

Développement d'une nouvelle technique de compression pour les codes variables à fixes quasi-instantanés

Danny Dubé
Fatma Haddad

Université Laval

24 Mars 2017

Plan

- 1 Introduction
 - Les concepts généraux
 - Les types de code
- 2 La technique de Tunstall
 - Présentation de la technique de Tunstall
 - L'algorithme de Tunstall
- 3 Les codes variables à fixes quasi-instantanés (AIVF)
 - Présentation des codes AIVF
 - L'algorithme de Yamamoto et Yokoo
 - Les deux modes de la technique de Yamamoto et Yokoo
- 4 Amélioration de l'encodage variable à fixe
 - Le problème de la racine complète en le mode multi-arbre
 - Le problème de l'exécution complète de l'option II
 - La technique par programmation dynamique

La compression des données

La compression de données : La compression de données est effectuée à l'aide de deux opérations :

- L'opération de compression, $\text{COMPRESS} : \Sigma_{\text{IN}}^* \rightarrow \Sigma_{\text{OUT}}^*$.
- L'opération de décompression, $\text{DECOMPRESS} : \Sigma_{\text{OUT}}^* \rightarrow \Sigma_{\text{IN}}^*$.

L'alphabet d'entrée, Σ_{IN} , et l'alphabet de sortie, Σ_{OUT} , peuvent être différents.

Cadre des travaux présentés

Cas particulier :

- Compression de données sans perte, i.e. :

$$\text{DECOMPRESS}(\text{COMPRESS}(w)) = w, \quad \text{pour tout } w \in \Sigma_{IN}^*.$$

Cas particulier :

- Encodage par sous-chaînes de gauche à droite :

$$\begin{array}{rcc}
 & w & (\in \Sigma_{IN}^*) \\
 = & & \\
 & u_1 \cdot u_2 \cdot \dots \cdot u_n & \\
 & \Downarrow C_1 \quad \Downarrow C_2 \quad \dots \quad \Downarrow C_n & \\
 & v_1 \cdot v_2 \cdot \dots \cdot v_n & \\
 = & \text{COMPRESS}(w) & (\in \Sigma_{OUT}^*)
 \end{array}$$

Cas particulier :

- Encodage statique (\equiv dictionnaires constants) :

$$\begin{array}{rcc}
 & w & (w \in \Sigma_{\text{IN}}^*) \\
 = & & \\
 & u_1 \cdot u_2 \cdot \dots \cdot u_n & (u_i \in D) \\
 & \Downarrow C \quad \Downarrow C \quad \dots \quad \Downarrow C & \\
 & v_1 \cdot v_2 \cdot \dots \cdot v_n & (v_i \in E) \\
 = & & \\
 & \text{COMPRESS}(w) & (v \in \Sigma_{\text{OUT}}^*)
 \end{array}$$

où

$D \subseteq \Sigma_{\text{IN}}^*$ est le *dictionnaire d'entrée*,

$E \subseteq \Sigma_{\text{OUT}}^*$ est le *dictionnaire de mots de code* et

$C : D \rightarrow E$ est définie sur tout D .

Cas particulier :

- Fonction d'encodage C *variable à fixe* (VF) :

$$|v| = l \in \mathbb{N}, \quad \text{pour tout } v \in E.$$

Définitions

Code : synonyme de dictionnaire.

Encodage (fonction d') : fonction d'un dictionnaire vers un autre (par exemple, $C : D \rightarrow E$).

Soit $D \subseteq \Sigma^*$ un code.

Code préfixe : D est un code *préfixe* si, pour tous les mots $u, v \in D$ tels que u est un préfixe de v , on a $u = v$.

Par exemple,

le code $\{00, 01, 1\}$ est préfixe mais

le code $\{0, 00, 01, 1\}$ n'est pas préfixe.

Définition alternative :

Code préfixe : D est un code *préfixe* si, pour tout $w \in \Sigma^\infty$, il existe au plus un mot $u \in D$ tel que u est un préfixe de w .

Soit $D \subseteq \Sigma^*$ un code.

Code exhaustif : D est un code *exhaustif* si, pour tout $w \in \Sigma^\infty$, il existe au moins un mot $u \in D$ tel que u est un préfixe de w .

Par exemple, pour $\Sigma = \{0, 1\}$
le code $\{1, 01, 00\}$ est exhaustif mais
le code $\{00, 10, 11\}$ n'est pas exhaustif.

Soient $D \subseteq \Sigma_{\text{IN}}^*$ et $E \subseteq \Sigma_{\text{OUT}}^*$ deux codes et $C : D \rightarrow E$ une fonction d'encodage.

Encodage valide : C est une fonction d'encodage *valide* si
 C est définie sur tout D ,
 C est injective,
 D est exhaustif et
 E est préfixe.

(Soient $D \subseteq \Sigma_{\text{IN}}^*$, $E \subseteq \Sigma_{\text{OUT}}^*$ et $C : D \rightarrow E$.)

La 1ère condition est nécessaire :
si C n'est pas définie sur tout D ,
alors on ne pourra pas tout encoder.

Par exemple :

$\Sigma_{\text{IN}} = \{a, b\}$, $D = \{aa, ab, ba, bb\}$, C pas définie sur ba et
 $w = aabaa \dots$

(Soient $D \subseteq \Sigma_{\text{IN}}^*$, $E \subseteq \Sigma_{\text{OUT}}^*$ et $C : D \rightarrow E$.)

La 2ème condition est nécessaire :

si C n'est pas injective

on ne pourra pas décoder correctement.

Par exemple :

$\Sigma_{\text{IN}} = \{a, b\}$, $\Sigma_{\text{OUT}} = \{0, 1\}$, $D = \{aa, ab, ba, bb\}$, $E = \{0, 10, 11\}$,

$C(aa) = 0$, $C(ab) = 0$, $C(ba) = 10$, $C(bb) = 11$,

$w = aabba\dots$ et $w' = abbba\dots$

(Soient $D \subseteq \Sigma_{\text{IN}}^*$, $E \subseteq \Sigma_{\text{OUT}}^*$ et $C : D \rightarrow E$.)

La 3ème condition est nécessaire :
si D n'est pas exhaustif,
alors on ne pourra pas tout encoder.

Par exemple :

$\Sigma_{\text{IN}} = \{a, b\}$, $D = \{a, aa, aaa\}$ et $w = aabaa \dots$

(Soient $D \subseteq \Sigma_{\text{IN}}^*$, $E \subseteq \Sigma_{\text{OUT}}^*$ et $C : D \rightarrow E$.)

La 4ème condition est nécessaire :
si E n'est pas préfixe,
on ne pourra pas décoder correctement.

Par exemple :

$\Sigma_{\text{IN}} = \{a, b\}$, $\Sigma_{\text{OUT}} = \{0, 1\}$, $D = \{aa, ab, ba, bb\}$,

$E = \{0, 1, 10, 11\}$,

$C(aa) = 0$, $C(ab) = 1$, $C(ba) = 10$, $C(bb) = 11$,

$w = abaa \dots$ et $w' = ba \dots$

Longueur moyenne des mots d'entrée / de code :

Soit $\Sigma_{IN} = \{a_1, \dots, a_n\}$ l'alphabet d'entrée,

soient $p(a_1), \dots, p(a_n)$ les probabilités des symboles et

soit $p(u)$ la probabilité d'un mot $u \in D$

(on suppose l'indépendance des symboles consécutifs entre eux).

Longueur moyenne des mots d'entrée :
$$\sum_{u \in D} p(u) * |u|$$

Longueur moyenne des mots de code :
$$\sum_{u \in D} p(u) * |C(u)|$$

Pour calculer la longueur moyenne, les probabilités utilisées sont **toujours** celle des mots d'entrée.

Les types de code

Soient Σ_{IN} et Σ_{OUT} , $D \subseteq \Sigma_{\text{IN}}^*$, $E \subseteq \Sigma_{\text{OUT}}^*$ et $C : D \rightarrow E$.

- **Encodage fixe à variable (FV)** : C est *FV* s'il existe $k \in \mathbb{N}$ tel que $|u| = k$, pour tout $u \in D$.
Il faut **minimiser** la longueur moyenne des mots de code.
- **Encodage variable à fixe (VF)** : C est *VF* s'il existe $l \in \mathbb{N}$ tel que $|v| = l$, pour tout $v \in E$.
Il faut **maximiser** la longueur moyenne des mots d'entrée.
- **Encodage fixe à fixe (FF)** : ...
- **Encodage variable à variable (VV)** : ...

Exemples des différents types de code.

- **Encodage fixe à variable (FV)** : Huffman.
- **Encodage variable à fixe (VF)** : Objet de ces travaux.
- **Encodage fixe à fixe (FF)** : Souvent utilisé pour détecter et corriger des erreurs (rarement en compression de données).
- **Encodage variable à variable (VV)** : LZ77, LZ78, LZW, LZSS.

Plan

- 1 Introduction
 - Les concepts généraux
 - Les types de code
- 2 La technique de Tunstall
 - Présentation de la technique de Tunstall
 - L'algorithme de Tunstall
- 3 Les codes variables à fixes quasi-instantanés (AIVF)
 - Présentation des codes AIVF
 - L'algorithme de Yamamoto et Yokoo
 - Les deux modes de la technique de Yamamoto et Yokoo
- 4 Amélioration de l'encodage variable à fixe
 - Le problème de la racine complète en le mode multi-arbre
 - Le problème de l'exécution complète de l'option II
 - La technique par programmation dynamique

Hypothèses de travail

Comme nous nous concentrons sur les encodages VF, nous laissons l'alphabet de sortie (Σ_{OUT}), les mots de code (E) et leur longueur (l) implicites.

Nous laissons aussi la fonction d'encodage C implicite.

Nous nous concentrons sur la conception de D , pour une certaine taille M désirée, et la longueur moyenne des mots d'entrée.

Dorénavant, nous appelons l'alphabet d'entrée Σ .

Nous supposons que les symboles d'entrée, $\{a_1, \dots, a_n\} = \Sigma$, sont triés par probabilité :

$$p(a_1) \geq \dots \geq p(a_n).$$

Hypothèses de travail

- Correspondance code \leftrightarrow arbre.
- Notation : $\#T$.
- Notation $T + \{w\}$.
- Notation $\pi(T)$.
- *Maximal-munch parsing*.

Présentation de la technique de Tunstall

En 1967, Brian Parker Tunstall a inventé un nouvel algorithme de compression.

- Appartient à la famille des codes variables à fixes (VF).
- Adopte le principe de dictionnaire pour construire son code.
- Contient un ensemble de mots, appelé *mots du dictionnaire* qui sont associés aux différents symboles sources.

⇒ Le nombre du mots de code dépend de la taille du dictionnaire.

Le dictionnaire de Tunstall respecte :

- Crée un code qui respecte la propriété préfixe.
- Crée un arbre complet, où chaque mot de code est assigné à une feuille de l'arbre.

L'algorithme de Tunstall

Algorithm 1 : Algorithme de Tunstall avec M mots de code

Require: $M \geq |\Sigma|$

1: $T_{\text{new}} \leftarrow \text{racine}$

2: **repeat**

3: $T_{\text{old}} \leftarrow T_{\text{new}}$

4: $T_{\text{new}} \leftarrow \text{Option I} (T_{\text{old}})$

5: **until** $\#T_{\text{new}} > M$

6: **return** T_{old}

Algorithm 2 : Option I

Require: T un arbre représentant D

1: $V \leftarrow$ noeuds incomplets dans T

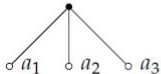
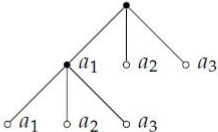
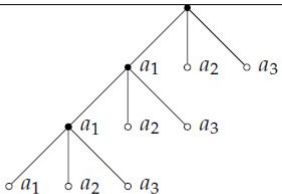
2: $n_{\max} \leftarrow \arg \max_{n \in V} p(n)$

3: $W \leftarrow \{\pi(n_{\max}) \cdot a \mid a \in \Sigma\}$

4: **return** $T + W$

Exemple

Soit $\Sigma = \{a_1, a_2, a_3\}$ où $p(a_1) = 0.6$, $p(a_2) = 0.3$ et $p(a_3) = 0.1$. Soit $M = 7$.

Initialisation	Itération 1	Itération 2
		
$L^3 = 1$	$L^5 = 1.6$	$L^7 = 1.96$

Mot du dictionnaire	Probabilité	Mot de code
$a_1 a_1 a_1$	0.216	A
$a_1 a_1 a_2$	0.108	B
$a_1 a_1 a_3$	0.036	C
$a_1 a_2$	0.18	D
$a_1 a_3$	0.06	E
a_2	0.3	F
a_3	0.1	G

Plan

- 1 Introduction
 - Les concepts généraux
 - Les types de code
- 2 La technique de Tunstall
 - Présentation de la technique de Tunstall
 - L'algorithme de Tunstall
- 3 Les codes variables à fixes quasi-instantanés (AIVF)
 - Présentation des codes AIVF
 - L'algorithme de Yamamoto et Yokoo
 - Les deux modes de la technique de Yamamoto et Yokoo
- 4 Amélioration de l'encodage variable à fixe
 - Le problème de la racine complète en le mode multi-arbre
 - Le problème de l'exécution complète de l'option II
 - La technique par programmation dynamique

Les codes VF quasi-instantanés

En 2001, Hirosuke Yamamoto et Hidetoshi Yokoo ont présenté un nouvel encodage VF basé sur les *codes quasi-instantanés* (AIVF).

- Le dictionnaire D est un code quasi-instantané.
- Un code quasi-instantané est complet mais n'est pas nécessairement préfixe.
- L'arbre correspondant au code peut avoir des noeuds internes incomplets.
- Les mots de code sont assignés aux noeuds incomplets (i.e. les feuilles et les noeuds internes incomplets).

⇒ Dans certaines situations, les codes AIVF surpassent les codes VF préfixes de Tunstall.

Les codes VF quasi-instantanés

La technique de Yamamoto et Yokoo repose sur deux stratégies :

- La première stratégie, appelée option I, est la même que celle de la technique de Tunstall.
- La deuxième stratégie, appelée option II, propose *un* prochain noeud : celui qui est le plus probable parmi tous les noeuds pas encore créés.

L'algorithme de Yamamoto et Yokoo

Algorithm 3 : Option II

Require: T un arbre représentant D

- 1: $U \leftarrow \{ \pi(n) \mid \text{noeud } n \text{ dans } T \}$
 - 2: $V \leftarrow U \cdot \Sigma$
 - 3: $W \leftarrow V - U$
 - 4: $w_{\max} \leftarrow \arg \max_{w \in W} p(w)$
 - 5: **return** $T + \{w_{\max}\}$
-

L'algorithme principal fait croître un arbre en faisant rivaliser les propositions faites par les deux options.

Algorithm 4 : Algorithme de Y&Y avec M mots de code et information partielle sur le prochain symbole ($\notin \{a_1, \dots, a_i\}$)

Require: i où $0 \leq i \leq |\Sigma| - 2$

Require: $M \geq |\Sigma| - i$

- 1: $T_{\text{new}} \leftarrow \text{racine} + \{a_{i+1}, \dots, a_{|\Sigma|}\}$
 - 2: **repeat**
 - 3: $T_{\text{old}} \leftarrow T_{\text{new}}$
 - 4: $T_{\text{I}} \leftarrow \text{Option I} (T_{\text{old}})$
 - 5: $T_{\text{II}} \leftarrow \text{Option II}^{\#T_{\text{I}} - \#T_{\text{old}}} (T_{\text{old}})$
 - 6: $T_{\text{new}} \leftarrow$ le meilleur arbre entre T_{I} et T_{II}
 - 7: **until** $\#T_{\text{new}} > M$
 - 8: **return** $\text{Option II}^{M - \#T_{\text{old}}} (T_{\text{old}})$
-

Les deux modes de la technique de Yamamoto et Yokoo

- **Mode mono-arbre** : Un arbre nommé T_0 utilisé en tout temps. L'arbre présume ne disposer d'aucune information sur l'entrée.
- **Mode multi-arbre** : Une famille d'arbres T_i spécialisés selon l'information détenue sur le prochain symbole de l'entrée, pour $0 \leq i \leq |\Sigma| - 2$. L'arbre T_i présume que le prochain symbole de l'entrée ne peut pas être a_1, a_2, \dots, a_i . Cette connaissance se traduit par l'interdiction de certaines branches à partir de la racine.

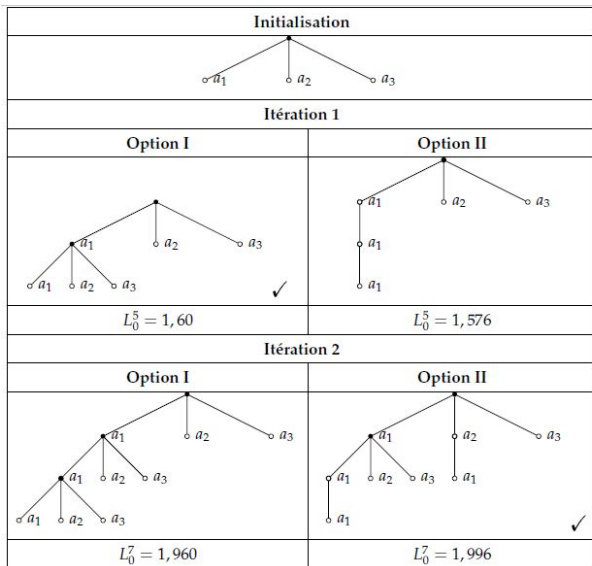
Exemple en mode mono-arbre

Soit $\Sigma = \{a_1, a_2, a_3\}$ où $p(a_1) = 0.6$, $p(a_2) = 0.3$ et $p(a_3) = 0.1$.

Soit $M = 7$.

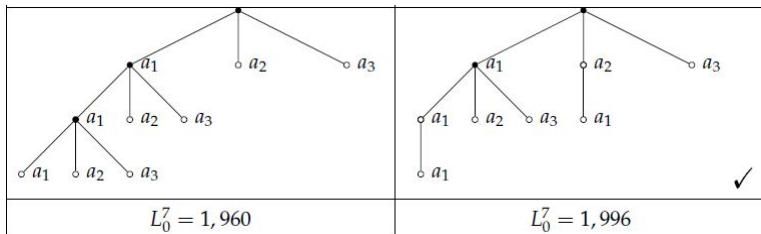
(Même situation que dans l'exemple pour l'algorithme de Tunstall.)

On ne crée que T_0 .



Comparaison entre le mode mono-arbre et l'arbre de Tunstall

L'arbre T_0 est différent de l'arbre de Tunstall. Nous remarquons que T_0 a la meilleure longueur moyenne $L_0^7 = 1.996$ comparativement à l'arbre de Tunstall, qui a $L^7 = 1.960$.



Exemple en mode multi-arbre

Soit $\Sigma = \{a_1, a_2, a_3\}$ où $p(a_1) = 0.6$, $p(a_2) = 0.3$ et $p(a_3) = 0.1$.

Soit $M = 7$.

T_0 est construit comme en mode mono-arbre.

Il reste à construire T_1 .

Initialisation	
Itération 1	
Option I	Option II
$L_1^4 = 1.75$	$L_1^4 = 1.72$

Itération 2	
Option I	Option II
$L_1^6 = 2.2$	$L_1^6 = 2.182$
Itération 3	
Option I	Option II
Bloquée	
	$L_1^7 = 2.362$

Plan

- 1 Introduction
 - Les concepts généraux
 - Les types de code
- 2 La technique de Tunstall
 - Présentation de la technique de Tunstall
 - L'algorithme de Tunstall
- 3 Les codes variables à fixes quasi-instantanés (AIVF)
 - Présentation des codes AIVF
 - L'algorithme de Yamamoto et Yokoo
 - Les deux modes de la technique de Yamamoto et Yokoo
- 4 Amélioration de l'encodage variable à fixe
 - Le problème de la racine complète en le mode multi-arbre
 - Le problème de l'exécution complète de l'option II
 - La technique par programmation dynamique

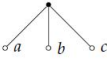
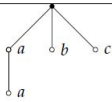


Le problème de la racine complète en le mode multi-arbre

Problème : La technique de Yamamoto et Yokoo exige la création d'une racine complète lors de l'initialisation.

Solution : Enlever l'obligation d'utiliser l'**option I** pour créer les enfants de la racine.

Exemple montrant le problème

Soit $\Sigma = \{a, b, c\}$ où $p(a) = 0.7$, $p(b) = 0.2$ et $p(c) = 0.1$. Soit $M = 4$. (**Erreur dans l'image !**)

	État initial	Itération 1	Itération 2
YY	•		
	$L_0^1 = 0$	$L_0^3 = 0.99$	$L_0^4 = 1.49$
DH	•		
	$L_0^1 = 0$	$L_0^3 = 1.19$	$L_0^4 = 1.53$

Le problème de l'exécution complète de l'option II

Problème : L'adoption de la proposition faite suite à plusieurs utilisations de l'option II est sous-optimale. (Rappel : l'option II est exécuté autant de fois qu'il y a de mots de code consommés par l'option I.)

Solution : Notre solution consiste à ne pas exécuter la proposition complète de l'option II. Nous ajoutons seulement la branche la plus probable.

Exemple montrant le problème

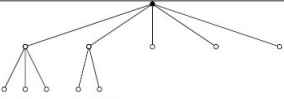
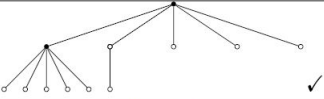
Nous construisons T_0 en mode mono-arbre.

Soit $\Sigma = \{a, b, c, d, e\}$ où

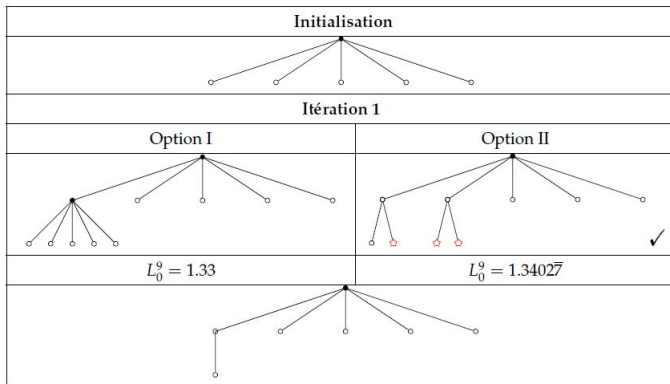
$$p(a) = \frac{1}{3}, p(b) = \frac{1}{4}, p(c) = \frac{1}{6}, p(d) = \frac{3}{20}, p(e) = \frac{1}{10}.$$

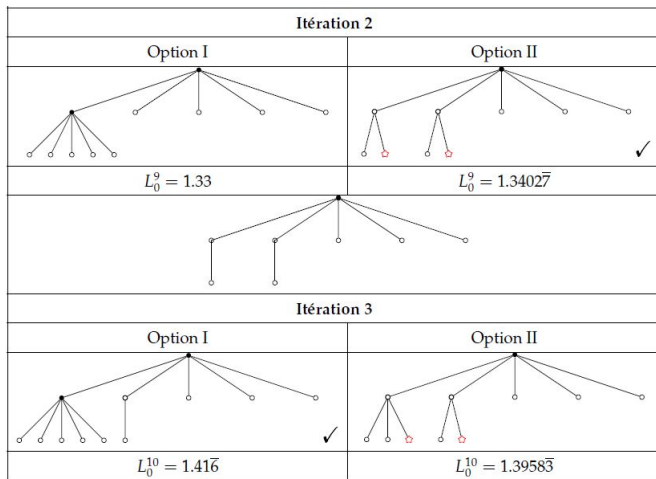
Soit $M = 10$.

Résultats avec/sans exécution complète :

Solution YY	Solution DH
	
$L_0^{10} = 1.3958\bar{3}$	$L_0^{10} = 1.41\bar{6}$

Trace de la construction avec adoption partielle de la proposition de l'option II.





La technique par programmation dynamique

Nous présentons une nouvelle technique pour la construction des codes AIVF en mode multi-arbre.

Cette technique est basée sur la programmation dynamique.

Elle construit des codes AIVF optimaux.

L'algorithme par programmation dynamique

Algorithm 5 : Programmation dynamique

Require: $M \geq 1$

- 1: **for** $N = 1$ **to** M **do**
 - 2: **for** $i = 0$ **to** $|\Sigma| - 1$ **do**
 - 3: $T_i^N \leftarrow$ Création de l'arbre (N, i)
 - 4: **end for**
 - 5: **end for**
-

L'algorithme 6 considère trois cas.

Le cas de base :

- $i \leq |\Sigma| - 2$ et $N = 1$.

Le prochain symbole peut être l'un de $a_{i+1}, \dots, a_{|\Sigma|}$.

Il n'est pas connu avec certitude.

Donc, T_i^1 est :

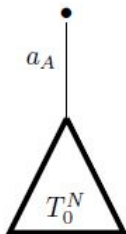
○

Le cas récursif trivial :

- $i = |\Sigma| - 1$.

Le prochain symbole ne peut être que $a_{|\Sigma|}$.

Donc, $T_{|\Sigma|-1}^N$ est :



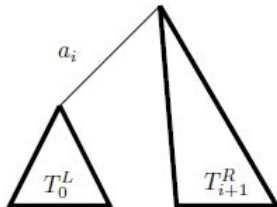
Le cas récursif non-trivial :

- $i \leq |\Sigma| - 2$ et $N \geq 2$.

Le prochain symbole peut être l'un de $a_{i+1}, \dots, a_{|\Sigma|}$.

Il y a plusieurs façons de construire T_i^N .

Néanmoins, T_i^N doit avoir la forme : **(Erreur dans l'image !)**




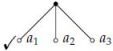
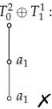

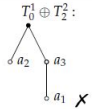
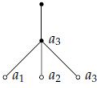


dénotée $T_0^L \oplus T_{i+1}^R$, où $L + R = N$ et T_0^L et T_{i+1}^R sont optimaux.

Exemple de construction par programmation dynamique

Soit $\Sigma = \{a_1, a_2, a_3\}$ où $p(a_1) = 0.6$, $p(a_2) = 0.3$ et $p(a_3) = 0.1$. Soit $M = 3$.

	$i = 0$	$i = 1$	$i = 2$
$N = 1$	T_0^1	T_1^1	T_2^1
	$L_0^1 = 0$	$L_1^1 = 0$	$L_2^1 = 1$
$N = 2$	T_0^2	T_1^2	T_2^2
	$T_0^1 \oplus T_1^1$: 	$T_0^1 \oplus T_2^1$: 	
	$L_0^2 = 0.6$	$L_1^2 = 1$	$L_2^2 = 1.6$
$N = 3$	T_0^3	T_1^3	T_2^3
	$T_0^1 \oplus T_1^2$: 	$T_0^2 \oplus T_1^1$: 	$T_0^2 \oplus T_2^1$:
	$L_0^3 = 1$ ou $L_0^3 = 0.96$	$L_1^3 = 1.45$ ou $L_1^3 = 1.15$	$L_2^3 = 2$

	T_0^3	T_1^3	T_2^3
YY			
	$L_0^3 = 1$	$L_1^3 = 1.45$	$L_2^3 = 2$
DH	$T_0^1 \oplus T_1^2$:  ✓ $T_0^2 \oplus T_1^1$:  ✗	$T_0^2 \oplus T_1^2$:  ✓ $T_0^1 \oplus T_2^2$:  ✗	
	$L_0^3 = 1$ ou $L_0^3 = 0.96$	$L_1^3 = 1.45$ ou $L_1^3 = 1.15$	$L_2^3 = 2$

Conclusion

Contributions :

- Identification de deux défauts dans la technique de Y&Y.
- Proposition de correctifs aux deux défauts.
- Proposition d'un algorithme par programmation dynamique.

Questions restantes :

- Technique de Y&Y + deux correctifs = technique optimale ?
- Véritable définition d'optimalité en mode multi-arbre.

Merci !

Questions ?

Références I

- [1] Mitsuharu Arimura and Ken-ichi Iwata.
The minimum achievable redundancy rate of fixed-to-fixed length source codes for general sources.
In Information Theory and its Applications (ISITA), 2010 International Symposium on, pages 595–600. IEEE, 2010.
- [2] David A Huffman et al.
A method for the construction of minimum-redundancy codes.
Proceedings of the IRE, 40(9) :1098–1101, 1952.
- [3] Brian Parker Tunstall.
Synthesis of Noiseless Compression Codes.
Ph.D. dissertation, Georgia Institute of Technology, Atlanta, GA, USA, September 1967.

Références II

- [4] Terry A. Welch.
A technique for high-performance data compression.
Computer, 17(6) :8–19, 1984.
- [5] Hirosuke Yamamoto and Hidetoshi Yokoo.
Average-sense optimality and competitive optimality for almost instantaneous VF codes.
Les IEEE Transactions on Information Theory, 47 :2174–2184, 2001.
- [6] Jacob Ziv and Abraham Lempel.
A universal algorithm for sequential data compression.
IEEE Transactions on information theory, 23(3) :337–343, 1977.

Références III

- [7] Jacob Ziv and Abraham Lempel.
Compression of individual sequences via variable-rate coding.
IEEE transactions on Information Theory, 24(5) :530–536,
1978.