Une implantation de la théorie de l'information

François CAYRE Nicolas LE BIHAN Isabelle SIVIGNON
CNRS, Univ. Grenoble Alpes, Grenoble-INP, GIPSA-Lab
F-38000 Grenoble

Résumé – Ce travail décrit une implantation des opérateurs usuels de la théorie de l'information sur des multi-ensembles de chaînes de symboles. Elle se fonde sur *(i)* un format de représentation compressée, et *(ii)* un algorithme d'inférence de la syntaxe la plus probable (MPS), permettant l'implantation de trois mesures d'une quantité d'information, dont une est exacte.

Abstract – This work describes an implementation of the usual information theory operators on multisets of symbol strings. It is based on *(i)* a compressed representation format and *(ii)* an algorithm that will infer the most probable syntax (MPS), thus enabling the implementation of three information measures, one of which is exact.

1 Introduction

Le versant algorithmique (déterministe) de la théorie de l'information attache une quantité d'information à une chaîne de symboles. Pour cela, on utilise généralement un compresseur, ce qui permet d'approcher la complexité de Kolmogorov [8] et fournit alors une mesure d'information contenue dans une chaîne. Soit $\{x\}_n$ avec $x_i \in \mathcal{A}^*$ un multi-ensemble de n chaînes de longueur finie sur l'alphabet $\mathcal{A}$. La chaîne vide est notée ϵ, la concaténation de plusieurs chaînes est notée par la juxtaposition de leur nom.

Le but de ce travail est l'obtention d'une mesure $K(\{x\}_n) = K(x_1, \ldots, x_n)$ vérifiant en machine les propriétés suivantes :

$$\begin{aligned} K(\epsilon) &= 0 \\ K(x,y) &= K(y,x) \\ K(x,x) &= K(x) \\ K(x) &\leq K(xy) \\ 0 \leq K(x), K(y) &\leq K(x,y) \leq K(x) + K(y) \end{aligned}$$

Une mesure K est dite exacte si et seulement si les relations ci-dessus sont vérifiées exactement en machine. La raison pour laquelle nous parlons de multi-ensembles de chaînes est, comme il est manifeste ci-dessus, qu'il est possible d'avoir plusieurs fois la même chaîne en entrée. Nous utiliserons plus bas ce cas limite aux fins d'illustration, sans perte de généralité évidemment.

À la différence de travaux précédents [2], ce travail substitue un *compresseur joint* à ceux habituellement utilisés, qui ne fonctionnent que sur une seule chaîne d'entrée, et nécessitent par là de concaténer les $\{x\}_n$ pour approcher les quantités jointes ci-dessus (introduisant ce faisant quantité d'erreurs numériques incontrôlables, également liées aux limites des tampons internes des compresseurs [1]). Nous proposons ici une telle mesure appelée `mpx`, reposant sur un compresseur joint et un format compressé (`mpz`).

Le lecteur trouvera de fortes ressemblances avec [6] puisque nous inférons également une grammaire hors-contexte adaptée au codage par plage. Néanmoins, l'intuition première ici est venue tant de Sequitur [9] que de la nécessité d'implanter un format de représentation pour un analyseur syntaxique modulaire [4] lors d'un projet annexe en enseignement.

D'une certaine manière, ce travail conserve une dimension pédagogique puisqu'il permet d'expliciter les liens entre différentes mesures d'information, allant d'une mesure exacte sur le treillis de Shannon [10], dont la relation d'ordre trouve alors une sémantique naturelle, à une approximation de la complexité de Kolmogorov bipartite [8] davantage cohérente que celle permise par un compresseur usuel.

2 Représentation syntaxique

Deux outils principaux sont introduits pour le codage de $\{x\}_n$: les *parselettes* et les *chaînes communes modales*.

2.1 Parselettes

Les chaînes $\{x\}_n$ sont décrites à l'aide de références post-fixées (par `?` ou `*`) ou non, et des paramètres de répétition. Par exemple, on a l'équivalence suivante en mémoire :

```
x   =   abbbccdabbbdabbbb
x   ->  a b* 3 c* 2 d a b* 3 d a b* 4
```

FIGURE 1 : Représentation en mémoire d'une chaîne.

Une *parselette* est la conjonction (le ET logique figurant la concaténation) de deux références post-fixées se répétant dans au moins une des $\{x\}_n$.[1]

Soit $\mathcal{P}$ l'ensemble des parselettes. Le dictionnaire $\mathcal{A} \cup \mathcal{P}$ permet de représenter les chaînes d'entrée :

```
x    =   abbbccdabbbdabbbb
p0   ->  a b*
p1   ->  d p0
x    ->  p0 3 c* 2 p1* 2 3 4
```

FIGURE 2 : Chaîne d'exemple après extraction des parselettes.

[1] En toute généralité, `mpx` implante les trois opérateurs de répétition, ainsi que la disjonction (OU logique), même si `?` n'est pas utilisé ici, non plus que la disjonction de références.

La structure syntaxique d'une chaîne est donnée par le parcours récursif de la grammaire des parselettes, qui consomme les paramètres de répétition à la volée, jusqu'aux feuilles non incluses. Dans notre exemple, on a la structure arborescente suivante :

```
(a b*) c* ((d (a b*)) (d (a b*)))
```

FIGURE 3 : Structure syntaxique inférée à partir de $x = abbbccdabbbdabbbb$.

2.2 Chaînes communes modales (MCS)

Si les parselettes permettent de capturer les structures syntaxiques, elles ne peuvent rendre compte de structures dans l'espace des paramètres de répétition. C'est tout l'objet des chaînes communes modales.

Des sous-chaînes sont réputées communes si elles partagent la même structure syntaxique : seuls les éventuels paramètres de répétition sur les feuilles de l'arbre syntaxique peuvent varier.

On construit la *chaîne commune modale* (MCS) d'un multi-ensemble de sous-chaînes communes en lui affectant les valeurs modales minimales de leurs paramètres de répétition.

Les MCS ne font pas partie *stricto sensu* du dictionnaire, mais doivent pouvoir être référencées. Une référence à une MCS est suivie d'un bit indiquant s'il y a des différences à prendre en compte pour instancier une sous-chaîne. Dans l'affirmative, on écrit pour chaque paramètre un bit indiquant si la valeur de répétition courante doit être modifiée, et de combien le cas échéant. Une MCS ne peut en référencer une autre, et une parselette ne peut référencer une MCS.

2.3 Exemple complet

Soit $x = abbbccdabbbdabbbb$ (comme ci-dessus), $y = abbcccdabbbdabb$ et $z = babbccdabbbdabbbb$.

Avant le calcul des MCS, on a :

```
p0  ->  a b*
p1  ->  d p0
x   ->  p0 3 c* 2 p1* 2 3 4
y   ->  p0 2 c* 3 p1* 2 3 2
z   ->  b p0 2 c* 2 p1* 2 3 4
```

Puis après calcul des MCS :

```
p0    ->  a b*
p1    ->  d p0
mcs0  :=  p0 2 c* 2 p1* 2 3 4
x     ->  mcs0 1 1 1 0 0 0 0
y     ->  mcs0 1 0 1 1 0 0 1 -2
z     ->  b mcs0 0
```

L'idée des MCS est d'assurer des descriptions de sous-chaînes communes dont la longueur varie aussi uniformément que possible avec l'ampleur de leurs différences.

Dans cet exemple, seule z commande une recopie exacte de la MCS, le bit qui suit la référence à la MCS est alors un zéro (`mpx` est donc capable d'émuler LZ78 [12] en ajoutant seulement un bit à une référence).

Pour les deux premières, le bit suivant la référence à la MCS est à un, indiquant au moins une différence avec les paramètres de répétition stockés dans cette dernière. La chaîne x se réécrit à partir de la MCS en faisant varier seulement le premier paramètre de répétition. Le codage de la chaîne y montre que nous devons être capable de représenter des entiers relatifs : nous utilisons pour cela une extension bien connue des entiers naturels (les valeurs impaires codent un entier négatif, et les paires un entier positif – et nous n'avons pas besoin de représenter le zéro).

2.4 Format compressé `.mpz`

On en déduit facilement un format de représentation compressé pour un multi-ensemble de chaînes d'entrée $\{x\}_n$ décrit par les parselettes $\mathcal{P} = \{p\}_{|\mathcal{P}|}$ et m MCS.

L'en-tête est constitué de n, $|\mathcal{P}|$ et m ($|\mathcal{A}|$ est réputé connu). Puis on écrit le dictionnaire (les parselettes). Ensuite les MCS sont écrites. Et enfin les chaînes $\{x\}_n$.

Comme l'on souhaite construire des mesures aussi exactes que possible, les MCS et chaque x_i sont codées à l'aide de leurs propres codeurs entropiques (un pour les références au dictionnaire et l'autre pour les valeurs des paramètres de répétition), cela afin d'assurer la séparation des contextes d'encodage (l'ordre dans lequel les chaînes sont compressées n'influe pas sur le codage des chaînes elles-mêmes). De même, les parselettes utilisent leur propre codeur entropique pour les références au dictionnaire qu'elles contiennent. Les opérateurs de répétition utilisent des codes préfixes statiques.

Ce format implante donc une variante de la complexité de Kolmogorov bipartite, similaire à MML ou MDL (le modèle étant les parselettes, et sa spécification étant l'ensemble des valeurs des paramètres de répétition).

3 Syntaxe la plus probable (MPS)

La construction du flux compressé consiste à extraire d'abord les parselettes, c'est le cœur de l'algorithme MPS. Une seconde phase consiste ensuite à extraire les MCS.

3.1 Deux algorithmes gloutons

MPS est un algorithme glouton qui trouve l'une après l'autre les parselettes les plus probables, et réécrit les chaînes $\{x\}_n$ à la volée en mémoire après l'extraction de chaque nouvelle parselette. Il termine lorsque plus aucune parselette ne peut être extraite.

Puis, un autre algorithme glouton extrait les MCS les unes après les autres, cette fois par longueur décroissante (on souhaite capturer des MCS de moins en moins longues pour que le codage d'un multi-ensemble soit le plus compact possible, dans l'esprit du calcul de la complexité de Kolmogorov).

3.2 Propriétés

Il ressort à la fois du fonctionnement de MPS et du format de représentation `.mpz` que :

1. Toutes les parselettes nécessaires à la réécriture de chacune des $\{x\}_n$ seront toujours trouvées, et uniques (MPS calcule l'union des parselettes des $\{x\}_n$) ;

2. Le format `.mpz` peut émuler tant le codage par plage qu'un codage de Lempel-Ziv (LZ78), mais est strictement plus expressif car il implante une classe de langage régulier (une grammaire d'expressions régulières) au lieu de simples copier-coller.

4 Trois mesures d'information

On définit maintenant les trois mesures suivantes associées à un multi-ensemble $\{x\}_n$:

$$\begin{aligned} K_2(\{x\}_n) &= \text{ nombre de bits du flux compressé complet,} \\ K_d(\{x\}_n) &= \text{ nombre de bits des parselettes compressées,} \\ K_{\mathcal{P}}(\{x\}_n) &= \text{ nombre de parselettes } |\mathcal{P}|. \end{aligned}$$

4.1 Interprétation

Notons que seules les deux premières sont *a priori* susceptibles de recevoir une interprétation physique en attachant une probabilité $p(\{x\}_n) = 2^{-K(\{x\}_n)}$ aux chaînes d'entrée (car exprimées en bits).

Aussi, par ordre décroissant de sensibilité aux données, tout autant que par exactitude numérique croissante, K_2 représente une approximation de la complexité de Kolmogorov bipartite des $\{x\}_n$, K_d représente une mesure de la seule *complexité syntaxique* des $\{x\}_n$, dont $K_{\mathcal{P}}$ est une variante d'intérêt encore plus grossière (la fameuse « taille du dictionnaire »).

4.2 Implantation des opérateurs usuels

Pour une mesure d'information K donnée, `mpx` implante les deux opérateurs usuels comme suit :

$$\begin{aligned} K(\{x\}_n|\{z\}_q) &= K(\{x\}_n,\{z\}_q) - K(\{z\}_q) \\ I(\{x\}_n;\{y\}_r|\{z\}_q) &= K(\{x\}_n,\{z\}_q) + K(\{y\}_r,\{z\}_q) \\ &\quad -K(\{x\}_n,\{y\}_r,\{z\}_q) - K(\{z\}_q) \end{aligned}$$

L'information jointe conditionnelle $K(\{x\}_n|\{z\}_q)$ trouve à s'appliquer en classification (*cf. infra* la définition de $\text{NID}_{\mathcal{P}}$), et l'information mutuelle conditionnelle $I(\{x\}_n;\{y\}_r|\{z\}_q)$ est l'opérateur fondamental utilisé en inférence de causalité.

Sous réserve d'universalité [5], à étudier, les quantités ci-dessus convergent vers leur version probabiliste – à une vitesse qui reste néanmoins à déterminer.

4.3 Exactitude de $K_{\mathcal{P}}$

En notant $\mathcal{P}_x$ l'ensemble des parselettes du multi-ensemble $\{x\}_n$, et par définition d'une parselette, on a nécessairement $\mathcal{P}_x, \mathcal{P}_y \subset \mathcal{P}_{x,y} = \mathcal{P}_x \cup \mathcal{P}_y$ et $\mathcal{P}_{x,z}, \mathcal{P}_{y,z} \subset \mathcal{P}_{x,y,z} = \mathcal{P}_x \cup \mathcal{P}_y \cup \mathcal{P}_z$. Donc $K_{\mathcal{P}}(\{x\}_n|\{z\}_q) \geq K_{\mathcal{P}}(\{x\}_n|\{y\}_r,\{z\}_q)$ et $I(\{x\}_n;\{y\}_r|\{z\}_q) \geq 0$.

De plus, concaténer une chaîne y à la suite d'une autre chaîne x ne peut que causer la découverte éventuelle de parselettes supplémentaires, donc $K_{\mathcal{P}}(x) \leq K_{\mathcal{P}}(xy)$.

Les propriétés manquantes se déduisent directement (l'invariance par permutation des chaînes d'entrée tenant au fait que toutes les décisions de création de nouvelles parselettes par MPS sont prises sur l'intégralité des représentations des chaînes en mémoire). Par défaut, `mpx` utilise $K_{\mathcal{P}}$.

4.4 Treillis métrique syntaxique

En suivant Shannon [10], on définit la relation d'ordre suivante :

$$\{x\}_n \leq \{y\}_r \iff K_{\mathcal{P}}(\{x\}_n|\{y\}_r) = 0.$$

L'égalité est réputée tenir lorsque $|\mathcal{P}_x| = |\mathcal{P}_y|$.

La structure de treillis de l'information trouve ici une sémantique plus précise que celle de Shannon (« y est une abstraction de x ») : la réécriture des $\{x\}_n$ ne nécessite pas de créer davantage de parselettes que celles nécessaires à la réécriture des $\{y\}_r$.

De même, `mpx` implante le calcul de trois distances, dont une distance informationnelle normalisée [7] :

$$\text{NID}_{\mathcal{P}}(\{x\}_n,\{y\}_r) = \frac{\max\{K_{\mathcal{P}}(\{x\}_n|\{y\}_r), K_{\mathcal{P}}(\{y\}_r|\{x\}_n)\}}{\max\{K_{\mathcal{P}}(\{x\}_n), K_{\mathcal{P}}(\{y\}_r)\}}.$$

5 Choix d'implantation pour `mpx`

Pour le codage entropique, `mpx` utilise des codeurs de Huffman dynamiques [11] qui approchent les performances des codes statiques en deux passes sans avoir leur complexité de mise en œuvre (comme par exemple dans `DEFLATE` [3]).

Une table des décalages par rapport à la fin du dictionnaire est ajoutée pour permettre un accès rapide à une chaîne compressée arbitraire. La taille de cette table ne saurait entrer dans les longueurs de flux compressé définies ci-dessus.

L'implantation `mpx` travaille sur des octets ($|\mathcal{A}| = 256$) et utilise une représentation compacte des parselettes sur 64 bits : les indices de parselettes et de MCS sont alors limités à 29 bits[2] (une référence post-fixée étant donc décrite sur 32 bits).

6 Performances

On compare ici `mpx` (compilé avec `-O3`) et `gzip`, qui dispose d'un tampon interne de 32KiB (la distance maximum dans le passé pour la fenêtre glissante de LZ77). Pour cela, on utilise le code source (en C) de `mpx` (`mpx.c`, 143446 octets).

6.1 En tant que simple compresseur

Compresseur	Temps (s)	Mém. (KiB)	Taille (octets)
`gzip`	0.01	1892	24372
`mpx`	1.53	10876	39104

TABLE 1 : Comparaison pour `gzip` *vs.* `mpx` des temps, occupations en mémoire et tailles sur disque.

De manière assez évidente, `mpx` n'est pas destiné à la compression usuelle de fichiers individuels. C'est d'autant plus vrai que la nécessité de charger l'intégralité des chaînes en mémoire interdit un fonctionnement sous forme de filtre.

Les raisons de la moindre compression de `mpx` tiennent assez vraisemblablement à trois facteurs, listés par ordre probable d'influence :

[2] Cet espace pourrait être plus grand, mais `mpx` implante davantage que strictement nécessaire (disjonction de parselettes et opérateur `?` notamment). Cela dit, disposer d'un peu plus de 500 millions de parselettes devrait permettre de voir venir.

1. L'entropie, entendue comme le nombre de bits nécessaires à fixer un état particulier d'un système, croît à mesure que la complexité interne du système en question est importante. Or, `gzip` ne manie que des copier-coller (des couples de distance dans le passé et de longueur de sous-chaîne à recopier), quand `mpx` implante une grammaire d'expressions régulières ;

2. Les codes de Huffman dynamiques utilisent un bit de plus par symbole qu'une version en deux passes ;

3. `mpx` implante un sur-ensemble de ce qui serait strictement nécessaire, et cela se paye avec quelques bits supplémentaires par parselette.

6.2 En tant que compresseur joint

On compresse maintenant avec `gzip` la concaténation de deux exemplaires du fichier de test précédent (on ne peut procéder autrement), et on donne deux fois le même fichier à compresser à `mpx`.

Compresseur	Temps (s)	RSS (KiB)	Taille (octets)
`gzip`	0.03	1696	48291
`mpx`	5.7	17516	39115

TABLE 2 : Comparaison pour `gzip` *vs.* `mpx` des temps, occupations en mémoire et tailles sur disque.

La taille du tampon interne de `gzip` lui interdit de faire le rappel qu'il aurait idéalement dû faire de l'intégralité de la première occurrence du fichier de test : la taille de son flux compressé est alors doublée par rapport à celle de l'expérience précédente. Cela contrevient à l'intuition sous-tendant l'utilisation d'un compresseur usuel pour estimer une complexité de Kolmogorov.

On objectera que `xz`, avec son tampon plus important (1 MiB par défaut), s'en serait mieux sorti. Néanmoins, la question de l'ordre dans lequel effectuer les concaténations pour le calcul de $I(x;y|z)$ reste entière, et d'autant plus prégnante qu'on aurait bien davantage de chaînes dans un calcul d'inférence de causalité. Par construction, `mpx` n'est pas concerné par ce problème.

6.3 Inexactitude relative de K_2 *vs.* $K_\mathcal{P}$

Comme anticipé, la valeur de $K_\mathcal{P}(\texttt{mpx.c}, \texttt{mpx.c}|\texttt{mpx.c})$ est nulle, alors que la complexité de Kolmogorov bipartite s'élève à seulement 85 bits (contre 8×23919 pour `gzip`) :

```
$ ./bin/mpx info -i=mpx.c,mpx.c -p=mpx.c
0
$ ./bin/mpx info -i=mpx.c,mpx.c -p=mpx.c
--bits=kolmogorov
85
```

FIGURE 4 : Exactitude de $K_\mathcal{P}$ et approximation de la complexité de Kolmogorov bipartite.

7 Perspectives

Il reste notamment à profiler et empaqueter le logiciel, les futurs développements seront recensés sur cette page :
`https://hackmd.uvolante.org/mpx`
Les opérateurs ici définis devront être intégrés dans des routines de partitionnement et d'inférence de causalité.

Références

[1] Manuel CEBRIÁN, Manuel ALFONSECA et Alfonso ORTEGA : Common Pitfalls Using the Normalized Compression Distance : What to Watch Out for in a Compressor. *Communications in Information in Systems*, 5(4):367–384, 2005. En ligne.

[2] Rudi L. CILIBRASI et Paul M.B. VITANIY : Clustering by Compression. *IEEE Transactions on Information Theory*, 51(4):1523–1545.

[3] L. Peter DEUTSCH : IETF RFC 1951 : DEFLATE Compressed Data Format Specification v. 1.3, 1996. En ligne.

[4] Bryan FORD : Parsing Expression Grammars : A Recognition Based Syntactic Foundation. *In Proc. 31st ACM SIGPLAN-SIGACT Symposium of Programming Languages*, pages 111–122, 2004.

[5] John C. KIEFFER et En-hui YANG : Grammar-Based Codes : A New Class of Universal Lossless Source Codes,. *IEEE Transactions on Information Theory*, 46(3):737–754. En ligne.

[6] Tomasz KOCIUMAKA, Gonzalo NAVARRO et Nicola PREZZA : Toward a Definitive Compressibility Measure for Repetitive Sequences. *IEEE Transactions on Information Theory*, 69(4):2074–2092, 2023.

[7] Ming LI, Xin CHEN, Xin LI, Ma BIN et Paul M.B. VITANYI : The Similarity Metric. *IEEE Transactions on Information Theory*, 50(12):3250–3264, 2004.

[8] Ming LI et Paul M.B. VITANIY : *An Introduction to Kolmogorov Complexity and its Applications*. Springer Publishing, New York, USA, 2019.

[9] Craig G. Nevill MANNING et Ian H. WITTEN : Identifying Hierarchical Structure in Sequences : A Linear-Time Algorithm. *Journal of Artificial Intelligence Research*, 7:67–82, Aug. 1997. En ligne.

[10] Claude SHANNON : The Lattice Theory of Information. *Transactions of the IRE Professional Group on Information Theory*, 1(1):105–107, 1953.

[11] Jeffrey S. VITTER : Design and Analysis of Dynamic Huffman Codes. *Journal of the ACM*, 34, 1987.

[12] Jacob ZIV et Abraham LEMPEL : Compression of Individual Sequences via Variable-Rate Coding. *IEEE Transactions on Information Theory*, 24(5):530–536, 1978.