergent vers la version publiée : les dernières versions ne diffèrent que par quelques mots.

Toussaint

Caractéristique : huit versions successives d'un paragraphe de *La Réticence* de Jean-Philippe Toussaint.

Longueur de 1600 octets à 2200 octets.

Alphabet : α est environ 64.

Nom : toussaint.

Annexe E

SALZA : notes d'implémentation

E.1 Environnement et contraintes

E.1.1 Environnement d'exécution

Pour l'exécution, nous considérons une machine standard, dotée de plusieurs coeurs physiques d'exécution se partageant l'accès à la mémoire centrale. Nous ne considérons pas les architectures de type GPU. Nous considérons que l'implémentation est la plus rapide lorsque le débit du bus mémoire est saturé, et faisons l'hypothèse que la capacité de la mémoire centrale est de quelques giga-octets à quelques dizaines de giga-octets.

E.1.2 Contraintes de calcul

Contrairement à DEFLATE qui ne produit pas de références de longueur 2 ou supérieure à 258, `SALZA` doit pouvoir produire des références de longueur arbitraire, dans la limite de la taille des pointeurs utilisés. `SALZA` utilise des pointeurs de 32 bits. Étant donnée la capacité typique de la mémoire centrale des machines considérées, il est inutile d'utiliser des pointeurs plus longs, y compris sur les machines 64 bits (en réalité, des pointeurs de 24 bits suffiraient).

E.2 Structure de données pour la recherche

La première phase des calculs consiste à indexer l'intégralité des chaînes de symboles pour permettre une recherche rapide de sous-chaînes de longueur arbitraire (en particulier, y compris de longueur 2). La structure de données pour cette indexation est un tableau de 64K entrées indexant les sous-chaînes par leurs deux premiers octets. Chaque entrée de ce tableau est un pointeur vers un tableau de 256 tableaux dynamiques, permettant la recherche par le troisième octet des sous-chaînes recherchées (le cas échéant). Lorsqu'il existe au moins une sous-chaîne débutant par les trois premiers octets recherchés, son index par rapport au début de la chaîne indexée est stocké dans un tableau dynamique.

Cette structure permet la recherche rapide de sous-chaînes de longueur arbitraire, tout en minimisant la place occupée en mémoire centrale. Soit la recherche d'une sous-chaîne débutant par les octets xy. Si une telle sous-chaîne n'existe pas, alors le tableau de 64K entrées contiendra le pointeur nul à l'index x«8+y et on émettra le litéral x. Sinon, on

lit le troisième octet de la sous-chaîne recherchée (z) et on teste là encore si le tableau de 256 entrées contient le pointeur nul à l'index z. Si ce pointeur est nul, alors on sait qu'on peut trouver une sous-chaîne commençant par xy mais pas par xyz (et on renvoie la première occurence que l'on trouve). Sinon, on parcourt les index contenus dans le tableau dynamique associé, qui concernent tous des sous-chaînes débutant par xyz. Ces index sont rangés par ordre croissant, permettant de spécifier l'intervalle dans lequel chaque recherche de sous-chaîne doit être effectuée. Si l'on le souhaitait, c'est ainsi que l'on fixerait une taille de fenêtre glissante pour un codeur `LZ77` classique.

Il existe un compromis à réaliser entre la taille en mémoire de la structure de données et la vitesse à laquelle celle-ci est construite. Ce compromis est fixé en utilisant une stratégie d'allocation amortie pour les tableaux dynamiques d'index, permettant ainsi de minimiser le nombre d'appels à l'allocateur système (réputés coûteux), tout en ne laissant que peu de mémoire allouée mais inoccupée. Lorsque l'on doit augmenter la taille des tableaux dynamiques d'index, on l'augmente donc d'un facteur 1.25. De fait, la construction de ces structures de données pour l'indexation représente typiquement moins de 10% du temps de calcul pour une matrice de semi-distances. Nous verrons ci-dessous, en outre, comment la paralléliser.

E.3 Stratégies pour la parallélisation

Les étapes des calculs pouvant tirer parti de l'utilisation de plusieurs coeurs physiques d'exécution sont (1) la construction des structures de données d'indexation et surtout (2) les factorisations des chaînes de symboles. Toutefois, ces deux étapes n'ayant aucune raison de présenter le même motif d'accès à la mémoire centrale, on devra utiliser un mécanisme générique de parallélisation permettant de spécifier avec souplesse les paramètres des différentes parties des calculs (i.e., des longueurs de blocs différentes pour l'indexation et la factorisation). Tous les paramètres des calculs parallèles sont modifiables en ligne de commande dans notre implantation.

Le mécanisme générique de parallélisation utilisé est une file d'attente (dispatch queue) fonctionnant suivant le modèle producteur/consommateurs (voir le chapitre 6 de [LB96]), modifiée pour permettre d'utiliser un nombre arbitraire de coeurs physiques d'exécution parmi ceux disponibles à l'aide du paramètre –cpus passé en ligne de commande. Ainsi, on créera des tâches découpant les calculs, qui seront mises dans la file d'attente et exécutées en parallèle. En toute généralité, la parallélisation requiert une étape de finalisation du calcul.

E.3.1 Parallélisation de l'indexation

La construction parallèle d'une structure de données d'indexation utilise un découpage par blocs de la chaîne à indexer. Chaque bloc est indexé dans une structure telle que décrite précédemment. Enfin, la finalisation consiste à rassembler ces structures en une seule. Cette finalisation est elle-même parallélisée, chaque coeur étant utilisé pour rassembler une partie des tableaux de 64K entrées. Cette finalisation ne présente aucun problème

de synchronisation puisque les parties à rassembler sont disjointes. L'inconvénient majeur de cette étape de finalisation est le nombre de fautes de cache générées. Mais comme la construction séquentielle de ces structures pour l'indexation est déjà rapide (voir ci-dessus), nous proposons de les paralléliser avec une taille de blocs conséquente par défaut (128KiB), modifiable à l'aide du paramètre –dict-block. Bien évidemment, lorsque plusieurs chaînes doivent être indexées, elles le sont en parallèle également.

E.3.2 Parallélisation de la factorisation

Nos essais ont révélé que paralléliser la recherche de la plus longue sous-chaîne parmi l'ensemble des index valides contenus dans les tableaux dynamiques présente une granularité trop faible. Nous devons donc, là encore, paralléliser en factorisant par blocs.

Toutefois, si cette stratégie somme toute grossière peut être utilisée telle quelle dans un compresseur sans autre forme de procès, il n'en va pas de même pour `SALZA`, dont les calculs utilisent implicitement le résultat séquentiel exact des factorisations mises en jeu. Il convient donc de se focaliser sur les frontières de blocs de symboles LZ. L'étape (ici séquentielle) de finalisation consiste alors à retrouver la factorisation séquentielle exacte à partir des factorisations parallèles par blocs.

Pour que cela soit possible, il faut autoriser le dernier symbole LZ à dépasser la limite droite du bloc factorisé, le cas échéant. On dispose alors de toute l'information nécessaire pour recoudre ensemble les blocs de symboles LZ. Recoudre le bloc i avec le bloc i+1 consiste à supprimer les premiers symboles LZ du bloc i+1 couverts par le dernier symbole LZ du bloc i, puis à itérer jusqu'à synchronisation entre la suppression de symboles LZ dans le bloc i+1 et la recherche supplémentaire de plus longue sous-chaîne. En pratique, cette resynchronisation est rapide, ne nécessitant pas plus de quelques recherches supplémentaires de plus longues sous-chaînes aux frontières de blocs. On garantit ainsi qu'une fois recousus, les blocs ont convergé vers la factorisation séquentielle usuelle exacte. Par défaut, la taille des blocs à factoriser en parallèle est de 4KiB, modifiable en ligne de commande par le paramètre –fact-block. Là encore, lorsque le calcul nécessite plusieurs factorisations, elles sont effectuées en parallèle.

E.4 État du code

`SALZA` est implanté en un peu plus de 9000 lignes de C et n'utilise que la librairie standard, la librairie mathématique et la librairie des threads POSIX. Concernant les accès à la mémoire, valgrind ne détecte pas de fuite ni d'erreur à l'exécution, tout comme helgrind pour la correction de la parallélisation.

Notations algorithmiques

$< \cdot, \cdot >$	Fonction standard bijective et récursive qui associe une paire à un singleton. 15, 28, 67
$\mathbb{1}$	fonction indicatrice. 58, 63, 64, 66, 68, 69, 82, 91, 144, 147, 149, 150, 162, 163
$\Rightarrow$	Symbole de production. 21–23, 30
$\rightarrow$	Symbole de reproduction. 21, 23, 30
$\mathcal{A}$	Alphabet de taille finie α. 4, 5, 8, 9, 17, 18, 21, 24, 27, 28, 30, 31, 37, 50, 59, 66, 68, 69, 71, 89, 91, 147, 162
α	Nombre de caractères dans l'alphabet $\mathcal{A}$. 4, 8, 24, 25, 30, 37, 48, 57, 59, 68, 80, 89, 91, 97–99, 101, 102, 133, 134, 161–163, 165–168
a	Un symbole de $\mathcal{A}$. 8
$\mathcal{A}^n$	Ensemble de toutes les chaînes sur $\mathcal{A}$ de longueur n i.e $\mathcal{A}^n = \{X \in \mathcal{A}^* \vert l = n\}$. 5, 17, 18, 24, 89
$\mathcal{A}^*$	Ensemble de toute les chaînes de taille finie sur $\mathcal{A}$. 4, 5, 13–16, 18–23, 25, 27, 28, 31, 42, 44, 53, 56–59, 61–63, 66, 67, 88–91, 143, 144, 147, 161
C	Taille d'un fichier compressé. v, 42–44, 90, 151, 162
D	Historique de production (ou décomposition). 22, 23
Δ	Divergence de Ziv-Merhav. 89, 92, 161
E	Historique exhaustif. 23, 30
$\mathbb{E}$	Fonction exponentielle. v, 58, 64, 65, 69–72, 96, 107–109, 111, 113, 115, 126, 151, 155, 157
ϕ_0	Fonction récursive additivement optimale. 15
ϕ	Méthode de programmation ou fonction récursive. 13–15
I	Information mutuelle algorithmique. 18, 19, 28, 74–77, 148

K Complexité de Kolmogorov. 14–19, 27, 69, 89

L Complexité de Lempel-Ziv. 22, 24, 26–28, 56, 57, 89, 161

$\mathcal{L}$ Ensemble des longueurs d'une décomposition de `SALZA`. 49, 53–63, 78, 80, 82, 143–147, 149

`L` Symbole de l'opération `insérer` dans la compression `GZIP`. 38–41, 49, 53–55, 60, 62, 63, 69

Λ Chaîne de longueur nulle supposée être dans $\mathcal{A}^*$. 4, 5, 16, 21, 22, 25, 31, 34–37, 63–66, 147, 151, 155

l_0 Longueur seuil par défaut. 59, 80, 81

l_b Longueur du buffer dans `LZ77`. 30, 31, 33, 34, 37, 38, 48

l Longueur de la chaîne X en nombre de symboles. 4, 5, 13–15, 21, 22, 24, 27, 33, 42, 57–59, 61–63, 68, 75, 79, 80, 82, 89–91, 143, 147, 149

l_{max} Longueur maximale d'un copier dans `LZ77`. 30, 31, 33, 37, 48

l_{min} Longueur minimale d'un copier dans `GZIP`. 37, 38, 48

$l^\star$ Longueur seuil pour les fonctions admissibles. 58–62, 64, 69, 80–86, 90, 99, 125, 126, 143

NCD Normalized Compression Distance. 3, 90–92, 96–105, 107–109, 111, 113, 115, 138, 161, 162, 166

ND Non-descendant dans un graphe de causalité. 119, 120, 122

NID Normalized Information Distance. 3, 4, 89, 90, 92, 104

NSD Normalized `SALZA` Distance. 3, 90–92, 96–105, 107–109, 111, 113, 115, 138, 161–163

O Octet. 38, 41

PA Parents dans un graphe de causalité. 119, 120, 122

p Programme exécuté par ϕ. 13–15

`R` Opération retard. 76, 129–132, 148

S `SALZA` estimateur de complexité. v, 61–74, 76, 77, 79, 81, 82, 90, 91, 122, 130, 143, 144, 147–151, 155, 157, 162

$\mathbb{S}$ Fonction sigmoïde. 58, 62, 63

$\mathbb{T}$ Fonction seuil. 58, 60, 64

t Pointeur de reproduction. 21, 30–33

v Vocabulaire de X i.e. toutes les sous-chaînes de X. 20

W *Quantité significative* (well explain quantity). 59–61, 143

X Chaîne de caractères dans $\mathcal{A}^*$. iii, v, vii, 4, 5, 8, 10–31, 33, 34, 37–45, 47–82, 85, 88–91, 117–122, 128–132, 138, 143–151, 155, 157, 161–163

$\mathcal{X}$ Ensemble des chaînes étudiées. 13, 48, 90, 92, 93, 95, 104, 128, 130–132

Y Chaîne de caractères dans $\mathcal{A}^*$. iii, v, vii, 5, 8, 10, 11, 15, 16, 18, 19, 26–29, 42–44, 47–82, 85, 88–91, 117–122, 128–131, 138, 143–151, 155, 157, 161–163

Z Chaîne de caractères dans $\mathcal{A}^*$. vii, 18, 19, 27, 28, 42, 51, 53–55, 60, 62–64, 66, 68, 73, 88, 91, 118–122, 129–131, 144–147, 161–163

`Z` Symbole de l'opération `copier` dans la compression `GZIP`. 38, 40, 41, 48, 49, 52–55, 60, 62–64, 69, 98

Notions algorithmiques

Normalized Compression Distance — Distance basée sur la compression. 3, 90–92, 96–105, 107–109, 111, 113, 115, 138, 161, 162, 166

Normalized Information Distance — Distance basée sur la complexité de Kolmogorov. 3, 4, 89, 90, 92, 104

Normalized `SALZA` Distance — Distance basée sur `SALZA`. 3, 90–92, 96–105, 107–109, 111, 113, 115, 138, 161–163

octet — Symbole sur 8 bits (c'est-à-dire pouvant aller de 0 à 255). 37, 38, 41

pouvoir discriminant — Permet d'évaluer le pouvoir discriminant d'une mesure d'information. vi, 81, 82, 149, 150

production — Une chaîne non nulle X est dite productible à partir de son préfixe $X(1, j)$ si $X(1, j) \to X\pi$ et $j < l(X)$. 21–23, 27, 30

processus de production — Produit la chaîne X en m étapes de production. v, 7, 22–24, 26–28, 30, 31, 34, 37, 38, 45, 52

quantité significative — Quantité de la chaîne X bien expliquée par Y. 48, 59–63, 66, 138, 143

reproduction — Une extension $R = XQ$ de X est reproductible à partir de X si $Q \in v(R\pi)$. 21, 23, 27, 30

rétroaction — Si X apporte de l'information à Y et que Y apporte de l'information à X. 76, 128, 148

`retard` — Retarde une chaîne d'un caractère. 76, 129–132, 148

`SALZA` jointe concaténée — `SALZA` jointe basée sur l'encodage concaténé. 69–71, 155

`SALZA` jointe simultanée — `SALZA` jointe basée sur l'encodage simultané. 71, 76

`SALZA` — Nouvelle mesure d'information basée sur la décomposition de `GZIP`. iii–vii, 2–4, 47–50, 52, 54, 56, 58, 60–87, 90–92, 104, 117, 120, 122–127, 129, 130, 132–141, 143, 144, 147–151, 155–159, 162, 169–171

`SALZA` jointe cardinale — `SALZA` jointe basée sur l'encodage cardinal. 68

théorie algorithmique de l'information — Quantifie le contenu moyen en information d'un ensemble de messages grâce à la complexité de Kolmogorov. iii, 2–4, 7–9, 11–13, 15, 17–19, 26, 27, 29, 30, 42, 43, 45–47, 61, 67, 69, 71, 73, 75, 118, 122, 130, 137, 139

Notations probabilistes et combinatoire

X Séquence de n tirage de la variable aléatoire **X**. 9, 10, 75–77, 127–129

Y Séquence de n tirage de la variable aléatoire **Y**. 75–77, 127–129

H Entropie de Shannon. 75, 76, 129

I_c Information combinatoire. 8, 9

X Variable aléatoire discrète. 9, 10, 17, 18, 75

Y Variable aléatoire discrète. 75

Notions probabilistes et combinatoire

entropie combinatoire	Mesure le nombre de possibilités combinatoires. 7–9, 11, 12
entropie de Shannon	Nombre de bits moyen pour transmettre un message. 1, 2, 7–12, 17, 19, 24–26, 28, 45, 75, 76, 127, 129, 135, 137, 148
information combinatoire	Information transmise lorsque a est fixé. 8, 9
information mutuelle de Shannon	Mesure de la dépendance statistique de deux variables. 10, 18
théorie probabiliste de l'information	Quantifie le contenu moyen en information d'un ensemble de messages grâce à entropie de Shannon. 2, 3, 7, 8, 10, 12, 17–19, 45, 129, 130, 137

Bibliographie

[Abo+06] Mateo Aboy et al. "Interpretation of the Lempel-Ziv complexity measure in the context of biomedical signal analysis". In : *IEEE Transactions on Biomedical Engineering* 53.11 (2006), p. 2282-2288.

[AM11] Pierre-Olivier Amblard et Olivier JJ Michel. "On directed information theory and Granger causality graphs". In : *Journal of computational neuroscience* 30.1 (2011), p. 7-16.

[BCL02] Dario Benedetto, Emanuele Caglioti et Vittorio Loreto. "Language trees and zipping". In : *Physical Review Letters* 88.4 (2002), p. 048702.

[Ben+98] Charles H Bennett et al. "Information distance". In : *IEEE Transactions on information theory* 44.4 (1998), p. 1407-1423.

[CF05] David Pereira Coutinho et Mário AT Figueiredo. "Information theoretic text classification using the Ziv-Merhav method". In : *Iberian Conference on Pattern Recognition and Image Analysis*. Springer. 2005, p. 355-362.

[CGP05] Madalena Costa, Ary L Goldberger et C-K Peng. "Multiscale entropy analysis of biological signals". In : *Physical review E* 71.2 (2005), p. 021906.

[Cha66] Gregory J Chaitin. "On the length of programs for computing finite binary sequences". In : *Journal of the ACM (JACM)* 13.4 (1966), p. 547-569.

[Cil+16] Rudi Cilibrasi et al. *Complearn tool based on [CV05]*. 2016. url : `http://complearn.org/` (visité le 01/06/2018).

[CKV06] Haixiao Cai, Sanjeev R Kulkarni et Sergio Verdú. "Universal divergence estimation for finite-alphabet sources". In : *IEEE Transactions on Information Theory* 52.8 (2006), p. 3456-3475.

[CRAO05] Manuel Cebrián Ramos, Manuel Alfonseca et Alfonso Ortega. "Common pitfalls using the normalized compression distance : What to watch out for in a compressor". In : *Communications in information and systems* (2005).

[CT12] Thomas M Cover et Joy A Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[CV05] Rudi Cilibrasi et Paul MB Vitányi. "Clustering by compression". In : *IEEE Transactions on Information theory* 51.4 (2005), p. 1523-1545.

[Deu91] L. Peter Deutsch. *DEFLATE Compressed Data Format Specification, version 1.3*. 1991.

[Dia69] Robert B Dial. "Algorithm 360 : Shortest-path forest with topological ordering [H]". In : *Communications of the ACM* 12.11 (1969), p. 632-633.

[Eic00] Michael Eichler. "Graphical models in time series analysis". Thèse de doct. Universitat, Heidelberg, 2000.

[FB11] Y Rakhshani Fatmehsari et F Bahrami. "Lempel-ziv complexity criteria for nonlinear analysis of gait in patients with parkinson's disease". In : *Biomedical Engineering (ICBME), 2011 18th Iranian Conference of*. IEEE. 2011, p. 137-141.

[FDK15] Peter Foster, Simon Dixon et Anssi Klapuri. “Identifying cover songs using information-theoretic measures of similarity”. In : *IEEE/ACM Transactions on Audio, Speech and Language Processing (TASLP)* 23.6 (2015), p. 993-1005.

[FMG92] Meir Feder, Neri Merhav et Michael Gutman. “Universal prediction of individual sequences”. In : *IEEE transactions on Information Theory* 38.4 (1992), p. 1258-1270.

[Gra69] Clive WJ Granger. “Investigating causal relations by econometric models and cross-spectral methods”. In : *Econometrica : Journal of the Econometric Society* (1969), p. 424-438.

[Hes+03] Wolfram Hesse et al. “The use of time-variant EEG Granger causality for inspecting directed interdependencies of neural assemblies”. In : *Journal of neuroscience methods* 124.1 (2003), p. 27-44.

[Huf52] David A Huffman. “A method for the construction of minimum-redundancy codes”. In : *Proceedings of the IRE* 40.9 (1952), p. 1098-1101.

[IM+15] Antonio J Ibáñez-Molina et al. “Multiscale Lempel–Ziv complexity for EEG measures”. In : *Clinical Neurophysiology* 126.3 (2015), p. 541-548.

[JOP+01] Eric Jones, Travis Oliphant, Pearu Peterson et al. *SciPy : Open source scientific tools for Python*. [Online ; accessed <today>]. 2001–.

[JS10] Dominik Janzing et Bernhard Scholkopf. “Causal inference using the algorithmic Markov condition”. In : *IEEE Transactions on Information Theory* 56.10 (2010), p. 5168-5194.

[KB07] Markus Kalisch et Peter Bühlmann. “Estimating high-dimensional directed acyclic graphs with the PC-algorithm”. In : *Journal of Machine Learning Research* 8.Mar (2007), p. 613-636.

[Kol65] Andrei N Kolmogorov. “Three approaches to the quantitative definition of information”. In : *Problems of Information Transmission* 1.1 (1965), p. 1-7.

[Kol68] Andrei Kolmogorov. “Logical basis for information theory and probability theory”. In : *IEEE Transactions on Information Theory* 14.5 (1968), p. 662-664.

[LB96] Bil Lewis et Daniel L Berg. *A Guide to Multithreaded Programming : Threads Primer*. 1996.

[Li+04] Ming Li et al. “The similarity metric”. In : *IEEE transactions on Information Theory* 50.12 (2004), p. 3250-3264.

[Li+08] Yingjie Li et al. “Abnormal EEG complexity in patients with schizophrenia and depression”. In : *Clinical Neurophysiology* 119.6 (2008), p. 1232-1241.

[LV08] Ming Li et Paul Vitányi. *An introduction to Kolmogorov complexity and its applications. Texts in Computer Science*. T. 8. Springer, New York, 2008.

[LZ76] Abraham Lempel et Jacob Ziv. “On the complexity of finite sequences”. In : *IEEE Transactions on information theory* 22.1 (1976), p. 75-81.

[Mac+08] A Macedonas et al. “Dictionary based color image retrieval”. In : *Journal of Visual Communication and Image Representation* 19.7 (2008), p. 464-470.

[Mas90] James Massey. "Causality, feedback and directed information". In : *Proc. Int. Symp. Inf. Theory Applic.(ISITA-90)*. Citeseer. 1990, p. 303-305.

[Mor92] Michel Morvan. "Les Origines Linguistiques du Basque : L'Ouralo-Altaïque". In French. Thèse de doct. Presses Universitaires de Bordeaux : Université de Bordeaux 3, 1992.

[Onu] *United Nations General Assembly Resolution 217 A (III) of 10 December 1948*. 1948. URL : `http://www.un.org/Overview/rights.html` (visité le 01/06/2018).

[Pav] Igor Pavlov. *7z Format*.

[Pea09] Judea Pearl. *Causality*. Cambridge university press, 2009.

[Per69] Georges Perec. *La disparition*. Editions Gallimard, 1969.

[Qui+11] Christopher J Quinn et al. "Estimating the directed information to infer causal relationships in ensemble neural spike train recordings". In : *Journal of computational neuroscience* 30.1 (2011), p. 17-44.

[Roh73] F James Rohlf. "Algorithm 76. Hierarchical clustering using the minimum spanning tree". In : *Comp. J.* 16 (1973), p. 93-95.

[Sew97] Julian Seward. "The bzip2 home page". In : *URL http ://www. bzip. org* (1997).

[Sha48] Claude Elwood Shannon. "A mathematical theory of communication". In : *Bell system technical journal* 27.3 (1948), p. 379-423.

[SJS10] Bastian Steudel, Dominik Janzing et Bernhard Schölkopf. "Causal Markov condition for submodular information measures". In : *arXiv preprint arXiv :1002.4020* (2010).

[Sol60] R.J. Solomonoff. "A preliminary report on a general theory of inductive inference". In : *Technical Report ZTB-138* 1.1 (1960).

[SS+73] Peter HA Sneath, Robert R Sokal et al. *Numerical taxonomy. The principles and practice of numerical classification.* 1973.

[TCM11] Mehran Talebinejad, Adrian DC Chan et Ali Miri. "A Lempel–Ziv complexity measure for muscle fatigue estimation". In : *Journal of Electromyography and Kinesiology* 21.2 (2011), p. 236-241.

[VDG12] Miguel Angel Veganzones, Mihai Datcu et Manuel Grana. "Dictionary based Hyperspectral Image Retrieval." In : *ICPRAM (1)*. 2012, p. 426-432.

[ZL77] Jacob Ziv et Abraham Lempel. "A universal algorithm for sequential data compression". In : *IEEE Transactions on information theory* 23.3 (1977), p. 337-343.

[ZL78] Jacob Ziv et Abraham Lempel. "Compression of individual sequences via variable-rate coding". In : *IEEE transactions on Information Theory* 24.5 (1978), p. 530-536.

[ZM93] Jacob Ziv et Neri Merhav. "A measure of relative entropy between individual sequences with application to universal classification". In : *IEEE transactions on information theory* 39.4 (1993), p. 1270-1279.

[ZRB05] Steeve Zozor, Philippe Ravier et Olivier Buttelli. “On Lempel–Ziv complexity for multidimensional data analysis”. In : *Physica A : Statistical Mechanics and its Applications* 345.1-2 (2005), p. 285-302.

Résumé

Les données sous forme de chaîne de symboles sont très variées (ADN, texte, EEG quantifié,...) et ne sont pas toujours modélisables. Une description universelle des chaînes de symboles indépendante des probabilités est donc nécessaire. La complexité de Kolmogorov a été introduite en 1960 pour répondre à cette problématique. Le concept est simple : une chaîne de symboles est complexe quand il n'en existe pas une description courte. La complexité de Kolmogorov est le pendant algorithmique de l'entropie de Shannon et permet de définir la théorie algorithmique de l'information. Cependant, la complexité de Kolmogorov n'est pas calculable en un temps fini ce qui la rend inutilisable en pratique.

Les premiers à rendre opérationnelle la complexité de Kolmogorov sont Lempel et Ziv en 1976 qui proposent de restreindre les opérations de la description. Une autre approche est d'utiliser la taille de la chaîne compressée par un compresseur sans perte. Cependant ces deux estimateurs sont mal définis pour le cas conditionnel et le cas joint, il est donc difficile d'étendre la complexité de Lempel-Ziv ou les compresseurs à la théorie algorithmique de l'information.

Partant de ce constat, nous introduisons une nouvelle mesure d'information universelle basée sur la complexité de Lempel-Ziv appelée SALZA. L'implémentation et la bonne définition de notre mesure permettent un calcul efficace des grandeurs de la théorie algorithmique de l'information.

Les compresseurs sans perte usuels ont été utilisés par Cilibrasi et Vitányi pour former une métrique universelle très populaire : la distance de compression normalisée [NCD]. Dans le cadre de cette application, nous proposons notre propre estimateur, la NSD, et montrons qu'il s'agit d'une semi-distance universelle sur les chaînes de symboles. L'application à de la classification hiérarchique montre que la NSD surclasse la NCD en s'adaptant naturellement à davantage de diversité des données et en définissant le conditionnement adapté grâce à SALZA.

En utilisant les qualités de prédiction universelle de la complexité de Lempel-Ziv, nous explorons ensuite les questions d'inférence de causalité. Dans un premier temps, les conditions algorithmiques de Markov sont rendues calculables grâce à SALZA. Puis en définissant pour la première l'information dirigée algorithmique, nous proposons une interprétation algorithmique de la causalité de Granger algorithmique. Nous montrons, sur des données synthétiques et réelles, la pertinence de notre approche.

Mots clés : Complexité de Lempel-Ziv, Théorie algorithmique de l'information, Mesure d'information universelle, Classification non supervisée, Causalité.

Abstract

Data in the form of strings are varied (DNA, text, quantify EEG) and cannot always be modeled. A universal description of strings, independent of probabilities, is thus necessary. The Kolmogorov complexity was introduced in 1960 to address the issue. The principle is simple : a string is complex if a short description of it does not exist. The Kolmogorov complexity is the counterpart of the Shannon entropy and defines the algorithmic information theory. Yet, the Kolmogorov complexity is not computable in finit time making it unusable in practice.

The first ones to make operational the Kolmogorov complexity are Lempel and Ziv in 1976 who proposed to restrain the operations of the description. Another approach uses the size of the compressed string by a lossless data compression algorithm. Yet these two estimators are not well-defined regarding the joint and conditional complexity cases. So, compressors and Lempel-Ziv complexity are not valuable to estimate algorithmic information theory.

In the light of this observation, we introduce a new universal information measure based on the Lempel-Ziv complexity called SALZA. The implementation and the good definition of our measure allow computing efficiently values of the algorithmic information theory.

Usual lossless compressors have been used by Cilibrasi and Vitányi to define a very popular universal classifier : the normalized compression distance [NCD]. As part of this application, we introduce our own estimator, called the NSD, and we show that the NSD is a universal semi-distance between strings. NSD surpasses NCD because it gets used to a large data set and uses the adapted conditioning with SALZA.

Using the accurate universal prediction quality of the Lempel-Ziv complexity, we explore the question of causality inference. At first, we compute the algorithmic causal Markov condition thanks to SALZA. Then we define, for the first time, the algorithmic directed information and based on it we introduce the algorithmic Granger causality. The relevance of our approach is demonstrated on real and synthetic data.

Keywords : Lempel-Ziv complexity, Algorithmic information theory, Universal information measure, Unsupervised classification, Causality.