Les données sont D[1]

Étude de méthodes de compression sans pertes

Thomas Bagrel

Lycée Henri Poincaré, Nancy

TIPE session 2018

Aperçu

Régularités et gains
- Théorie
- ZIP recursif

Composantes de la compression

Codage
- Problème résolu
- Inefficacité de HUFFMAN – pourquoi

Modèles généraux
- BITWISE ENCODER et PPM
- BITWISE PPM et BITWISE PPM FLAT

Impact de la transformée
- BWT régularise nos données
- La BWT en action

Régularités et gains

Théorie

Compression des données sans pertes

Régularités et gains

Théorie

Compression des données sans pertes

- exploiter les régularités des données

Régularités et gains

Théorie

Compression des données sans pertes

- exploiter les régularités des données
- données aléatoires : pas de gain

Régularités et gains

Théorie

Compression des données sans pertes

- exploiter les régularités des données
- données aléatoires : pas de gain

Théorème. (Entropie de Shannon)

$$\mathrm{H}(\mathrm{S}) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

$\mathrm{H}(\mathrm{S})$: *nb. de bits moyen par symbole de la source*

Régularités et gains

Théorie

Compression des données sans pertes

- exploiter les régularités des données
- données aléatoires : pas de gain

Théorème. (Entropie de SHANNON)

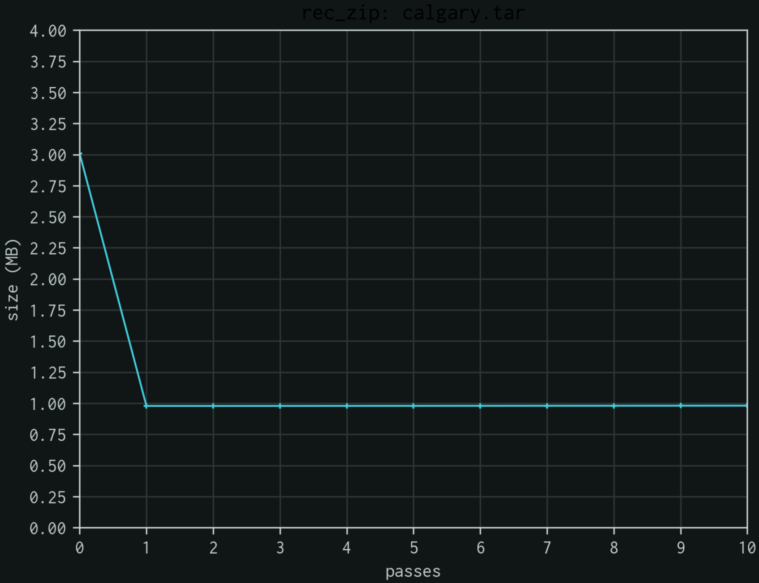
$$\mathrm{H}(\mathrm{S}) = -\sum_{i=1}^{n} p_i \log_2(p_i)$$

$\mathrm{H}(\mathrm{S})$: *nb. de bits moyen par symbole de la source*

- une fois les données compressées (dérivées) une fois, plus aucune régularité

Régularités et gains

ZIP recursif



Composantes de la compression

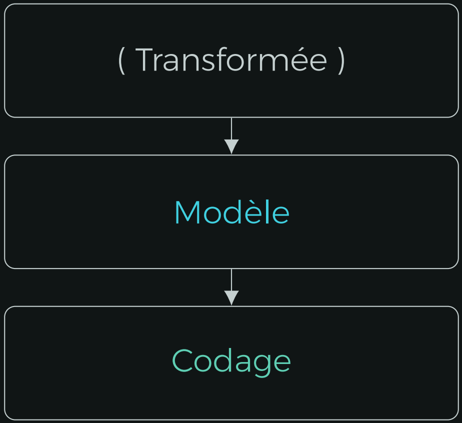
(Transformée)

Composantes de la compression

(Transformée)

Modèle

Composantes de la compression



Codage

Problème résolu

Le codage, contrairement aux apparences, est un problème résolu

Codage

Problème résolu

Le codage, contrairement aux apparences, est un problème résolu

- Si les p_i sont connus, la limite de compression théorique est donnée par SHANNON

Codage

Problème résolu

Le codage, contrairement aux apparences, est un problème résolu

- Si les p_i sont connus, la limite de compression théorique est donnée par SHANNON
- HUFFMAN permet d'approcher cette limite

Codage

Problème résolu

Le codage, contrairement aux apparences, est un problème résolu

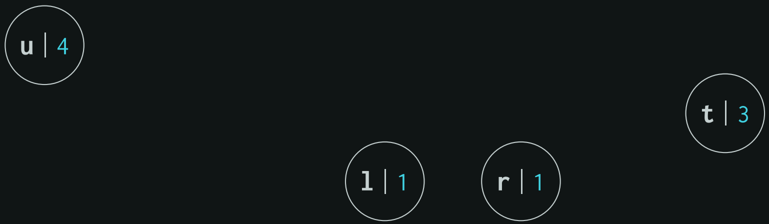
- Si les p_i sont connus, la limite de compression théorique est donnée par SHANNON
- HUFFMAN permet d'approcher cette limite
- Le codage arithmétique l'atteint

Codage

Problème résolu

Le codage arithmétique sera détaillé plus loin.

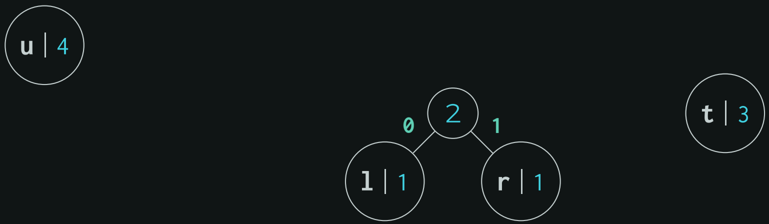
Codage d'HUFFMAN pour `turlututu`



Codage

Problème résolu

Le codage arithmétique sera détaillé plus loin.
Codage d'HUFFMAN pour `turlututu`

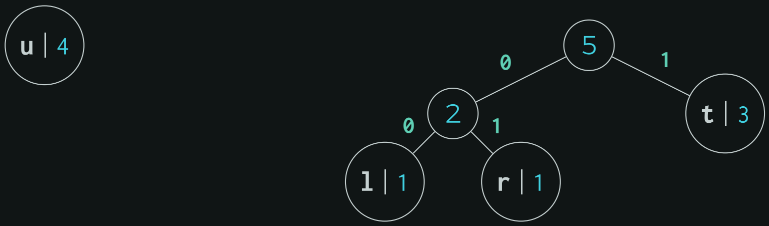


Codage

Problème résolu

Le codage arithmétique sera détaillé plus loin.

Codage d'Huffman pour `turlututu`

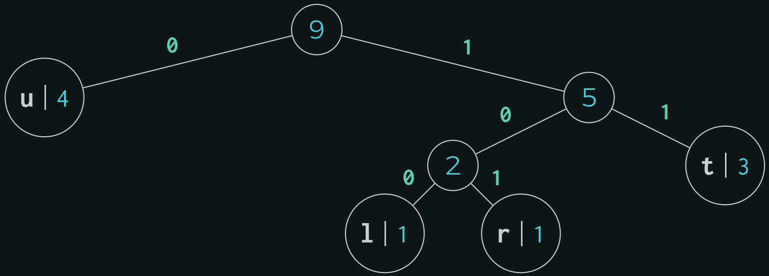


Codage

Problème résolu

Le codage arithmétique sera détaillé plus loin.

Codage d'HUFFMAN pour `turlututu`

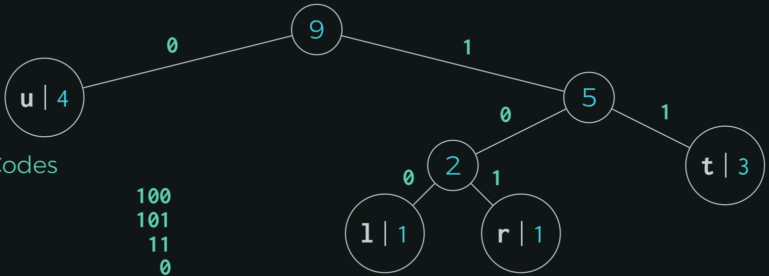


Codage

Problème résolu

Le codage arithmétique sera détaillé plus loin.

Codage d'HUFFMAN pour `turlututu`



Codes

l	100
r	101
t	11
u	0

Codage

Inefficacité de HUFFMAN – pourquoi

En pratique, efficacité de 32 %

Codage

Inefficacité de HUFFMAN – pourquoi

En pratique, efficacité de 32 %

En appliquant directement SHANNON aux fréquences d'apparition, on commet des erreurs

Codage

Inefficacité de HUFFMAN – pourquoi

En pratique, efficacité de 32 %

En appliquant directement SHANNON aux fréquences d'apparition, on commet des erreurs

- un symbole n'est pas indépendant des précédents

Codage

Inefficacité de HUFFMAN – pourquoi

En pratique, efficacité de 32 %

En appliquant directement SHANNON aux fréquences d'apparition, on commet des erreurs

- un symbole n'est pas indépendant des précédents
- par exemple, en Français, q→u est plus fréquent que q→z

Codage

Inefficacité de HUFFMAN – pourquoi

En pratique, efficacité de 32 %

En appliquant directement SHANNON aux fréquences d'apparition, on commet des erreurs

- un symbole n'est pas indépendant des précédents
- par exemple, en Français, q→u est plus fréquent que q→z
- en quelque sorte, on oublie le caractère lipschitzien de nos données

Codage

Inefficacité de HUFFMAN – pourquoi

En pratique, efficacité de 32 %

En appliquant directement SHANNON aux fréquences d'apparition, on commet des erreurs

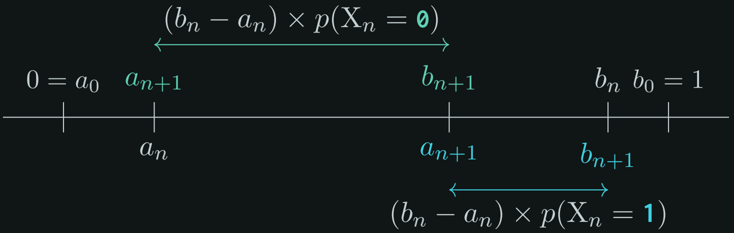
- un symbole n'est pas indépendant des précédents
- par exemple, en Français, q→u est plus fréquent que q→z
- en quelque sorte, on oublie le caractère lipschitzien de nos données

Il faut donc un modèle

Modèles généraux

BITWISE ENCODER et PPM

Codage arithmétique $\quad p(\mathrm{X}_n = \mathtt{0}) = 0{,}65$

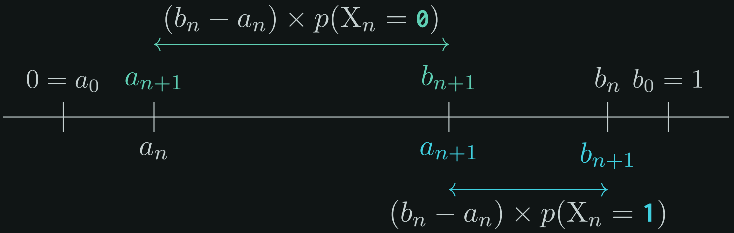


Modèles généraux

BITWISE ENCODER et PPM

Codage arithmétique

$p(X_n = \mathbf{0}) = 0{,}65$

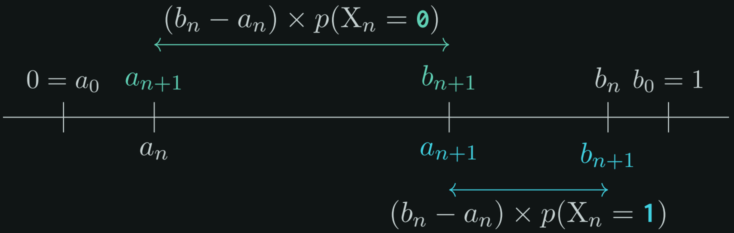


- Sous-intervalles proportionnels aux p_i

Modèles généraux

BITWISE ENCODER et PPM

Codage arithmétique $\quad p(\mathrm{X}_n = \mathtt{0}) = 0{,}65$

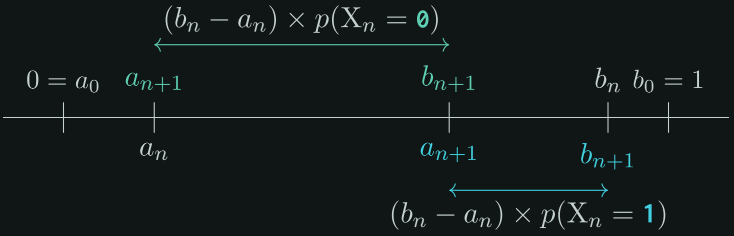


- Sous-intervalles proportionnels aux p_i
- En pratique, avec des p_i non fixes, gérer plus de quelques symboles est compliqué

Modèles généraux

BITWISE ENCODER et PPM

Codage arithmétique $p(\mathrm{X}_n = \mathtt{0}) = 0{,}65$



- Sous-intervalles proportionnels aux p_i
- En pratique, avec des p_i non fixes, gérer plus de quelques symboles est compliqué
- Implémenté avec des entiers pour représenter $[0;1[$

Modèles généraux

BITWISE ENCODER et PPM

PPM

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents
- Un jeu de p_i pour chaque contexte différent

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents
- Un jeu de p_i pour chaque contexte différent
- Si le contexte n'a jamais été rencontré, on retombe sur un contexte d'ordre $\mathrm{N}-1$ (*order fallback*) etc.

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents
- Un jeu de p_i pour chaque contexte différent
- Si le contexte n'a jamais été rencontré, on retombe sur un contexte d'ordre $\mathrm{N}-1$ (*order fallback*) etc.
- Codage arithmétique

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents
- Un jeu de p_i pour chaque contexte différent
- Si le contexte n'a jamais été rencontré, on retombe sur un contexte d'ordre $\mathrm{N}-1$ (*order fallback*) etc.
- Codage arithmétique

Inconvénients et solutions

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents
- Un jeu de p_i pour chaque contexte différent
- Si le contexte n'a jamais été rencontré, on retombe sur un contexte d'ordre $\mathrm{N}-1$ (*order fallback*) etc.
- Codage arithmétique

Inconvénients et solutions

- Encoder 256 symboles avec des p_i variables est dur à implémenter sans pertes

Modèles généraux

BITWISE ENCODER et PPM

PPM

- Le contexte est constitué des N symboles précédents
- Un jeu de p_i pour chaque contexte différent
- Si le contexte n'a jamais été rencontré, on retombe sur un contexte d'ordre $\mathrm{N} - 1$ (*order fallback*) etc.
- Codage arithmétique

Inconvénients et solutions

- Encoder 256 symboles avec des p_i variables est dur à implémenter sans pertes
- On pourrait donc encoder bit par bit au lieu de symbole par symbole

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit
- 49 % d'efficacité !

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit
- 49 % d'efficacité !
- Mais l'*order fallback* occasionne des pertes évitables…

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit
- 49 % d'efficacité !
- Mais l'*order fallback* occasionne des pertes évitables…

BITWISE PPM FLAT

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit
- 49 % d'efficacité !
- Mais l'*order fallback* occasionne des pertes évitables…

BITWISE PPM FLAT

- On retire seulement l'*order fallback* en utilisant à la place des probabilités neutres $(0.5 : 0.5)$

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit
- 49 % d'efficacité !
- Mais l'*order fallback* occasionne des pertes évitables…

BITWISE PPM FLAT

- On retire seulement l'*order fallback* en utilisant à la place des probabilités neutres $(0.5 : 0.5)$
- Performances améliorées : 63 % d'efficacité !

Modèles généraux

BITWISE PPM et BITWISE PPM FLAT

BITWISE PPM

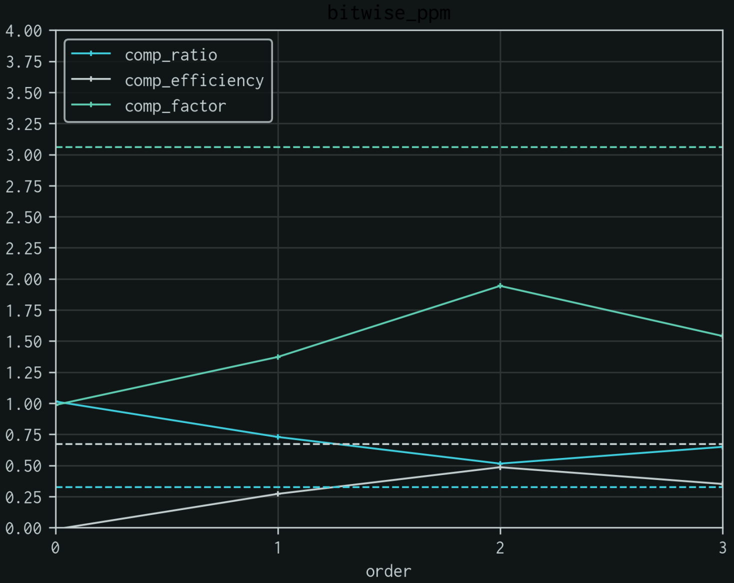
- Utiliser une PPM (d'ordre max. 3 octets) avec un encodage bit par bit
- 49 % d'efficacité !
- Mais l'*order fallback* occasionne des pertes évitables...

BITWISE PPM FLAT

- On retire seulement l'*order fallback* en utilisant à la place des probabilités neutres $(0.5 : 0.5)$
- Performances améliorées : 63 % d'efficacité !
- Simplement un *bitwise encoder* d'ordre fixe (0 - 28 bits)

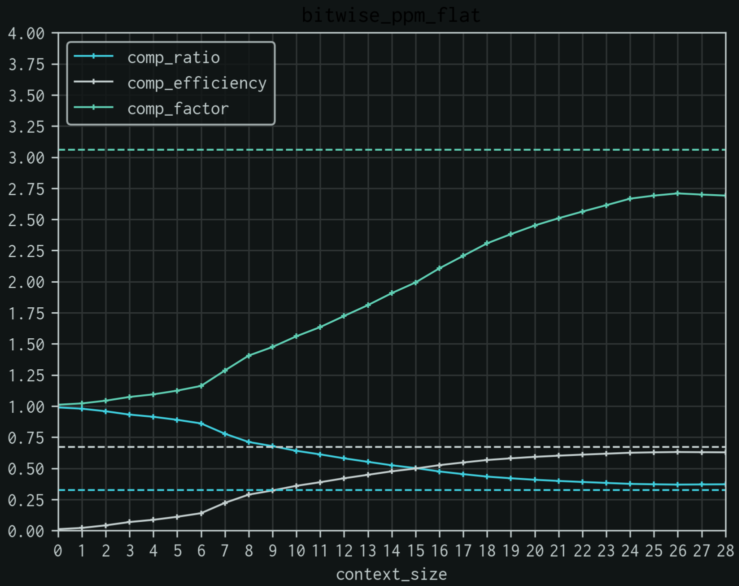
Modèles généraux

BITWISE PPM



Modèles généraux

BITWISE PPM FLAT



Impact de la transformée

BWT régularise nos données

BWT

Impact de la transformée

BWT régularise nos données

BWT

- Tranformation bijective

Impact de la transformée

BWT régularise nos données

BWT

- Tranformation bijective
- Tend à placer côte-à-côte les mêmes symboles

Impact de la transformée

BWT régularise nos données

BWT

- Tranformation bijective
- Tend à placer côte-à-côte les mêmes symboles

```
turlututu|              |turlututu
|turlututu              lututu|tur
u|turlutut              rlututu|tu
tu|turlutu              tu|turlutu
utu|turlut    sort      turlututu|
tutu|turlu  -------->   tutu|turlu
ututu|turl              u|turlutut
lututu|tur              urlututu|t
rlututu|tu              utu|turlut
urlututu|t              ututu|turl
```

Impact de la transformée

BWT régularise nos données

BWT

- Tranformation bijective
- Tend à placer côte-à-côte les mêmes symboles

```
turlututu|                  |turlututu
|turlututu                  lututu|tur
u|turlutut                  rlututu|tu
tu|turlutu                  tu|turlutu
utu|turlut      sort        turlututu|
tutu|turlu   ---------->    tutu|turlu
ututu|turl                  u|turlutut
lututu|tur                  urlututu|t
rlututu|tu                  utu|turlut
urlututu|t                  ututu|turl
```

- BWT(turlututu) $=$ uruulutttl

Impact de la transformée

BWT régularise nos données

BWT

- Tranformation bijective
- Tend à placer côte-à-côte les mêmes symboles

```
turlututu|
|turlututu
u|turlutut
tu|turlutu
utu|turlut
tutu|turlu
ututu|turl
lututu|tur
rlututu|tu
urlututu|t
```

$\xrightarrow{\text{sort}}$

```
|turlututu
lututu|tur
rlututu|tu
tu|turlutu
turlututu|
tutu|turlu
u|turlutut
urlututu|t
utu|turlut
ututu|turl
```

- $\textsc{bwt}(\texttt{turlututu}) = \texttt{uruulutttl}$
- Transforme C_m^0 en C^0

Impact de la transformée

La BWT en action

Avec la RLE

Impact de la transformée

La BWT en action

Avec la RLE

- Très simple : `uruuluttttl` devient 1`u` 1`r` 2`u` 1| 1`u` 3`t` 1`l`

Impact de la transformée

La BWT en action

Avec la RLE

- Très simple : `uruuluttttl` devient 1`u` 1`r` 2`u` 1| 1`u` 3`t` 1`l`
- Efficacité de 41 % !

Impact de la transformée

La BWT en action

Avec la RLE

- Très simple : `uruuluttl` devient 1`u` 1`r` 2`u` $1|$ 1`u` 3`t` 1`l`
- Efficacité de 41 % !

Avec BITWISE PPM FLAT

Impact de la transformée

La BWT en action

Avec la RLE

- Très simple : `uruuluttl` devient 1u 1r 2u 1| 1u 3t 1l
- Efficacité de 41 % !

Avec BITWISE PPM FLAT

- On gagne (seulement) 2 % d'efficacité supplémentaire

Impact de la transformée

La BWT en action

Avec la RLE

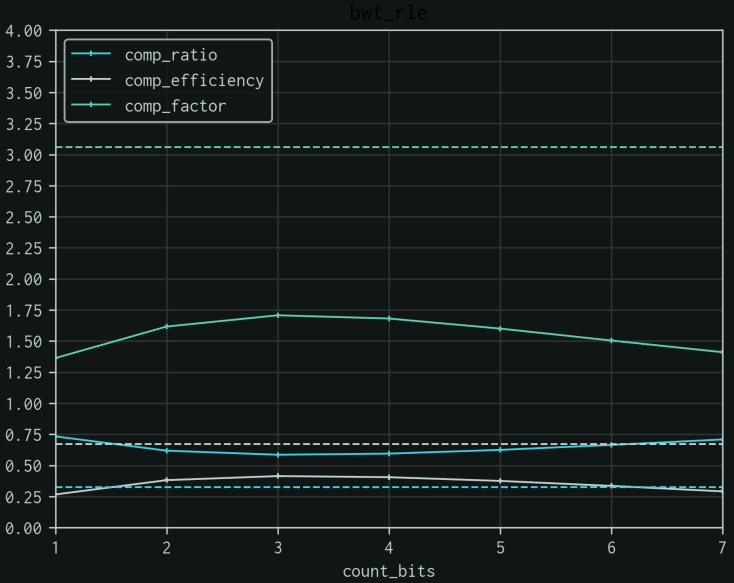
- Très simple : `uruuluttl` devient $1\mathtt{u}\,1\mathtt{r}\,2\mathtt{u}\,1|\,1\mathtt{u}\,3\mathtt{t}\,1\mathtt{l}$
- Efficacité de 41 % !

Avec BITWISE PPM FLAT

- On gagne (seulement) 2 % d'efficacité supplémentaire
- Permet d'atteindre mon record personnel :
  65 % d'efficacité
  facteur de compression **2,85x**
  35 % de la taille initiale

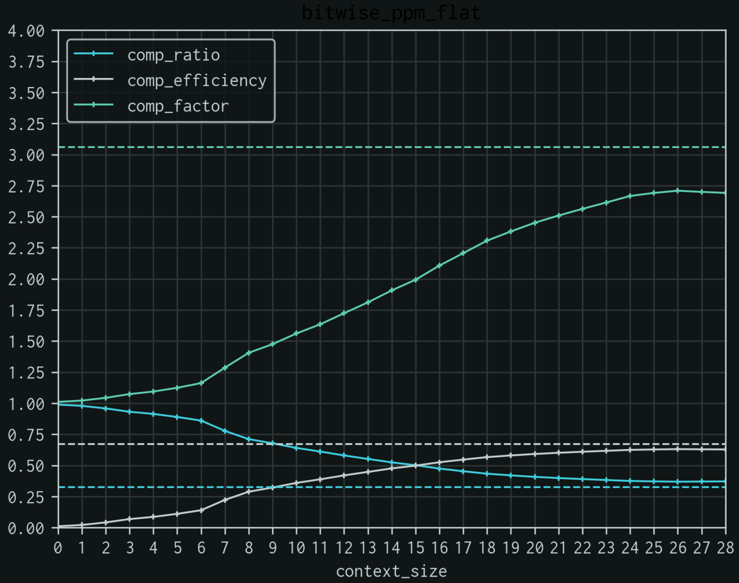
Impact de la transformée

BWT + RLE



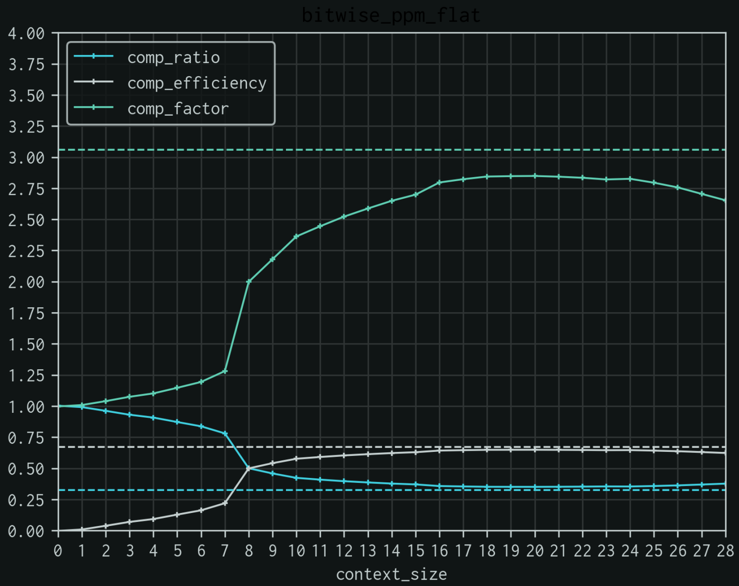
Impact de la transformée

BITWISE PPM FLAT seul (déjà vu)



Impact de la transformée

BWT + BITWISE PPM FLAT



Conclusion

Merci pour votre attention !

Sources principales

- *Data Compression Explained*, Matt MAHONEY
  mattmahoney.net/dc/dce.html
- *Suffix Array by Induced Sorting* (pour la BWT), G. NONGS. ZHANG, W. H. CHAN
  code.google.com/archive/p/ge-nong/

Tout le dossier disponible

- https://github.com/tbagrel1/tipe

Thomas BAGREL - Lycée Henri POINCARÉ, Nancy