INF-8770 - Techniques Multimédia

Automne 2018

Travail Pratique 1
Comparaisons des encodeurs LZ-77 et Arithmétique

Groupe 1

1795724 - Alexandre Hua
1788028 - Francois Toulouse

[2018 / 9 / 21]

Question 1

Voici les hypothèses sur les encodages LZ-77 et Arithmétique que nous allons tester dans ce travail pratique:

- **Hypothèse 1: Pour des messages contenant des séquences de symboles répétées, la taille du message encodé sera plus court avec l'encodage LZ-77.**
  La répétition pourra permettre à l'encodage par dictionnaire de simplifier des séquences entières, contrairement à l'encodeur Arithmétique qui compresse caractère par caractère.

- **Hypothèse 2: Pour des messages plus longs, le temps de compression sera plus court avec l'encodage LZ-77.**
  Pour les messages de plus grande tailles, l'encodage Arithmétique deviendra significativement plus lent à compléter. Il serait donc possible de voir une meilleure performance de l'algorithme LZ-77 relativement à celle de l'algorithme Arithmétique.

- **Hypothèse 3: Pour des messages avec beaucoup de symboles différents, le temps d'encodage sera plus long avec l'encodage Arithmétique.**
  Un message complexe réduit les plages de probabilité pour chaque symbole additionnel, ce qui augmente davantage la précision requise par l'algorithme et le nombre d'opération arithmétique pour y arriver.

Question 2

Expériences:

Nous allons procéder à une série d'expérience consistant à vérifier l'influence des différents paramètres sur l'efficacité et la performance de l'encodage de message pour chaque algorithme. On génère une centaines de messages à compresser, et pour chaque cas, une combinaison de paramètre aléatoire est choisie par le générateur. Les paramètres à varier sont les suivants:

- La complexité du message (nombre de symboles différents),
- La longueur du message (en caractères)
- La probabilité de répétition de chaînes de symboles.

Nous avons également considérer la possibilité de faire varier la taille du dictionnaire pour contrôler le nombre de caractère que l'encodeur LZ-77 puisse se souvenir. Cependant, comme la taille du dictionnaire dépend de la capacité de mémoire du programme d'encodage et non du message original lui-même, nous pouvons juste assigner une taille de sorte que

l'encodeur puisse retenir tout le message au complet afin de pouvoir comparer avec la méthode LZ-77 dans ses meilleures conditions.

Les données seront ensuite recueillies et analysées dans un graphe afin de mieux voir l'effet de variation de chacun de ces paramètres.

Base de données

Avant de pouvoir commencer à tester les méthodes d'encodage, nous avons d'abord besoin d'une base de donnée de message avec laquelle nous pouvons utiliser pour encoder et nous avons même besoin d'une grande quantité afin d'avoir une plus grande variété de données en entrée et de pouvoir couvrir le plus de cas possible. Nous avons donc choisi d'implémenter notre propre générateur aléatoire de message afin de pouvoir générer facilement autant. Le contenu du message s'inspire des paramètres mentionnés dans la partie plus haute.

Pour la longueur du message, nous avons choisi de la faire varier entre 2 et 50. Nous avons omis les messages de longueur de 1 car il est impossible d'encoder un message contenant seulement un symbol unique. Nous avons également choisi de limiter la longueur à une valeur maximale de 50, car nous pensons que c'est une longueur relativement raisonnable.

Pour le nombre de symbole, nous avons décidé que le message pourrait contenir entre 2 et 26 symboles différents. Comme expliqué plus haut, nous avons choisi une valeur minimale de 2 symboles, car un message doit avoir au moins 2 caractères différents pour être encodés. De plus, pour cette expérience, nous avons voulu simplifier un peu en utilisant seulement les 26 lettres de l'alphabet en tant que symbole pour notre message.

Afin d'avoir des cas plus intéressant pour la méthode de LZ-77, nous avons mis un facteur de répétition pour contrôler la fréquence de récurrence d'une même séquence. Nous l'avons fait varier entre les valeurs 0 et 0.8. Nous avons décidé de limiter le maximum à 80% afin de ne pas trop exagérer la répétition des séquences.

Par ailleurs, nos messages générés sont des chaînes de caractère non espacées avec aucun caractère spécial et aucun chiffre.

Critères d'évaluation:

Les critères d'évaluation utilisés pour mesurer les prédictions des hypothèses sont les suivants:

- **Taille du message encodé** (en bits)
  Ceci permet d'évaluer l'efficacité de compression des algorithmes. En effet, le but d'une compression de donnée est de réduire la taille du contenu originale, donc plus le code est court, plus l'algorithme est meilleur et efficace.
- **Temps d'encodage** (en millisecondes)
  Ceci permet de quantifier et de comparer la vitesse des algorithmes. Le temps de

compression a une place importante, car une méthode rapide est toujours mieux qu'une même méthode plus lente.

Code utilisé:

Pour effectuer notre expérience, nous nous sommes servis des codes provenant de sources externes, mais également des codes implémentés par nous-mêmes. Notre programme consiste d'un composant principal, d'un composant d'encodage arithmétique, d'un composant d'encodage LZ-77 et d'un générateur de message.

Tout d'abord, le programme principal qui a été implémenté par nous-mêmes est le composant qui fait le lien avec les 3 autres composants. Il gère une interface utilisateur qui permet de recevoir en entrée plusieurs paramètres pour contrôler les cas de tests générés. Il est également responsable pour appeler les fonctions d'encodage pour chacune des méthodes.

Ensuite, l'algorithme d'encodage arithmétique que nous avons utilisé pour cette expérience est une version adaptée à partir de l'exemple disponible sur le répertoire Github du cours:
https://github.com/gabilodeau/INF8770/blob/master/Codage%20arithm%C3%A9tique.ipynb

Seul l'algorithme LZ-W était disponible dans le répertoire Github de INF-8770, donc nous avons dû adapter le code du répertoire d'un tiers parti disponible à l'adresse suivante:
https://github.com/manassra/LZ77-Compressor/blob/master/src/LZ77.py

Enfin, le générateur de message qui a été également implémenté par nous fournit des messages aléatoires à encoder. Celui-ci est appelé avant chaque méthode d'encodage.

Question 3

Description et fonctionnement du code

Si des modules sont manquants lors de l'exécution du code, il est possible d'installer les dépendances avec la commande décrite dans le fichier READ_ME.txt.

Programme principal

Le programme principal représente le coeur de notre application. Celui-ci est responsable à gérer une interface utilisateur qui permet de recevoir des paramètres de l'utilisateur par la ligne de commande. Il les passe ensuite au générateur de message afin d'obtenir un message aléatoire correspondant au caractéristique du besoin de l'utilisateur. Respectivement, ce message est envoyé aux méthodes d'encodage arithmétique et LZ-77 afin d'obtenir le message codé de chacun. Ces étapes sont répétés jusqu'à la production du nombre de cas de test désiré. Ensuite, le programme principal sauvegarde les données recueillis au cours de l'expérience dans un fichier Excel. Ces données en sortie sont le code compressé, sa taille, le ratio de compression et la durée du traitement pour chacune des méthodes.

Les paramètres disponibles à l'utilisateur sont:

- **-n**, (entier) pour spécifier le nombre de cas de test à générer,
- **-l**, (entier) pour spécifier la longueur du message à générer,
- **-s**, (entier) pour spécifier le nombre exact de symbole à avoir dans le message,
- **-r**, (réel) pour spécifier le taux de répétition, qui correspond à la probabilité que le générateur de message recule et copie des groupes de symboles précédents lors du processus de génération.
- **-w**, (entier) pour spécifier la taille du dictionnaire pour l'encodeur LZ-77

Il est aussi possible d'utiliser le paramètre **-m** (chaîne) et directement fournir un message spécifique à encoder et à analyser.

Dans le cas où un paramètre n'est pas spécifié, une valeur aléatoire est générée pour celui-ci selon l'intervalle de valeur définie plus haut.

Générateur de message

La responsabilité du générateur de message est de fournir un message généré de façon aléatoire au programme principal. La procédure pour générer un message utilisé par ce générateur est de simplement générer aléatoirement chaque caractère une à la fois pour à la fin les assembler et la longueur du message représente le nombre de caractère à créer. Cependant, nous avons rencontré une difficulté lors de l'implémentation du code qui était la suivante. Le nombre de symbole reçu en paramètre représente le nombre de symbole disponible lors du processus de sélection aléatoire, ainsi que le nombre exact de symbole unique qu'on désire retrouver dans le message. Le problème est qu'il est possible qu'il ait des symboles qui ne sont pas utilisés du tout bien qu'ils soient disponibles. Nous devons donc trouver un moyen pour s'assurer le message contient toujours exactement le même nombre de symbole unique tel que spécifié. Notre solution était de mettre une étape de vérification de la validité du message à la fin de la génération. Si celui-ci n'est pas valide, on recommence le processus.

Programme d'encodage arithmétique

L'algorithme commence par compter et calculer la fréquence et le taux d'occurrence, respectivement, des différents symboles à encoder dans le message fourni. Ensuite, pour chaque symbole du message original, ce taux est utilisé pour restreindre l'intervalle. Une valeur float est ensuite générée pour encoder cet intervalle avec le moins de bits possible. Cette valeur est encodée sous format binaire et sa taille est récupérée pour les comparaisons futures.

Programme d'encodage LZ-77

Cet algorithme transforme le message en bytes par encodage UTF-8, puis itère sur le message d'entrée pour trouver des séquences de caractères similaires avec une taille de dictionnaire déterminée à l'avance. Si une séquence est trouvée, elle est enregistrée sur 17 bits ("1" pour indiquer une séquence à copier, 8 bits pour la distance, 8 bits pour la quantité à copier). Si aucune séquence n'est trouvée, un nouveau symbole est encodé sur 9 bits ("0" pour indiquer un symbole, puis 8 bits pour le symbole). Le code final est récupéré sous format binaire pour comparaison.

Différemment de ce que nous avons vu dans le cours, la façon d'encodage de l'auteur de ce code est légèrement différente. En effet, au lieu d'encoder juste des triplets, cet algorithme procède de cette façon:

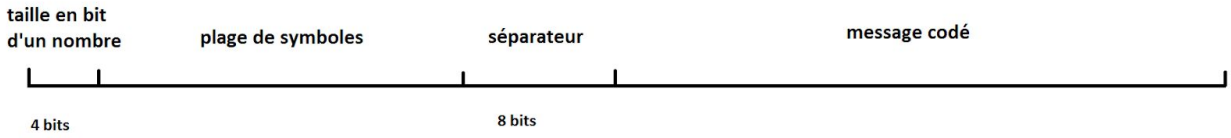
- **Encodage d'un doublet <0, lettre>:** Dans le cas où la séquence lue ne se trouve pas dans le dictionnaire.
- **Encodage d'un triplet <1, début, longueur>:** Dans le cas où la séquence lue existe dans le dictionnaire

Exceptionnellement, premier caractère 0 d'un doublet et le caractère 1 d'un triplet est encodé sur 1 bit, car il ne peut y avoir que ces deux valeurs. Celle-ci permet d'indiquer si ce qui suit est un doublet ou un triplet.

Adaptation du code:

Code d'encodage arithmétique

Pour la plupart, nous avons gardé le code tel qu'il était. Par contre, nous avons modifié un peu le code au niveau de l'affichage des résultats. En effet, le code original affichait le message codé sous un format abrégé, mais nous avons voulu l'afficher au complet de façon plus explicite. Nous avons également ajouté du contenu supplémentaire dans le message codé final. En effet, le code original affichait seulement le message codé, mais il ne faut pas oublier qu'en plus du message codé, la plage des symboles uniques est également un élément important afin de pouvoir décoder le message, ainsi que d'autres subtilités. Entre autres, voici le contenu du message codé final de notre code adapté:



- **Taille en bit d'un nombre**: cette valeur représente la taille en bits qui est accordée pour encoder un nombre entier. En effet, dans la plage de symboles, il faut encoder le nombre d'occurrence pour chaque symbole en nombre entier. La raison pourquoi nous avons choisi d'ajouter cette information dans le message codé est le fait que la taille en bits d'un nombre est variable selon la taille du message. En effet, dans le pire cas, si nous avons un message d'une longueur de 50, il se peut que le message ait une occurrence de 49 fois de la lettre "A" et 1 fois de la lettre "B", ce qui signifie que nous devons être capable d'encoder les nombres allant de 1 à 49. Pour cela, nous avons besoin d'au plus 6 bits et pour représenter le chiffre 6, nous avons besoin de 4 bits qui est le chiffre que nous avons sur l'image. La taille en bit d'un nombre aurait pu être encore plus grande, mais comme nous nous sommes fixés sur une longueur maximale de 50, nous avons besoin juste de 4 bits.

  Une autre raison est pour avantager la méthode arithmétique par rapport à la méthode binaire qui permet de trouver la longueur originale du message codé. En effet,

il est possible de remarquer que la méthode binaire n'a pas de nombre entier à encoder, tandis que la méthode arithmétique en a besoin. Nous aurons pu déterminer un nombre de bits standard plus coûteux à accorder pour un nombre entier comme nous l'avons fait pour l'encodage d'une lettre, mais cela risque d'affaiblir la performance de la méthode arithmétique par rapport à la méthode binaire.

- **Plage de symboles:** cette plage correspond au nombre d'apparition de chaque symbole. Il s'agit d'un élément nécessaire pour pouvoir décompresser le message codé. Pour chaque symbole unique, on encode d'abord la lettre en 8 bits suivi par son nombre d'occurrence en une taille en bit défini par les 4 premiers bits expliqués plus haut.

- **Séparateur:** il s'agit d'un caractère spécial utilisé pour signaler la fin de la plage de symboles et le début du message codé. Pour ce cas-ci, nous avons choisi "1111 1111" comme caractère spécial. Remarquez que nous n'avons pas mis de séparateur entre la taille en bits d'un nombre et la plage de symboles, car nous savons que la plage de symboles commence toujours à partir du 5e bit.

- **Message codé:** dans le cas de la méthode arithmétique, le message codé représente la valeur décimale en binaire qui sert pour décompresser le message.

- **Choix du 8 bits:** la raison pour choisir d'encoder une lettre en 8 bits plutôt qu'une valeur plus optimale comme pour l'encodage d'un nombre est le fait que c'est une taille standard. De plus, ca n'affecte pas beaucoup le taux de compression, car la longueur du message compressé reste proportionnelle par rapport à la méthode binaire qui l'utilise également.

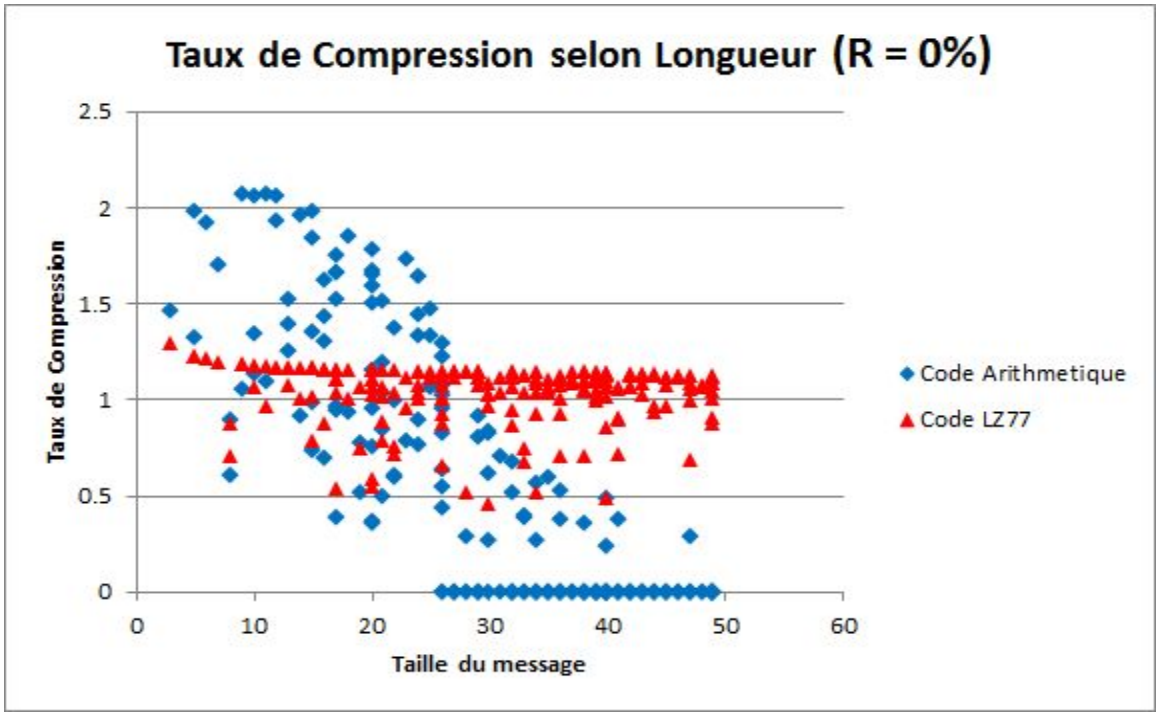
Code d'encodage LZ-77

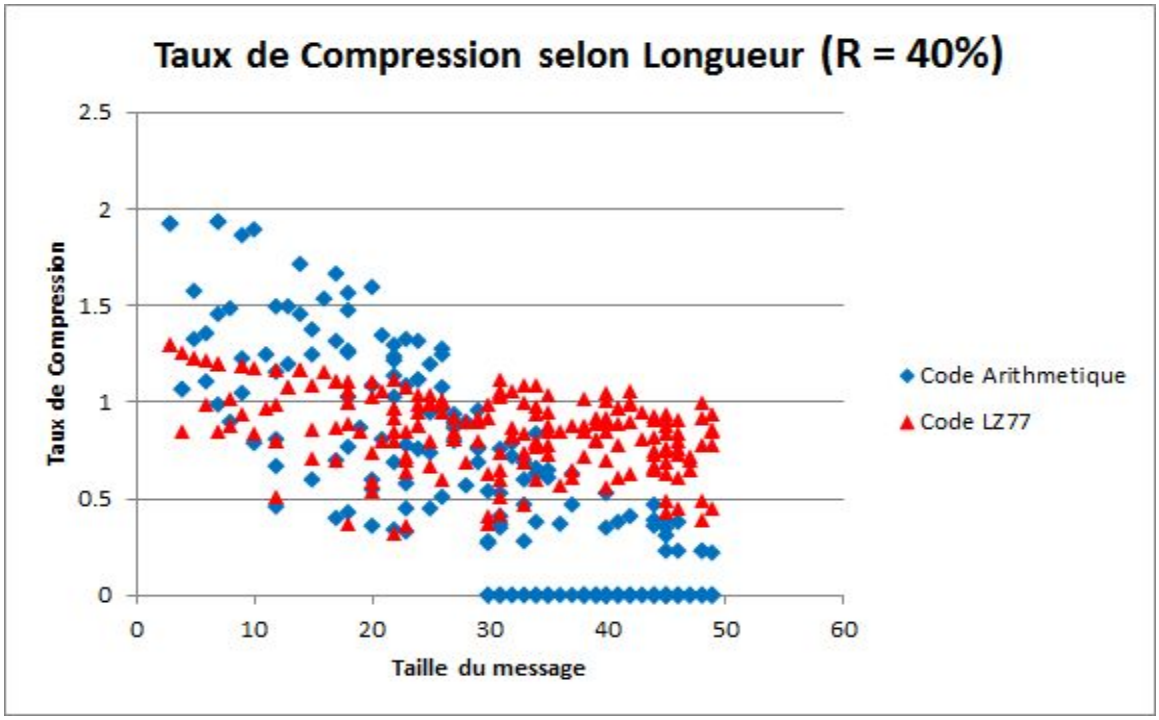
Pour ce code, nous avons changé plusieurs aspects afin d'adapter le code pour nos expériences. Tout d'abord, nous avons modifié la façon dont le programme reçoit les données en entré. En effet, le code original lisait un message à partir d'un fichier texte, donc le message en entré était de type binaire, tandis que le message que nous passons à l'encodeur était de type String. Nous avons dû effacer le code en lien avec la lecture par fichier de texte et ajouter une étape pour convertir notre message de type String en type binaire. Ensuite, nous avons également modifié la façon dont l'application produit les données en sortie. En effet, le code original écrit le message codé dans un fichier de texte en sortie, tandis que ce que nous désirons était de simplement retourner le message. De plus, le code original encodait le message en utilisant des bitarray, donc le message codé est aussi du même type. Nous avons donc dû entrer assez profondément dans le code pour adapter le code de façon à ce qu'il puisse supporter la sortie du message codé en String.

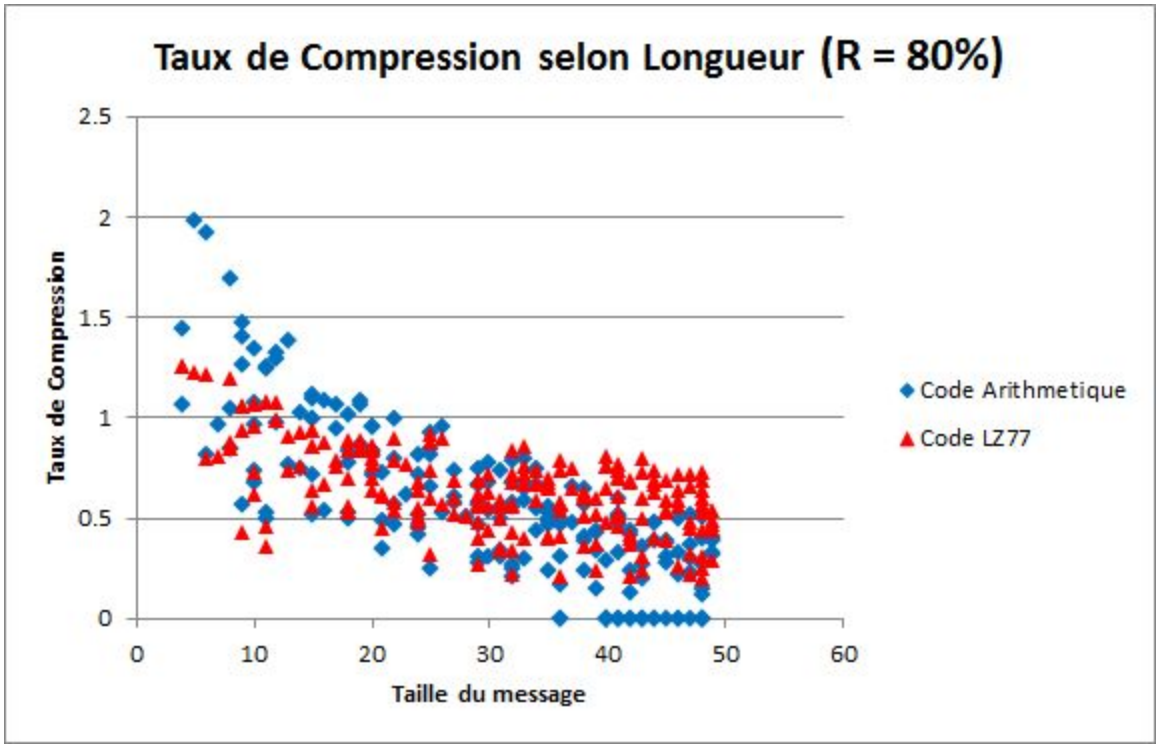
Tout comme pour la méthode arithmétique, notre encodeur LZ-77 encode les lettres sur 8 bits et les nombres sur la taille indiquée par les 4 premiers bits. Ce qui suit ces 4 bits est les données en doublet et triplet.

Résultats:

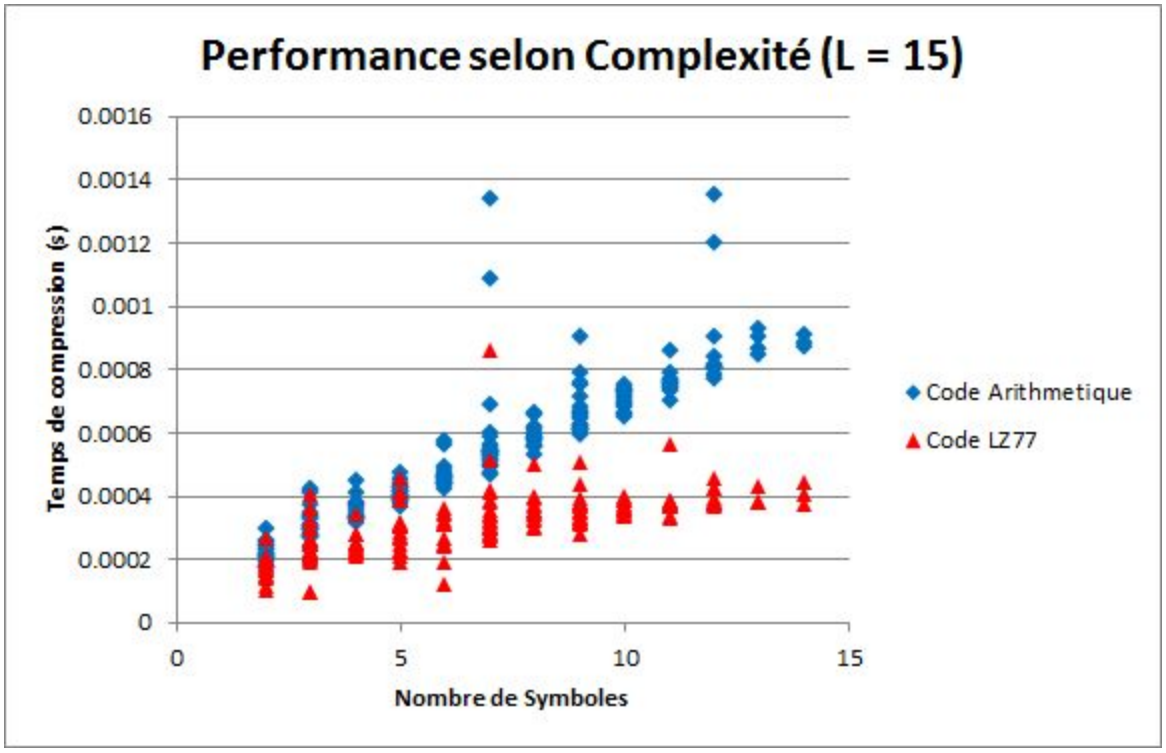
Les expériences pour l'hypothèse 1, relatives au taux de compression et sa relation au taux de répétition en gardant ce dernier constants sur des groupes de tests, donnent les résultats suivants:

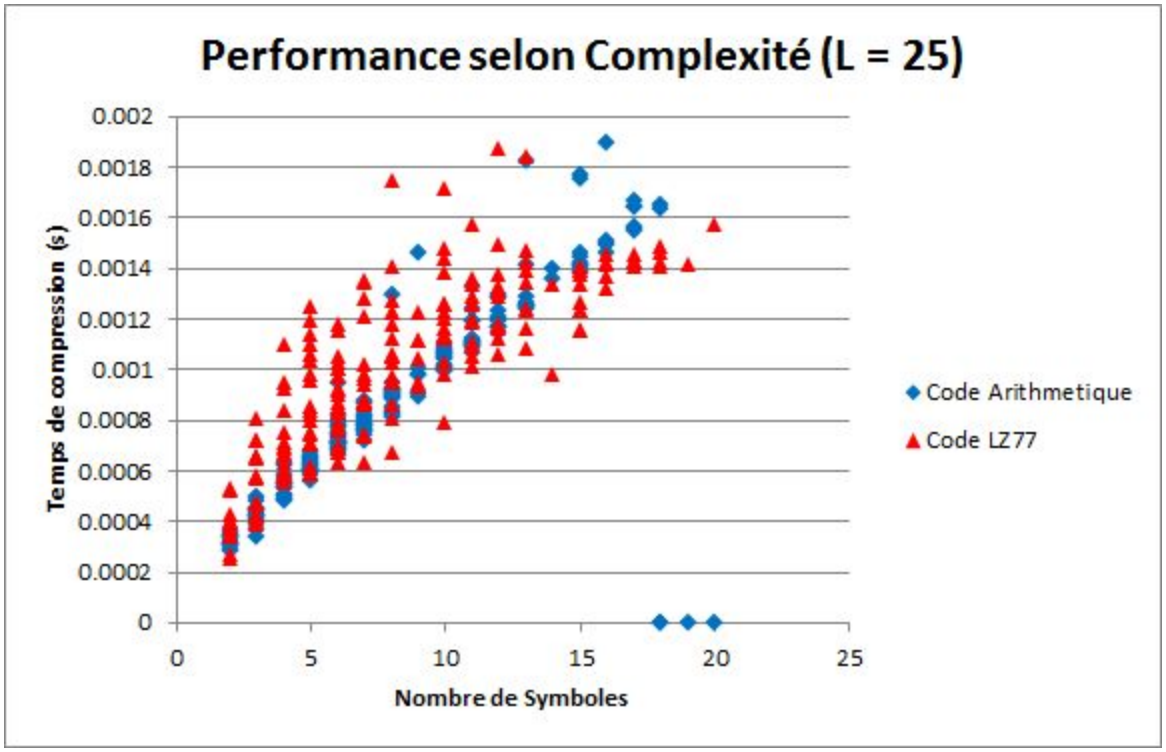


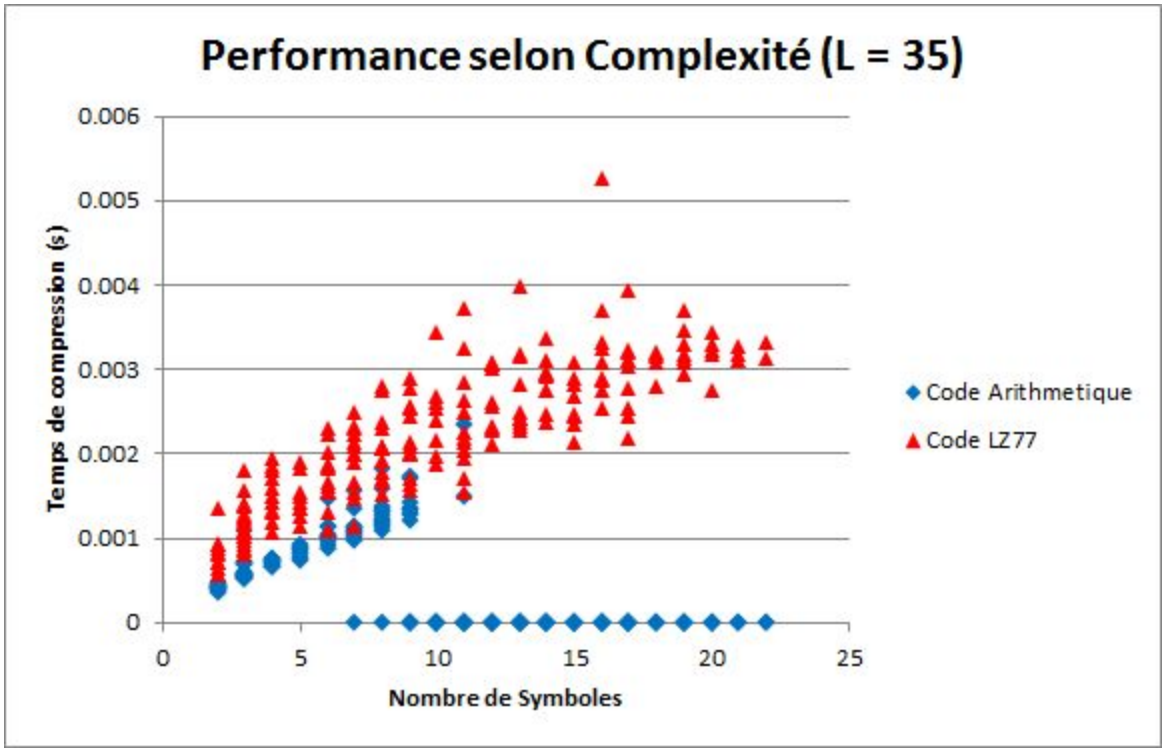


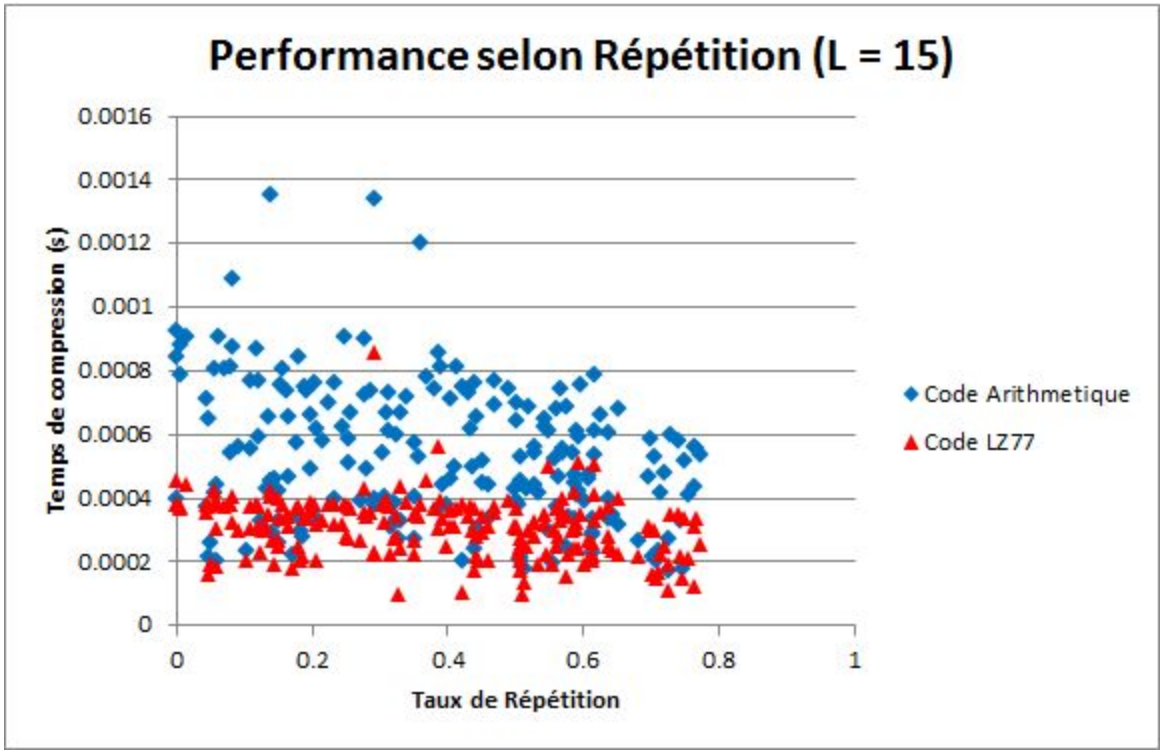


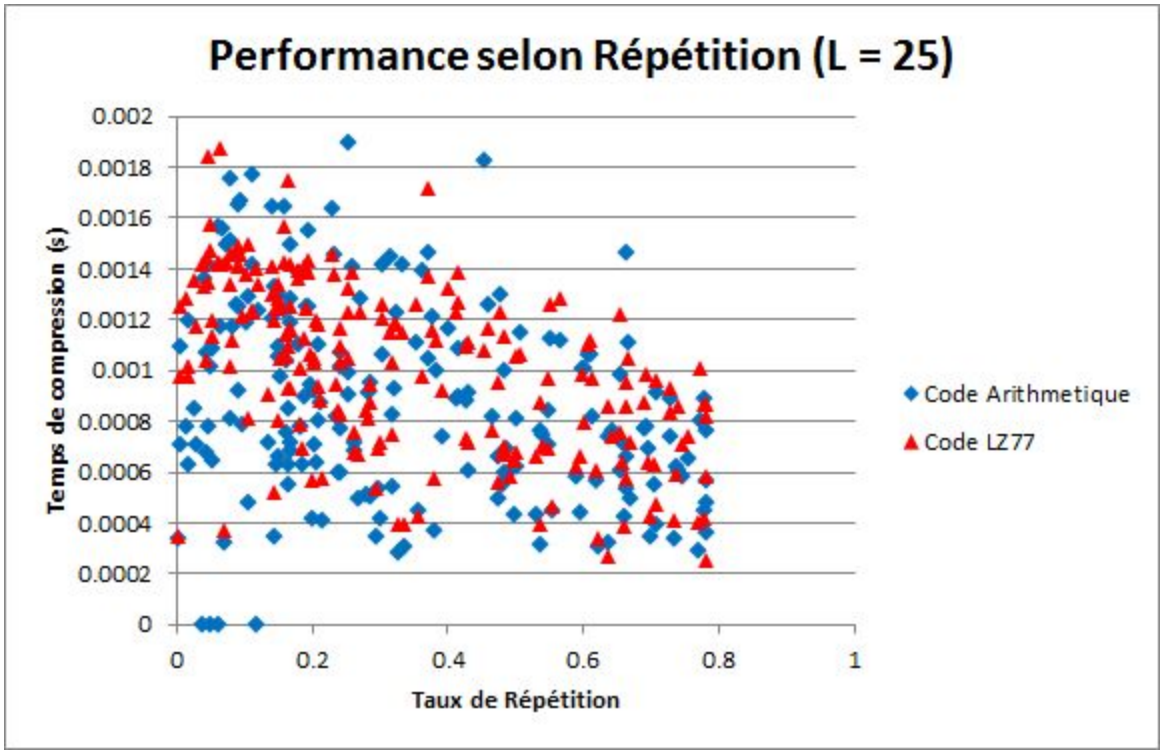
Les expériences pour l'hypothèse 2, qui compare la performance de l'algorithme selon la longueur du message en gardant des longueurs constantes pour des gammes de tests aléatoires, produit les graphes ci-dessous:

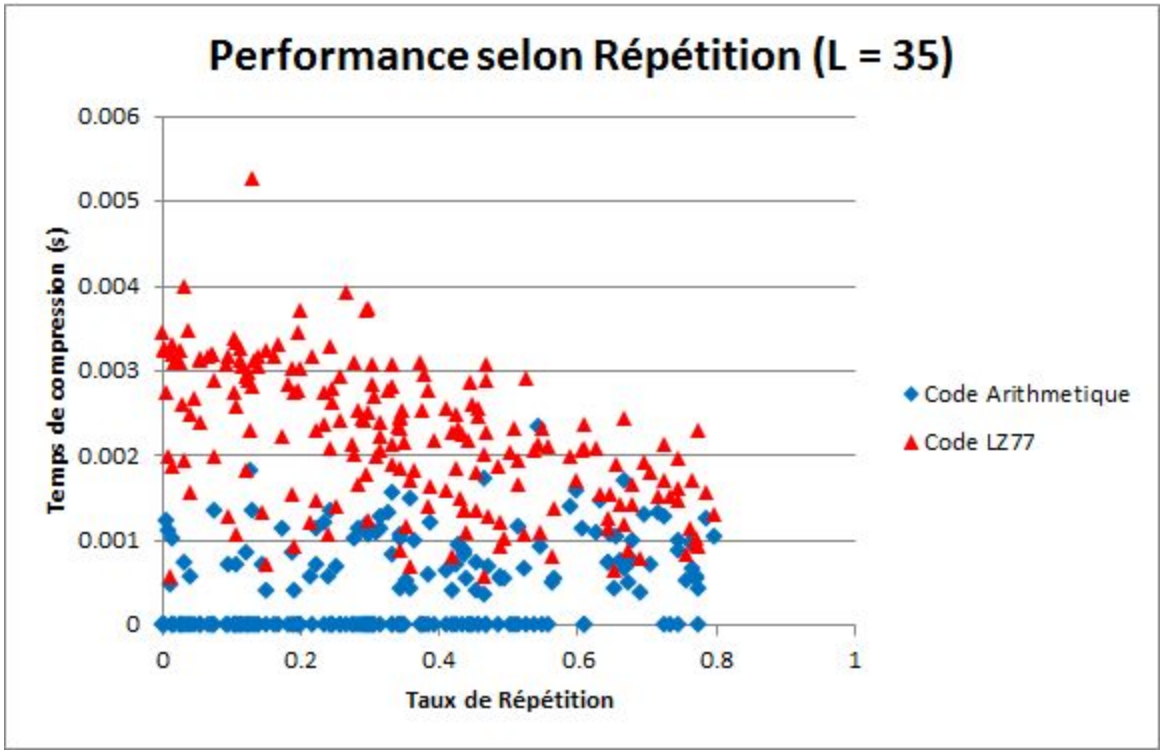



Performance selon Complexité (L = 25)
Temps de compression (s)
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
5
10
15
20
25
Nombre de Symboles
Code Arithmetique
Code LZ77

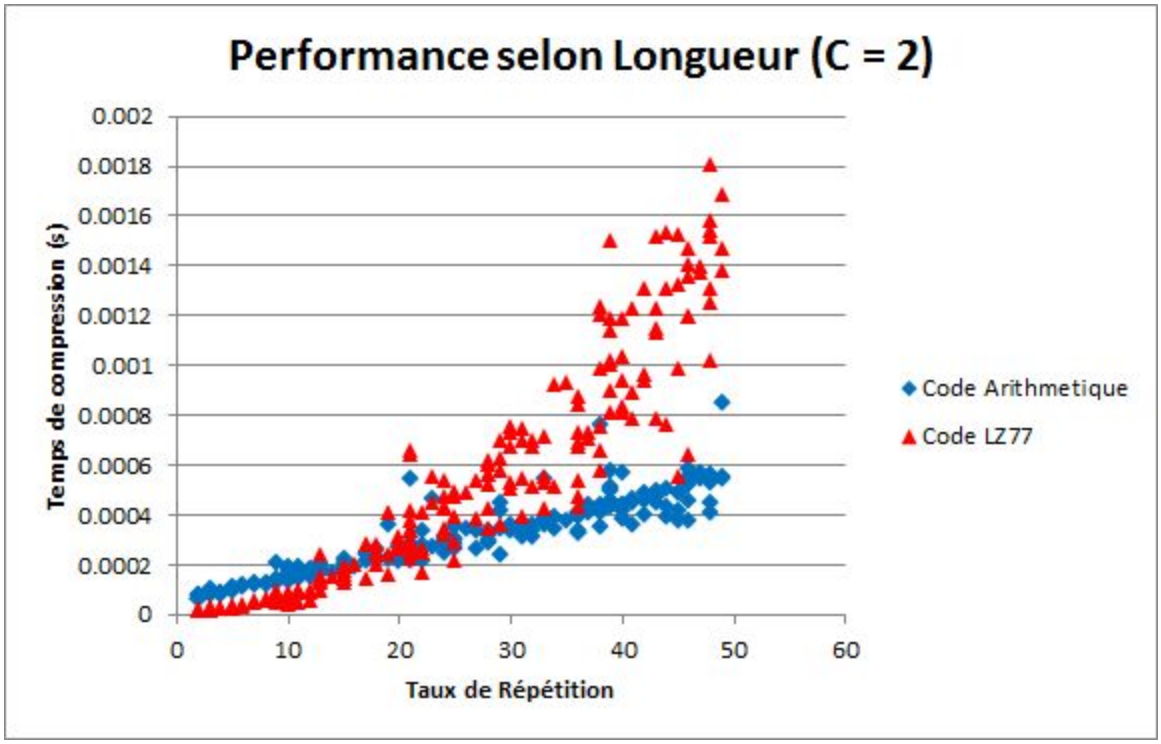

Performance selon Complexité (L = 35)
Temps de compression (s)
0.006
0.005
0.004
0.003
0.002
0.001
0
0
5
10
15
20
25
Nombre de Symboles
Code Arithmetique
Code LZ77

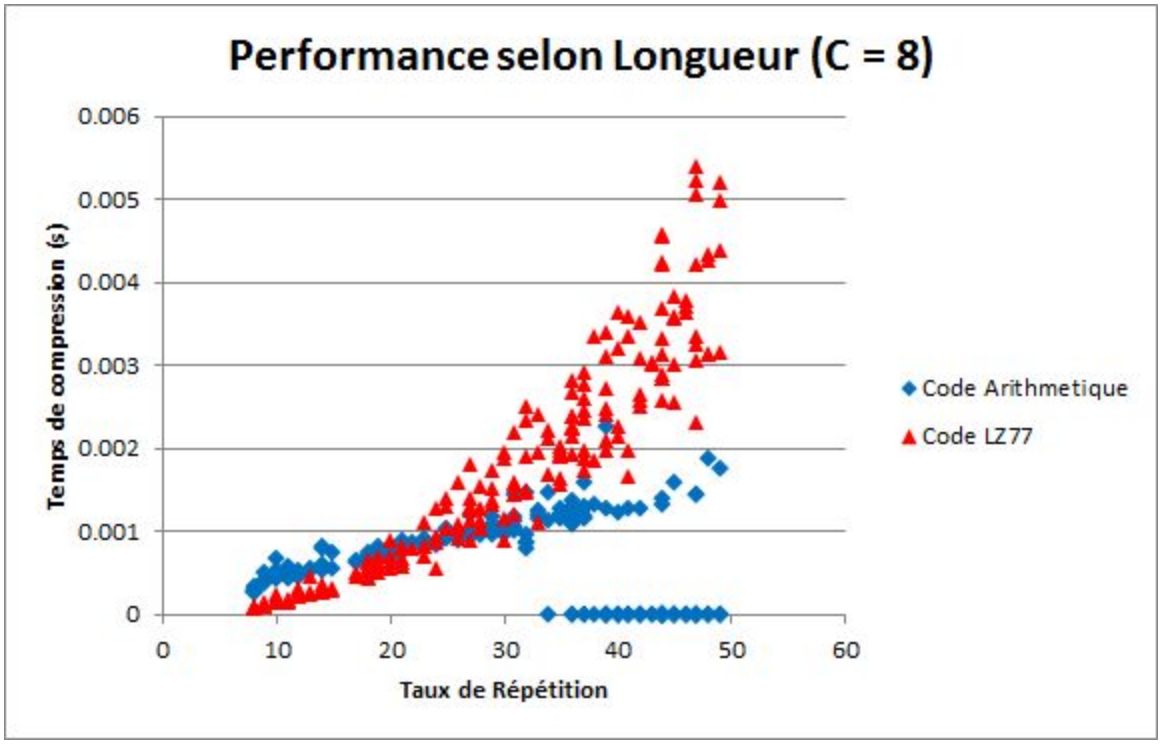

Performance selon Répétition (L = 15)
Temps de compression (s)
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
0.2
0.4
0.6
0.8
1
Taux de Répétition
Code Arithmetique
Code LZ77

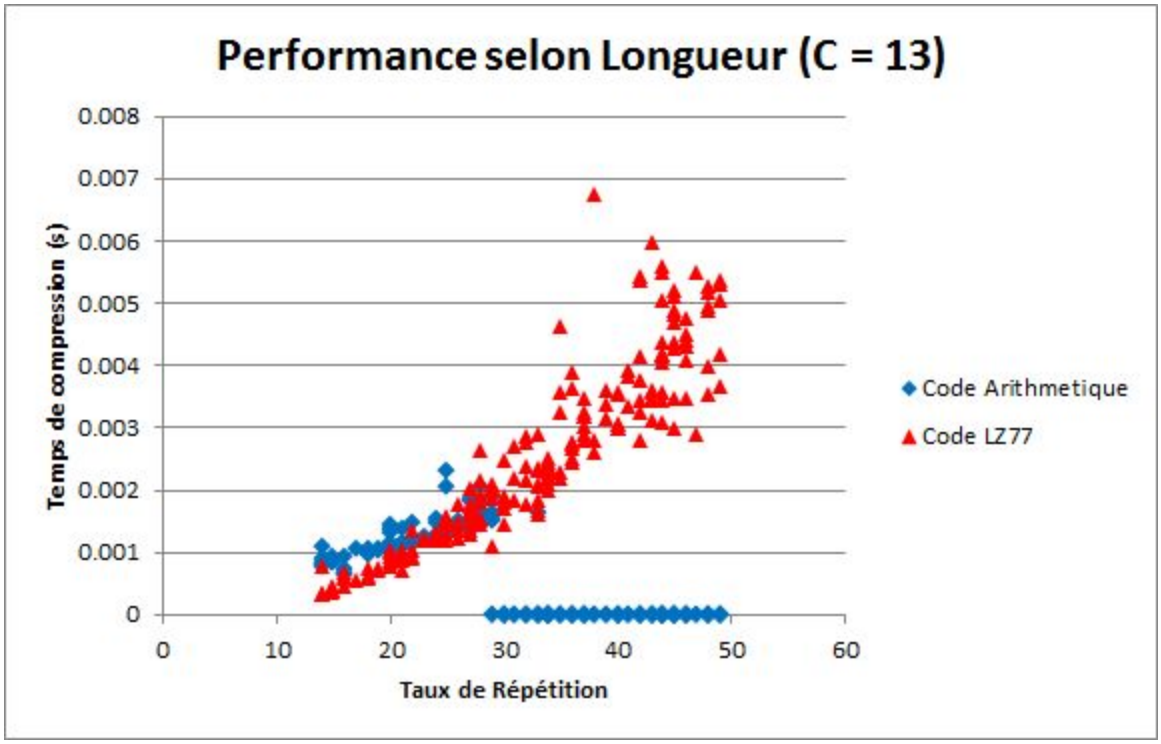

Performance selon Répétition (L = 25)
Temps de compression (s)
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0
0.2
0.4
0.6
0.8
1
Taux de Répétition
Code Arithmetique
Code LZ77

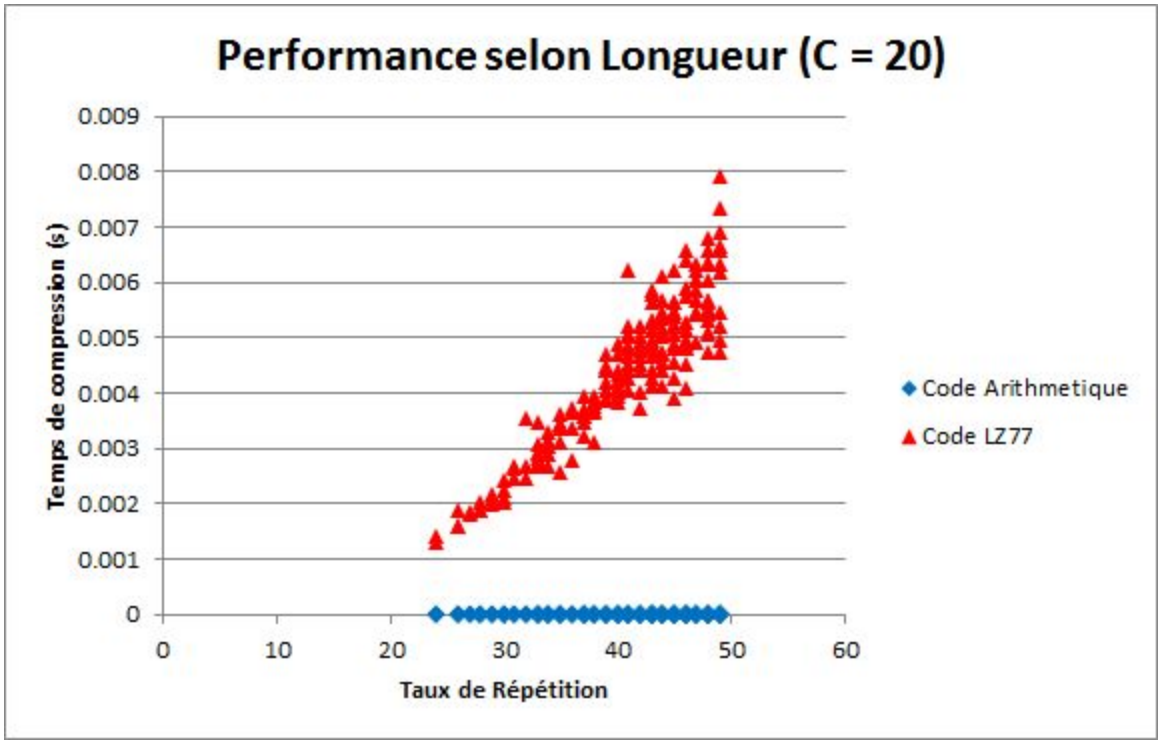

Performance selon Répétition (L = 35)
Temps de compression (s)
0.006
0.005
0.004
0.003
0.002
0.001
0
0
0.2
0.4
0.6
0.8
1
Taux de Répétition
Code Arithmetique
Code LZ77

Les expériences pour l'hypothèse 3, qui compare la performance de l'algorithme selon la complexité, donne les graphes suivants lorsque la complexité est maintenue constante:









Il faut toutefois noter que l'algorithme d'encodage Arithmétique approche rapidement de la limite de précision du type Décimal de python, et ainsi laisse certains points de données à zéro (NaN) sur les graphiques.

Question 4

Analyse

Hypothèse 1: Pour des messages contenant des séquences de symboles répétées, la taille du message encodé sera plus court avec l'encodage LZ-77.

L'algorithme LZ-77 devient marginalement plus efficace que l'algorithme Arithmétique, en termes de taux de compression, pour des messages plus courts. Un manque de données en raison de la limite de précision de l'algorithme Arithmétique empêche de porter une conclusion sur les messages plus longs, mais la tendance générale pour LZ-77 est de devenir, en moyenne, plus efficace que l'encodeur Arithmétique.

Hypothèse 2: Pour des messages plus longs, le temps de compression sera plus court avec l'encodage LZ-77.

Ici, on peut clairement voir que l'hypothèse avancée est fausse: plus les messages générés sont longs, plus le nuage de points de l'encodeur LZ-77 s'élève au dessus de celui de l'encodeur Arithmétique. Il est bon de noter qu'Il manque toutefois beaucoup de points pour les résultats de l'algorithme Arithmétique, mais selon la courbe qui apparaît dans les figures de Performance selon Complexité avec **L = 35**, la relation restera probablement linéaire.

Hypothèse 3: Pour des messages avec beaucoup de symboles différents, le temps d'encodage sera plus long avec l'encodage Arithmétique.

Selon les résultats observés par les séries d'expériences, la performance des algorithmes est beaucoup plus affectée par la longueur que par la complexité du message en entrée. Pour les messages plus court, l'encodeur Arithmétique n'est que marginalement plus lent que la méthode LZ-77, une tendance qui semble rester consistante avec une augmentation de la complexité. Par contre, les expériences sont non conclusives pour les messages plus long: nécessitant plus de caractères, la précision maximale de l'encodeur Arithmétique est dépassée et aucune donnée n'est disponible pour le cas de **C = 20**.