INF431 - Projet d'informatique

Sujet n°7 proposé par Bruno SALVY

ÉCHANTILLONNAGE DE GRANDES DONNÉES ET APPLICATIONS : SHAKESPEARE, LE WEB ET LES RÉSEAUX

mai 2013

Michel BLANCARD
Chantal DING

1 Dossier de présentation

Le sujet propose d'implémenter une petite bibliothèque d'analyse de grands flux de données, en faisant particulièrement attention à la complexité des algorithmes compte tenu du volume supposément grand des données traitées et avec toujours la contrainte de n'effectuer qu'une seule passe. Ce document résume la démarche que nous avons suivie pour répondre aux attentes du sujet.

Nous avons choisi le langage de programmation Java.

Nous avons utilisé trois sources de données à des fins de tests :

- Les œuvres complètes de Shakespeare, comme suggéré dans le sujet.
- Les œuvres complètes de Molière, pour disposer d'extraits écrits dans une autre langue.
- Les « pages of reverse engineering » de Fravia, auteur prolifique et plus proche de notre époque, pour étudier les différences de styles entre différentes littératures d'une même langue.

1 Hachage

Pour calculer les hashs des mots traités, nous avons choisi d'utiliser la bibliothèque `lookup3.c` de Robert Jenkin. De nombreux sites (Wikipedia, StackOverflow...) la donnent en référence (en plus du sujet). Sachant que l'auteur a lui-même consacré beaucoup d'efforts à chercher des biais, nous n'avons pas implémenté de tests poussés. Le test du χ^2, effectué sur le reste des hashs par la division euclidienne par des nombres allant de 100 à 4000, donne des résultats identiques à ceux obtenus par génération de nombre aléatoires, et conforme à la théorie qui prévoie une valeur moyenne du χ^2 égale à m, le nombre de groupes.

Pour interfacer la bibliothèque écrite en langage C avec le reste du programme en Java, nous avons naturellement utilisé la JNI (Java Native Interface). Nous bénéficions ainsi de la rapidité du langage C pour le calcul des hashs au détriment de la portabilité de Java, car la bibliothèque partagée exportant la fonction de hachage doit être recompilée pour chaque environnement. La durée nécessaire à un appel à la fonction de hachage est estimé à 0,5µs sur un netbook, ce qui nous paraît correct pour une utilisation en production sur un serveur.

2 Comptage approché

Le sujet propose d'implémenter l'algorithme HyperLogLog de Flajolet *et alii* pour compter de manière approchée, avec une complexité en espace constante, le nombre d'élément différents d'un

multi-ensemble.

Nous obtenons les estimations suivantes pour les œuvres complètes de Shakespeare, en fonction du paramètre b (pour un nombre exact de mots égal à 22929) :

b	4	5	6	7	8	9	10	11	12	13	14	15	16
n	39069	31038	26584	28354	27503	25904	24463	23650	23755	30447	58889	156710	506161

La meilleure approximation s'obtient pour b=11 (erreur=3%). En dessous, les résultats sont imprécis car le nombre de groupes est trop faible. Au dessus, des groupes se retrouvent vides, ce qui fait diverger le résultat. En effet, la précision de l'algorithme repose sur l'hypothèse que le nombre de groupes est bien plus petit que le nombre d'éléments comptabilisés. Nous en concluons que le choix de b doit être adapté à la taille des données étudiées. Conformément à la littérature sur le sujet, nous avons modifié la fonction `HyperLogLog` pour retourner une erreur (en l'occurence on retourne la valeur -1) si un groupe est vide à la fin de la passe sur les données. Dans le cas où l'on peut s'autoriser plusieurs passes sur les données, la procédure à suivre pour obtenir la meilleure approximation est d'incrémenter b jusqu'à provoquer une erreur. On obtient bien b_{max}=11 pour des données contenant 20.000 éléments distincts (Shakespeare : erreur = 3%; Molière : erreur = 1%), et b_{max}=13 pour Fravia (100.000 mots distincts, erreur = 1%).

Pour obtenir le décompte exact des mots différents, nous avons implémenté un algorithme de tri dichotomique prédictif de complexité quasi-linéaire (cf *R. Sedgewick*, Algorithms in C). Il prend 4,5s pour s'exécuter sur les œuvres complètes de Shakespeare, à comparer avec le léger gain de temps pour l'algorithme approché (2,7s) sur un netbook. L'algorithme exact tire son avantage non de son temps d'exécution mais de sa capacité à s'exécuter en complexité spatiale constante, ce qui permet de traiter des données de taille supérieure à la mémoire disponible.

3 Similarités

Pour obtenir le hachage des k-shingles, la classe `MyReader` utilisée pour lire les fichiers d'entrée garde à chaque instant en mémoire les k derniers mots lus. Pour lire le k-shingle suivant, il suffit de lire le mot suivant dans le texte, puis de l'ajouter à la fin de la liste des mots gardés en mémoire tout en supprimant le premier mot (le plus ancien). Le nouvel élément créé est alors formé à partir de la concaténation des mots de la mémoire. Notons que pour la lecture du premier k-shingle, il faut initialiser la mémoire et donc lire k mots consécutifs, d'où la présence d'un champ booléen dans la classe permettant de distinguer ce premier appel des suivants. La lecture du texte mot-à-mot correspond alors simplement au cas k=1.

Les mesures de similarités dépendent de la taille des deux extraits comparés. Nous avons choisi de comparer des extraits de 1Mo avec une valeur de b égale à 10 afin d'uniformiser les résultats et de leur donner une réelle signification.

On obtient les résultats suivants pour des valeurs de k allant de 1 à 4 (on indique les bornes min et max):

k=1	**Shakespeare**	**Fravia**	**Molière**
Shakespeare	0.42<0.46		
Fravia	0.06<0.16	0.19<0.30	
Molière	-0.04<0.02	-0.003<0.04	0.356

k=2	Shakespeare	Fravia	Molière
Shakespeare	0.11<0.15		
Fravia	-0.008<0.06	0.06<0.14	
Molière	-0.04<0.04	-0.02<0.02	0.150

k=3	Shakespeare	Fravia	Molière
Shakespeare	0.008<0.06		
Fravia	-0.02<0.05	-0.02<0.04	
Molière	-0.006<0.05	0.003<0.07	0.021

k=4	Shakespeare	Fravia	Molière
Shakespeare	-0.015<0.03		
Fravia	-0.04<0.04	-0.06<0.008	
Molière	-0.02<0.02	-0.02<0.06	0.059

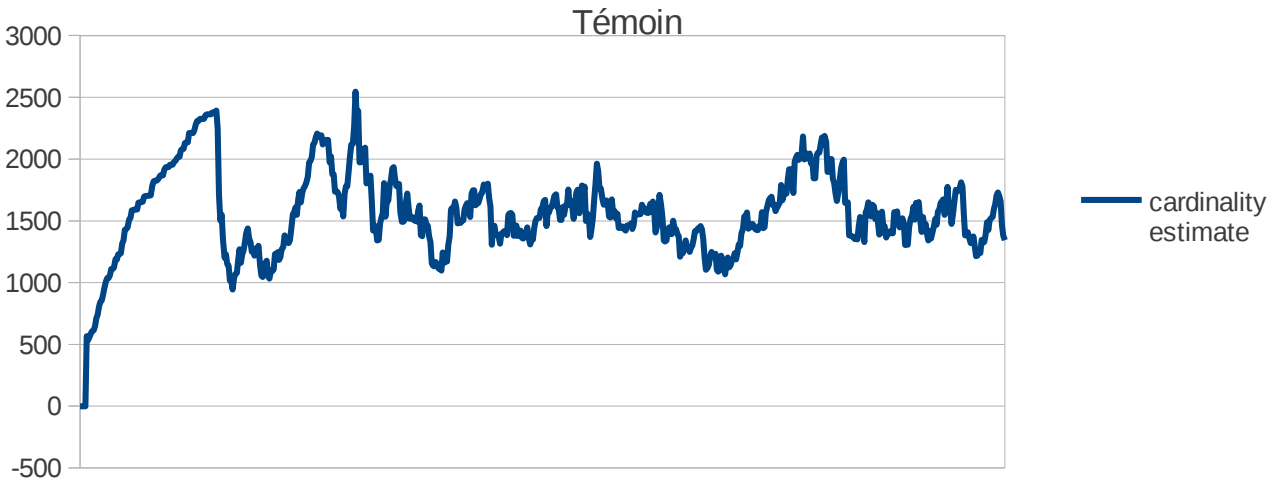
Pour k=1, le seuil de 0,05 discrimine les paires d'extraits écrits dans des langues différentes, et le seuil de 0,17 discrimine les paires d'extraits ayant des auteurs différents.

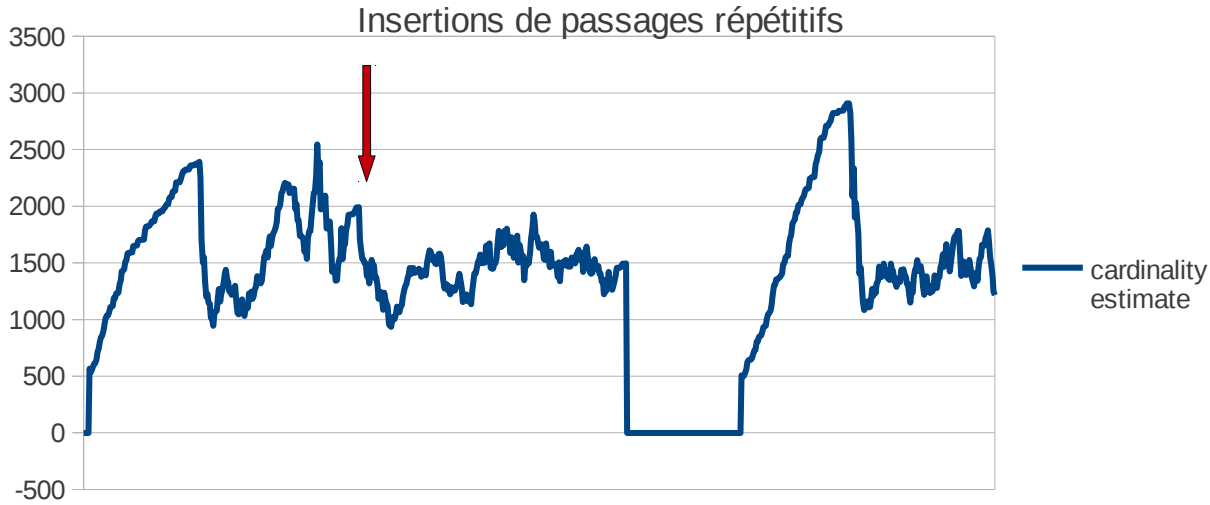
Pour k=2, le seuil de 0,05 discrimine toujours les paires d'extraits écrits dans des langues différentes, mais on ne distingue plus le style d'écriture. Pour k=3 ou k=4, les résultats ne donnent rien, sans doute en raison du nombre trop petit de tournures (k-shingles faisant sens) par rapport au nombre total de k-shingles.

4 Fenêtre glissante

L'estimation du nombre d'éléments distincts peut être limité aux éléments récemment rencontrés, pour être utilisable sur des flux de données non finis, comme des logs d'un serveur. Contrairement au problème précédent, on s'autorise un certain nombre de groupes vides fixé à 10 %.En effet, il arrive souvent qu'un mot qui a été précédemment gardé en mémoire soit devenu trop vieux et n'ait pas encore été remplacé. Si cette proportion de 10 % de groupes vides est atteinte, le résultat de la fonction `hyperLogLog` est considéré comme trop peu fiable (la fonction renvoie -1).

Pour nos tests, nous avons utilisé une fenêtre de 10.000 mots sur un passage de Shakespeare de 70000 mots. Une estimation est faite tous les 100 mots et le résultat est gardé en mémoire afin de constituer un graphique. Nous avons inséré des passages fortement répétitifs dans un premier test, puis des passages composés de mots tous différents dans un deuxième test. La première insertion, située au tiers du passage étudié, comporte 1.000 mots (tous identiques pour le premier test, tous différents pour le deuxième). La deuxième insertion, située aux deux tiers, contient 10.000 mots. Après stabilisation des valeurs d'estimations (2 fenêtres glissantes), les valeurs obtenues oscillent entre 1.000 et 2.000 mots. La première insertion (localisée par une flèche rouge) passe toujours inaperçue, mais la deuxième, plus importante d'un ordre de grandeur, se repère facilement au vu du graphique.


Témoin
3000
2500
2000
1500
1000
500
0
-500
cardinality estimate


Insertions de passages répétitifs
3500
3000
2500
2000
1500
1000
500
0
-500
cardinality estimate


Insertions de passages aléatoires
8000
7000
6000
5000
4000
3000
2000
1000
0
-1000
cardinality estimate

5 Échantillonnage

L'objectif est ici de sélectionner arbitrairement quelques mots de l'extrait traité, sans que la fréquence des mots influence ce choix. La stratégie mise en œuvre est de garder les k mots ayant les plus petits hashs pour un ordre arbitraire.

Lors du parcours, on garde en mémoire dans un tableau non trié de taille k les mots ayant b+1 zéros en début de hash, et les mots ayant seulement b zéros en début de hash dans un tableau trié.

Avec 6 mots ainsi extraits de la pièce Macbeth (*alarum, his noble, ambition, foolish, strange*), nous obtenons les résultats suivants par une recherche sur Google :

1. en.wikipedia.org/wiki/Banquo
2. www.gutenberg.org/files/3186/3186-h/3186-h.htm
3. shakespeare.mit.edu/macbeth/full.html
4. www.barnesandnoble.com/w/endurance-alfred-lansing/...
5. www.one-act-plays.com/dramas/macbeth.html
6. http://www.opensourceshakespeare.org/views/plays/...
7. www.fullbooks.com/The-Foolish-Lovers7.html
8. www.tomfotherby.com/blog/index.php/short-quotes/
9. en.wikiquote.org/wiki/Futurama
10. thedivinersseries.com/

Les résultats 1, 3, 5 et 6 sont particulièrement pertinents.

6 Souris

Pour obtenir une estimation des k-souris, il suffit de garder en mémoire le nombre d'apparitions des éléments gardés en mémoire.

7 Icebergs

La recherche des icebergs avec θ égal à 0.005 donne les résultats suivants pour Shakespeare, Molière et Fravia respectivement :

- *where it an he since us thee were mate ll my vows every and partake richly so i perform king old hand which clown honourable thou sir find many they vain mind the not never born this some wife me demand each more would am in turtle florizel heavens shall shouldst shepherd now s know have preserved you paulina far your from camillo like away his of art good till love no what consent we let how speak justified o time wide e leisurely on polixenes son place first by there prayer much both unto match kings thought with will him dissever as precious but holy law swear er put directing troth daughter seek say wither life are here made worth may winners partly one lost lament honesty who bough gap whose autolycus do if lived thy come think exultation pair leontes hast brother dead look t hence found lord between again for mine part upon father see grave gentleman said she then make husband take be proteus her looks thine wing valentine answer that d saw a was plight exeunt all pardons noted hastily hear lead stone is question ill suspicion peace to*
- *verser justes retour d'un de lui y que les tartuffe bien succès des devoir souhaitez loué sein qu'on te qu'il témoigner cette couronner voilà à autre mal dit tout maintenant ce puisse mérite pas vertu dom 1945que me justice faire avec oui demande indignités il loyal genoux bonne laissez amant vice l'aurait sait traitement d'une soins adoucir du au s'est favorable dire*

acquittés moi dorine droits plutôt dans misérable raison doux corrige bontés ah … ciel nous le mais cœur sa souvient vos joignez descendez fasse on soit frère son géronte hymen osé grand rendre l point mon pouvoir fin cléante faudra pourvoir remords prince emmène madame l'accable vie mariane monsieur bonté sganarelle scapin zèle tous appuyant 1960aux heureux généreux est vous joie détestant vit pernelle jamais je quand mieux 8 ses pèdre damis une mauvais irez premier pour hé ferme rien jour action montrer votre et en 1955tandis respire 1950et qui elmire peu donne ma puis c'est pieds qu'à traître la plus louer allons fait se dont orgon 1940on toutes flamme arrêtez perd pense moins sincère un a scène par déploie ne destin si qu'autrefois valère récompense l'exempt

- *value ideale it width mov noanon gt xoom anonymity and div i src scn 99 images lab quot void html reverse fp ebp com head img the engines fravia www this form input hr homepage 200 in 3 br new botstart tppabs cocktails s greythorne else char 04x push javascript you far option list 0 bots jump 222 cocktail 2 103 129 of mail msgboard a7 what asm orc mcb members reversing center we let button len w tools wars search chain e javascri searengi href p student size 13 illegal reverser define 136 printf legal align body alt l bulletr academy int vspace students 105 http type go gif 9 td env get ax segm display fravpage onclick relocator ebx page border red psp image hspace 137 88 w3svc1 tt byte if script 015f b 1 insidetheweb move typedef 6 block font walk word mk database select buf mbs return seg for tsehp main engineering links elements vec forms index antismut owner cgi eax mb155985 nbsp d0 end gods height unsigned cmdline lt color that a esi essays 143 htm info 5 bottom 116 is to*

Les mots sélectionnés sont effectivement souvent des mots très fréquemment rencontrés dans leurs textes.

2 Notice d'utilisation

1 Descriptif général des classes

Classes Java

`Data.java`
`Settings.java`
Ces deux classes comportent des méthodes publiques pour lire des données afin de les traiter.

`StaticAnalysis.java`
`DynamicAnalysis.java`
`Sampling.java`
`Icebergs.java`
`Mice.java`
Ces six classes comportent des méthodes publiques et permettent d'effectuer l'analyse statistique d'un flux de données.

`FingerPrint.java`
`MyReader.java`
`Element.java`
`SamplingBag.java`

`CountingBag.java`
Ces quatre classes sont les structures de données utilisées par le programme, leur visibilité est `protected` *et elles ne sont pas destinées à être utilisées en dehors de la bibliothèque.*

`HashTest.java`
Cette dernière classe permet de tester la fonction de hachage choisie en effectuant un test du χ^2.

Code natif (langage C)

La classe `Element` utilise la fonction `hash` exportée par la librairie partagée `Element`. Cette librairie doit être compilée en code natif avant de pouvoir être utilisée. Voici des exemples de commandes pour compiler respectivement sur Windows et Linux :

```
x86_64-w64-mingw32-gcc -shared -I"C:/Program
Files/Java/jdk1.7.0_17/include" -I"C:/Program
Files/Java/jdk1.7.0_17/include/win32" src/libElement.c -o
bin/libElement.dll
```

```
gcc -shared -I/opt/jdk1.7.0_21/include/
-I/opt/jdk1.7.0_21/include/linux src/libElement.c -o
bin/libElement.so
```

2 Pour démarrer : création d'un flux de données, les classes Data et Settings

L'ouverture et la lecture d'un flux d'entrée se fait grâce à la classe `Data`. Elle permet de lire un fichier via son nom ou une page Internet via son URL. Voici la syntaxe du constructeur pour créer un objet de type `Data` :

```
Data (String name, int type, Settings s)
name : le nom du fichier, ou l'URL de la page
type : Data.FILE pour un fichier, Data.URL pour une url
Settings : (optionnel) un objet de type settings (voir infra)
précisant le type de données et le mode de lecture. Si aucun
n'est donné, les valeurs par défaut de la classe Settings seront
appliquées.
```

Les objets de type `Settings` permettent d'indiquer au programme le type de données à traiter (texte, logs) ainsi que le mode de lecture (mot-à-mot ou par k-shingles). Voici la syntaxe du constructeur :

```
Settings(int type, int k)
type : Settings.TEXT ou Settings.LOG
k : nombre de mots consécutifs lus
```

Les valeurs par défaut sont : `type = Settings.TEXT`, `k = 1`.

3 Comptage approché, comparaisons: StaticAnalysis

La classe `StaticAnalysis` permet d'étudier un flux de données statique (typiquement : les

œuvres complètes de Shakespeare) en calculant une empreinte du fichier. Un objet de la classe `StaticAnalysis` se crée via la syntaxe suivante :

```
StaticAnalysis(Data d, int b)
d : le flux de données étudié
b (optionnel) : la précision désirée. Valeur par défaut : 11
```

Il existe également un constructeur permettant de faire l'union de deux empreintes de fichier :

```
StaticAnalysis(StaticAnalysis fp1, StaticAnalysis fp2)
fp1 : l'empreinte du premier fichier
fp2 : l'empreinte du deuxième fichier
```

Remarque : les deux empreintes doivent avoir été calculées avec la même précision, sinon le programme affiche un message d'erreur.

Pour obtenir une estimation du nombre d'éléments distincts d'un fichier à une précision b, on fait appel à la méthode statique `hyperLogLog` de la classe `StaticAnalysis`, dont la syntaxe est la suivante :

```
public static double hyperLogLog (Data input, int b)
input : le flux de données étudié
b : la précision désirée
```

On peut également appeler depuis `stdin` la méthode `main` de la classe qui donne les estimations pour les différentes valeurs de b en exécutant :

```
export LD_PATH_LIBRARY=.          # set java.library.path
java StaticAnalysis.class <nom_du_fichier>
```

Cette classe permet également le calcul de similarité entre deux fichiers grâce à la méthode statique `similarity` :

```
public static double similarity(StaticAnalysis fp1,
StaticAnalysis fp2)
fp1 : l'empreinte du premier fichier
fp2 : l'empreinte du deuxième fichier
```

On peut également donner en argument à `similarity` un tableau d'empreintes, auquel cas la méthode renvoie une matrice des comparaisons deux à deux des toutes les empreintes :

```
public static double [][] similarity(StaticAnalysis[] fp)
fp : un tableau regroupant les empreintes à comparer.
```

Pour comparer des fichier dans un dossier, on utilise `similarPairs`, qui affiche le résultat sous la forme d'un tableau à double entrée dans la sortie standard :

```
public static void similarPairs(String directory, int b, int k)
```

```
directory : le path du dossier d'entrée
b : la précision désirée
k : la taille des k-shingles utilisés pour la comparaison.
```

4 Étude dynamique, monitoring : DynamicAnalysis

Pour étudier de façon dynamique un flux de données, il faut commencer par créer un objet de type `DynamicAnalysis` :

```
DynamicAnalysis (int b, int W, int nbr)
b : la précision voulue
W : la taille de la fenêtre
nbr : le nombre de points gardés dans l'historique
```

Pour lire un nouvel élément et mettre à jour l'analyse :

```
public void newElement (Data d)
d : le flux de données étudié
```

Obtenir l'historique des estimations :

```
public double[] getMemory()
```

5 Échantillonnage statistique : Sampling, Mice, Icebergs

Pour effectuer un échantillonnage d'éléments représentatifs, on commence par créer un objet de la classe `Sampling` :

```
Sampling(Data d,int k)
d : le flux de données étudié
k : la taille de l'échantillon voulu
```

Le calcul et l'affichage de l'échantillon s'obtient par la méthode `printSample` :

```
public void printSample()
```

Pour obtenir une estimation du nombre de k-souris on crée un objet de la classe `Mice` puis on appelle la méthode `getMiceProportions` :

```
Mice(Data d,int n)
d : le flux de données étudié
n : la taille de l'échantillon utilisé pour l'estimation des k-souris
```

```
public double getMiceProportion(int occurrences)
occurrences : le nombre d'occurrences des souris (i.e. le paramètre k des k-souris)
```

Pour afficher les icebergs on dispose de `Icebergs.printIcebergs` :

public static void printIcebergs(Data input, double theta)
input : le flux de données étudié
theta : un double, la fréquence minimale voulue pour les icebergs