Parallélisation de l'échantillonnage de motifs séquentiels

Lamine Diop, Cheikh Ba

▶ **To cite this version:**

Lamine Diop, Cheikh Ba. Parallélisation de l'échantillonnage de motifs séquentiels. EGC (Extraction et Gestion des Connaissances) 2021, Jan 2021, Montpellier, France. hal-03239177

HAL Id: hal-03239177
https://hal.science/hal-03239177

Submitted on 27 May 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Parallélisation de l'échantillonnage de motifs séquentiels

Lamine Diop*, Cheikh Ba**

*Université de Tours, France, lamine.diop@univ-tours.fr
**Université Gaston Berger de Saint-Louis, Sénégal, cheikh2.ba@ugb.edu.sn

Résumé. Durant ces 10 dernières années, le domaine de la fouille de données a connu d'importants travaux sur la découverte de motifs par échantillonnage en sortie. Très récemment, ces méthodes d'échantillonnage ont été appliquées sur des données séquentielles qui sont d'une nature complexe. La complexité de ces données réside sur leur structure qui a un impact notoire sur la rapidité du calcul et notamment sur le pré-traitement. A cela s'ajoute la taille des bases de données qui, de nos jours, deviennent très volumineuses. Dans ce papier, nous avons montré comment bénéficier du modèle de programmation BSP (Bulk Synchronous Parallel) pour améliorer l'efficacité des méthodes d'échantillonnage en sortie sur les données séquentielles. En effet, nous proposons un algorithme distribué et parallèle qui s'opère sur des bases de données séquentielles sciemment distribuées afin d'accélérer le temps de calcul. Les analyses que nous avons faites montrent l'impact positif du framework sur le temps d'exécution de la méthode.

1 Introduction

La découverte de motifs séquentiels (Agrawal et Srikant, 1995) est une tâche très étudiée dans de nombreux domaines de la vie réelle telles que la médecine, la biologie, l'exploration Web, etc. Au début, l'objectif principal était de trouver de manière exhaustive tous les motifs séquentiels qui ont une fréquence supérieure à un seuil donné. Les méthodes d'extraction exhaustive de motifs fréquents rencontrent deux problèmes majeurs. 1) Il n'est pas du tout facile de régler le seuil minimal de fréquence. S'il est trop élevé, la sortie risque d'être vide et s'il est très petit, l'ensemble des motifs renvoyés devient trop grand et alors difficile à analyser. 2) Le temps de calcul devient très élevé si la base de données est très volumineuse. Par conséquent, des méthodes de fouille de motifs dites optimales (Fournier Viger et al., 2017) ont été proposées pour supprimer le seuil tout en étant très rapides. Cependant, ces dernières rencontrent un problème sur la diversité des motifs produits. Par exemple, l'ensemble vide qui est le plus fréquent dans une base de données n'est pas souvent intéressant pour l'utilisateur.

Dans ce contexte, les méthodes d'échantillonnage en sortie (Al Hasan et Zaki, 2009; Boley et al., 2011; Diop et al., 2018) ont été proposées pour trouver des motifs très rapidement. Le but de ces méthodes est d'obtenir un échantillon de motifs où chaque motif est tiré avec une probabilité proportionnelle à une mesure d'intérêt choisie par l'utilisateur. Dans le cas des données séquentielles, l'algorithme CSSAMPLING (Diop et al., 2018) qui tire des motifs en fonction de la fréquence et sous des contraintes de normes minimales et maximales pour éviter la malédiction longue traîne a été étendu à un algorithme plus générique appelé NUSSAMPLING

(Diop et al., 2019). Ce dernier est générique dans le sens où il prend en compte toute mesure d'intérêt qui s'écrit sous le produit de la fréquence et d'une fonction d'utilité fondée sur la norme (aire, décroissance exponentielle $\alpha \in]0, 1]$, etc.). Cependant, si la phase de tirage d'un motif séquentiel est très rapide, la phase de prétraitement peut être très coûteuse en fonction des caractéristiques de la base de données séquentielles. Même si elle est réalisée en une seule fois, elle est refaite à chaque fois que la contrainte de norme maximale augmente.

Pour résoudre ce problème, cet article propose une version distribuée et parallélisée de l'algorithme NUSSAMPLING utilisant le framework Pregel (Malewicz et al., 2010). En effet, lorsque des données sont trop volumineuses ou complexes pour être traitées en temps raisonnable, une solution courante nécessite l'utilisation d'un grand nombre de machines pour un stockage distribué et un traitement parallèle. Récemment, Google avait proposé sa plateforme Hadoop, rapidement devenu de facto un standard, et son paradigme MapReduce (Dean et Ghemawat, 2008) comme un modèle de programmation parallèle. Cependant, il a été reconnu que ce paradigme ne convient pas aux algorithmes itératifs, dû à des entrée/sorties excessives avec HDFS (*Hadoop distributed file system*), le système de fichiers distribué de Hadoop, et le *shuffling* de données à chaque itération. C'est la raison pour laquelle plusieurs solutions de calcul distribué en mémoire ont été proposées, notamment Pregel (Malewicz et al., 2010), Spark/GraphX (Gonzalez et al., 2014) et PowerGraph (Gonzalez et al., 2012), pour accélérer l'exécution d'algorithmes itératifs. La plupart de ces plateformes suivent un modèle de programmation centré sommet (ou nœud de graphe). Par exemple, dans Pregel, basé sur le modèle BSP (Valiant, 1990), dans chaque itération, chaque nœud (ou sommet) peut recevoir des messages de ses voisins entrants, met à jour son état et peut envoyer des messages à ses voisins sortants. Certains travaux, à l'exemple de (Ba et Gueye, 2020), ont montré l'intérêt d'utiliser ces plateformes en ce qui concerne l'efficacité des solutions proposées.

Ainsi, l'algorithme NUSSAMPLING-Pregel vise à paralléliser l'échantillon de motifs à partir d'une base de données séquentielles sciemment fragmentée pour accélérer le calcul. Il est important de noter que, contrairement à DDSAMPLING (Diop et al., 2020), un algorithme centralisé qui opère sur des données nativement distribuées comme celles des triplestores, notre méthode est distribuée sur les différents sites du réseau sous Pregel et peut ensuite être exécutée en parallèle par les noeuds. À notre connaissance, c'est le premier algorithme parallèle et distribué d'échantillonnage de motifs en sortie.

2 Préliminaires

2.1 Définitions

Soit $\mathcal{E}$ un ensemble fini de littéraux appelés *items*. Un *itemset* Y est un sous-ensemble de $\mathcal{E}$, $Y \subseteq \mathcal{E}$. Une *séquence* $\gamma = \langle Y_1 \dots Y_n \rangle$ définie sur $\mathcal{E}$ est une liste ordonnée d'itemsets non vides $Y_i \subseteq \mathcal{E}$ ($1 \leq i \leq n$, $n \in \mathbb{N}$). La *taille* d'une séquence γ notée par $|\gamma|$ est le nombre d'itemsets qui forment γ, $|\gamma| = n$. La somme des cardinalités de tous les itemsets de la séquence γ est égale à sa *norme*, notée par $\|s\|$, i.e. $\|s\| = \sum_{i=1}^{n} |Y_i|$. Dans la suite de ce papier, nous convenons que γ^l représente le préfixe $\langle Y_1 Y_2 \dots Y_l \rangle$ de γ ($0 \leq l \leq n$, $l \in \mathbb{N}$), γ^0 étant la séquence vide (représentée par $\langle\rangle$) et $\gamma[j] = X_j$ est le j-ème itemset de la séquence γ ($1 \leq j \leq n$, $j \in \mathbb{N}$). Nous notons $\mathcal{L}_S$ le langage des motifs séquentiels, et une base de données séquentiel $\mathcal{S}$ sur $\mathcal{E}$ est un mutlti-ensemble de séquences définies sur $\mathcal{E}$.

Une séquence $\gamma' = \langle X'_1 \dots X'_{n'} \rangle$ est une sous-séquence d'une séquence $\gamma = \langle X_1 \dots X_n \rangle$, dénoté par $\gamma' \sqsubseteq \gamma$, s'il existe une séquence d'indices $1 \leq i_1 < i_2 < \cdots < i_{n'} \leq n$ telle que pour tout $j \in [1..n']$, on a $X'_j \subseteq X_{i_j}$. Nous notons par $\phi(\gamma)$ l'ensemble des sous-séquences de la séquence γ, i.e. $\phi(\gamma) = \{\gamma' \in \mathcal{L}_S : \gamma' \sqsubseteq \gamma\}$, et $\Phi(\gamma)$ sa cardinalité, i.e. $\Phi(\gamma) = |\phi(\gamma)|$.

Il est évident de noter que, dans le domaine des motifs séquentiels, une sous-séquence $\gamma' = \langle Y'_1 \dots Y'_m \rangle$ peut apparaître plusieurs fois dans une séquence $\gamma = \langle Y_1 \dots Y_n \rangle$ s'il existe plusieurs séquences d'indices $1 \leq i_1 < i_2 < \cdots < i_m \leq n$ telles que pour tout $j \in [1..m]$, on a $Y'_j \subseteq Y_{i_j}$. Dans ce cas, il existe multiples *occurrences* de la sous-séquences γ' dans la séquence γ. Les auteurs dans (Diop et al., 2019) expliquent comment chaque occurrence est représentée en utilisant une forme canonique :

Définition 1 (Occurrence) *Etant donné une séquence $\gamma = \langle X_1 \dots X_n \rangle$, une liste ordonnée d'itemsets $o = \langle Z_1 \dots Z_n \rangle$ de même taille que γ est une occurrence d'une sous-séquence $\gamma' = \langle X'_1 \dots X'_{n'} \rangle$ de γ s'il existe une séquence d'indices $1 \leq i_1 < \cdots < i_{n'} \leq n$ telle que pour tout $j \in \{i_1, \dots, i_{n'}\}$, on ait $Z_{i_j} = X'_j$, et tout $j \in \{1, \dots, n\} \setminus \{i_1, \dots, i_{n'}\}$, on ait $Z_j = \emptyset$. Cette séquence d'indices, appelée signature de o, est unique par définition.*

Exemple 1 *Pour la séquence $\gamma_2 = \langle (ab)c(ac) \rangle$, $o_1 = \langle (a)(c)\emptyset \rangle$ et $o_2 = \langle (a)\emptyset(c) \rangle$ sont deux occurrences de la sous-séquence $\gamma'_2 = \langle (a)(c) \rangle$.*

2.2 NUSSAMPLING algorithme d'échantillonnage de motifs séquentiels

NUSSAMPLING est une méthode d'échantillonnage de motifs en deux étapes qui vise à tirer au hasard un motif séquentiel X à partir d'un langage $\mathcal{L}_S$ proportionnellement à la fréquence pondérée par une mesure d'utilité fondée sur la norme.

Définition 2 *La fréquence d'une sous-séquence $X \in \mathcal{L}_S$ dans la base de données séquentielles $\mathcal{S}$, notée par $freq(X, \mathcal{S})$, est définie par : $freq(X, \mathcal{S}) = |\{X' \in \mathcal{S} : X \sqsubseteq X'\}|$. Une mesure d'utilité u est dite fondée sur la norme si et seulement s'il existe une fonction $f_u : \mathbb{N} \to \mathbb{R}$ tel que pour tout motif $X \in \mathcal{L}_S$, on a $u(X) = f_u(\|X\|)$.*

Dans la suite de ce papier, $X \sim \mathbf{P}(\mathcal{L})$ dénote un motif où $\mathbf{P}(.)$ est une distribution de probabilité sur $\mathcal{L}_S$ définie par $\mathbf{P}(X) = \frac{freq(X,\mathcal{S}) \times u(X)}{\sum_{X' \in \mathcal{L}_S} freq(X',\mathcal{S}) \times u(X')}$.

D'abord, NUSSAMPLING commence par un prétraitement de la base de données correspondant à la pondération des séquences. La complexité du prétraitement est en $O(|\mathcal{S}| \cdot L \cdot M^2 \cdot 2^P \cdot T)$ où L est la longueur maximale d'une séquence , M est la contrainte de norme maximale, P est la taille maximale des ensembles de positions $L(\gamma^{i-1}, s[i])$ et T est la taille maximale d'un itemset dans une séquence. Ensuite, il débute la phase d'échantillonnage par le tirage d'une séquence proportionnellement à la somme des utilités des sous-séquences qu'elle contient en $O(\log(|\mathcal{S}|))$. Enfin, une sous-séquence de la séquence précédemment tirée est retournée aléatoirement avec une probabilité proportionnelle à son utilité. Pour réaliser ce dernier tirage, il utilise une méthode d'échantillonnage par rejet et considère la première occurrence de chaque sous-séquence dans la séquence afin d'éviter de biaiser le tirage.

En pratique, si le temps de tirage reste efficace (en millisecondes) malgré la méthode de rejet, même avec des jeux de données volumineux, la phase de prétraitement, quant à elle, peut être très coûteuse. En effet, le temps de prétraitement augmente proportionnellement à la taille

de la base de données. Cela peut être un réel problème pour les grands ensembles de données réels, comme l'ont montré les expériences (Diop et al., 2019). Pour faire face à cette difficulté, nous proposons dans la section 3 un algorithme distribué et parallèle de NUSSAMPLING.

2.3 Le modèle BSP de programmation parallèle

Le modèle BSP (*Bulk Synchronous Parallel, Valiant (1990)*) est un modèle de programmation parallèle, avec une communication par passage de messages, développé pour la parallélisation de tâches sur plusieurs nœuds en vue d'une meilleure scalabilité. Il offre un haut degré d'abstraction et est défini par la combinaison des trois éléments suivants : (i) un ensemble d'unités (ou composantes ou paires processeur-mémoires) de calcul pour l'exécution des tâches, (ii) un système de communication permettant l'échange de messages entre ces unités de calcul et (iii) des *supersteps* pour synchroniser les tâches effectuées par les unités de calcul. Pendant un *superstep*, chaque composante exécute une tâche consistant en un traitement local et des réceptions et envois de messages. Une fois qu'un *superstep* est terminé pour toutes les composantes, ces dernières passent au *superstep* suivant (barrière de synchronisation). Un processus BSP consiste en une séquences de tels *supersteps*.

Pregel (Malewicz et al., 2010) est l'une des premières implémentations de BSP. Il offre une bibliothèque pour la programmation d'algorithmes de graphes, tout en rendant transparents les détails des communications sous-jacentes. Son paradigme de programmation peut être caractérisé par le concept "penser comme un nœud" (*think like a vertex*). Les traitements sur les graphes sont en effet définis en termes de ce que chaque nœud doit faire ; et les arcs des graphes sont les canaux de communication pour la transmission de données d'un nœud à un autre. Dans chaque *superstep*, un nœud peut exécuter une fonction définie par l'utilisateur (appelée `compute()`), envoyer ou recevoir des messages, et changer son état du statut actif à inactif. La barrière de synchronisation garantira qu'un message envoyé pendant un *superstep* S parviendra aux nœuds destinataires lors du *superstep* suivant $S+1$. Un nœud peut demander à être inactif (s'il appelle la primitive `voteToHalt()`), mais il sera réveillé (actif) lorsqu'il recevra un message. Un programme Pregel s'arrête une fois que tous les nœuds seront inactifs.

3 Parallélisation de l'échantillonnage de motifs séquentiels

Etant donné un ensemble de données séquentielles, l'algorithme 1 donne la fonction `compute()` de notre proposition concernant un échantillonnage distribué avec Pregel. Nous rappelons que cette fonction sera exécutée par chaque nœud actif durant le fonctionnement du système. Dans la mesure où nous nous intéressons à une structure de graphe, les nœuds et les arcs de Pregel correspondent respectivement aux données et canaux de communication. Un nœud peut être primaire (PN) ou secondaire (SN). Les nœuds secondaires contiennent toutes les données en entrée, c'est à dire les données séquentielles. Ils ont aussi le rôle de stocker les résultats de l'échantillonnage de motifs. Par conséquent, chaque SN représente une partition de données. Les SNs sont aussi dotés d'une fonction d'utilité fondée sur la norme u. L'unique nœud primaire PN contient le nombre N de motifs souhaités et il est par ailleurs responsable de coordonner la découverte de motifs. De ce fait, il y a un arc bidirectionnel entre le PN et chaque SN. L'algorithme 1 consiste en seulement trois *supersteps*. Dans le *superstep* 0, le PN et tous les SNs sont actifs, mais seuls les SNs vont exécuter un code avant de demander à être

Algorithm 1 $compute(vertex, messages)$

// $vertex$ représente un nœud primaire (PN) ou un nœud secondaire (SN)

`-- SUPERSTEP 0 --`

1: **if** $(vertex.isSecondaryNode())$ **then**
2: $w \leftarrow \sum_i \left(\sum_\ell \Phi_{[\ell..\ell]}(vertex.Data[i]) \times f_u(\ell)\right)$ ▷ Pondération de séquences
3: $sendMessage(PN.\texttt{id}, \langle vertex.getID(), w\rangle)$

`-- SUPERSTEP 1 --`

// Seul PN est actif

4: $w_i \leftarrow messages.getWeight(SN_i)$
5: $Z \leftarrow \sum w_i$
6: $N \leftarrow vertex.getSampleSize()$
7: **for** $k = 1$ to N **do**
8: $x \leftarrow random() \times Z$
9: Trouver j tel que : $\sum_{i=1}^{j-1} w_i < x \leq \sum_{i=1}^{j} w_i$
10: $sendMessage(SN_j.\texttt{id}, x - \sum_{i=1}^{j-1} w_i)$

`-- SUPERSTEP 2 --`

11: **if** $(vertex.isSecondaryNode())$ **then**
12: **for** x in $messages$ **do**
13: Trouver j tel que : $\sum_{i=1}^{j-1} w(vertex.Data[i]) < x \leq \sum_{i=1}^{j} w(vertex.Data[i])$
14: $\gamma_j = vertex.Data[j]$
15: $X \leftarrow randomPattern(\gamma_j, u)$
16: $vertex.Sample.add(X)$
17: $vertex.voteToHalt()$; ▷ Fin de la fonction compute()

18: Function $randomPattern(\gamma, u)$:
19: Calcule le poids défini par $w^\ell(\gamma) \leftarrow \Phi_{[\ell..\ell]}(\gamma) \times f_u(\ell)$ pour tout $\ell \in [0..\|\ell\|]$
20: Tire un entier ℓ proportionnellement à $w^\ell(\gamma)$
21: $X \sim unif(\{X \sqsubseteq \gamma : \|X\| = \ell\})$
22: return X

inactifs (lignes 1 à 3). Chaque SN procède à une pondération de séquences et envoie au PN son poids total ainsi que son identifiant (ID). Dans le *superstep* suivant (*superstep* 1), seul le PN est actif dans la mesure où les SNs lui ont envoyé des messages pendant le précédent *superstep*. Pour chacun des N motifs souhaités (ligne 7), le PN choisit un poids au hasard (ligne 8), en déduit le SN_j correspondant (ligne 9) et envoie (ligne 10) une valeur appropriée de telle sorte que, pendant le *superstep* suivante, SN_j sera capable de générer un motif correspondant au poids aléatoire. Dans le dernier *superstep* (*superstep* 2), seuls certains SNs sont actifs. Il s'agit de ceux qui ont reçu des messages de PN envoyés lors du précédent *superstep*. De ce fait, pour chaque message, un SN va générer un échantillon avec la routine prédéfinie (*"randomPattern()"*) qui utilise la séquence γ_j correspondante et la fonction d'utilité u. Le *superstep* suivant ne sert qu'à se rendre compte qu'il n'y a plus de nœud actif et donc va permettre au système entier de s'arrêter. Les N motifs stockés sur les SNs seront écrits sur HDFS.

Exemple 2 *Considérons les quatre données séquentielles dans la Figure 1. Nous décrivons les trois supersteps d'un échantillonage avec un objectif de 10 motifs. Nous supposons que nous avons deux SNs (SN_1 et SN_2) et un PN, et la partition des données est telle que les séquences 1 et 2 sont stockées dans SN_1 et le reste dans SN_2. Dans le superstep 0, SN_1 et SN_2 font des*

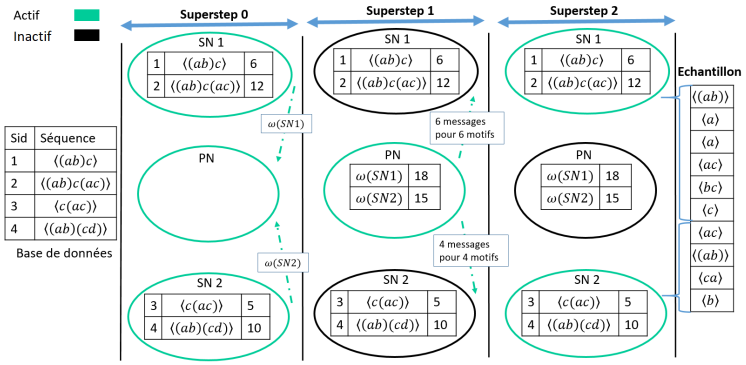

FIG. 1 – *Exemple du processus d'échantillonage*

pondérations et envoient leurs poids totaux au PN. Ces messages seront reçus par le PN au superstep 1 durant lequel il est le seul nœud actif. Ensuite, dans notre exemple, le PN envoie les informations appropriés pour tirer 10 motifs : 6 messages à SN_1 et 4 à SN_2. Dans le superstep suivant, chaque SN trouve et stocke autant de motifs que de messages qu'il a reçu de PN. Notons que certains motifs peuvent bien être identiques.

4 Analyse

La propriété 1 montre que l'algorithme 1 retourne un échantillon exact de motifs.

Propriété 1 (Correction) *Soient $\mathcal{S}$ une base de données séquentielles fragmentée et répartie sur k noeuds secondaires et u une mesure d'utilité fondée sur la norme. L'algorithme 1 tire un motif du langage $\mathcal{L}_S$ proportionnellement à sa fréquence dans $\mathcal{S}$ multipliée par son utilité.*

Preuve 1 *Soit Z une constante de normalisation définie par $Z = \sum_{\gamma \in \mathcal{S}} w(\gamma) = \sum_{j=1}^{k} w_j$ où $w_j = \sum_i (\sum_\ell \Phi_{[\ell..\ell]}(SN_j.Data[i]) \times f_u(\ell))$ est le poids d'un noeud secondaire qui est la somme des poids des séquences qu'il contient. Soit X une sous-séquence dans $\mathcal{L}_S$ et $\mathbf{P}(X)$ la probabilité de tirer le motif X avec l'algorithme 1. D'après les lignes 4 à 10, on a $\mathbf{P}(X) = \sum_j^k \frac{w_j}{Z} \times \mathbf{P}(X/SN_j)$. On sait que $\mathbf{P}(X/SN_j) = \sum_{i, X \sqsubseteq SN_j.Data[i]} \mathbf{P}(SN_j.Data[i]) \times \mathbf{P}(X/SN_j.Data[i])$. Si $\|X\| = m$, alors d'après la ligne 18, nous avons $\mathbf{P}(X/SN_j) = \sum_{i, X \sqsubseteq SN_j.Data[i]} \frac{\sum_\ell \Phi_{[\ell..\ell]}(SN_j.Data[i])}{w_j} \times \mathbf{P}(m/SN_j.Data[i]) \times \mathbf{P}(X/m, SN_j.Data[i])$. En faisant une simplification, nous avons $\mathbf{P}(X/SN_j) = \sum_{i, X \sqsubseteq SN_j.Data[i]} \frac{f_u(m)}{w_j} = \frac{freq(X, SN_j) \times f_u(m)}{w_j}$. Il en résulte que $\mathbf{P}(X) = \sum_j^k \frac{w_j}{Z} \times \frac{freq(X, SN_j) \times f_u(m)}{w_j} = \sum_j^k \frac{freq(X, SN_j) \times f_u(m)}{Z}$. Or $\sum_j^k freq(X, SN_j) = freq(X, \mathcal{S})$, donc $\mathbf{P}(X) = \frac{freq(X, \mathcal{S}) \times f_u(m)}{Z}$. D'ou le résultat.*

Dans la suite de cette section, nous donnons la complexité de notre solution, c'est à dire le nombre d'itérations et d'échanges de messages entre unités de calcul ou composantes. En général, le coût d'une solution basée sur BSP est le nombre de *supersteps* nécessaires multiplié

par le coût d'un *superstep*. Par contre, dans ce cas présent, notre solution consiste en seulement trois *supersteps* hétérogènes, et le coût global est par conséquent la somme des coûts respectifs.

Le coût d'un *superstep* est déterminé par la somme de trois termes ; le coût de la plus longue exécution locale, le coût du maximum de communication entre unités de calcul et le coût de la barrière de synchronisation à la fin des *supersteps*. Les coûts sont calculés en termes de paramètres abstraits modélisant le nombre d'unités de calcul ou processeurs p, le coût de la barrière de synchronisation l, le coût de l'exécution locale e_i, le nombre h_i de messages envoyés et reçus par le processeur p_i, et l'aptitude g qu'a le réseaux de communication à transmettre les données, définie de telle sorte qu'il faudra $h_i g$ unités de temps au processeur p_i pour transmettre h_i messages. Ainsi, le coût est $max_{i=1}^{p}(e_i) + max_{i=1}^{p}(h_i g) + l$. Cette expression est plus couramment notée par $e + hg + l$, e et h étant les maximums.

En ce qui concerne notre travail, nous supposons que les n séquences en entrée sont uniformément distribuées sur les k SNs (nœuds secondaires). Ainsi, chaque SN comportera n/k séquences. Les coûts respectifs des *supersteps* sont définis comme suit.

Superstep 0 : Le PN et les tous les SNs sont actifs, mais seuls ces derniers vont exécuter du code pour des pondérations de séquences et des envois de messages au PN. Il est clair que la complexité du prétraitement de NUSSAMPLING est divisée par le nombre de noeuds secondaires car la base de données initiale est fragmentée horizontalement et uniformément sur lesdits noeuds. Ainsi, e et h sont respectivement $O(\frac{|\mathcal{S}|}{k} \cdot L \cdot M^2 \cdot 2^P \cdot T)$ et $O(1)$ puisque chaque SN enverra un message contenant son poids.

Superstep 1 : Seul le PN est actif, comme cela a été décrit dans la section précédente. D'abord, il commence par construire une matrice de pondérations des noeuds secondaires en $O(k)$. Ensuite, pour le tirage d'un motif, le PN tire au hasard un SN proportionnellement à son poids en $O(\log(k))$. Enfin, de la valeur précédemment tiré, on en déduit celle qui correspond à la séquence dans PN où on va tirer le motif au *superstep* suivant. Au final, les valeurs de e et h sont respectivement $O(k + N \times \log(k))$ et $O(N)$, N étant le nombre souhaité de motifs.

Superstep 2 : Seuls certains SNs sont actifs, c'est à dire ceux qui ont reçu des messages du PN lors du précédent *superstep*. La complexité du tirage d'un motif par un noeud peut être scindée en deux étapes : (i) le tirage d'une séquence qui se fait en $O(\log(\frac{|\mathcal{S}|}{k}))$, (ii) la complexité du tirage d'un motif séquentiel qui n'est pas théoriquement estimable car le tirage se base sur du rejet. Dans (Diop et al., 2019) les auteurs ont montré l'efficacité de leur méthode par rejet en calculant le nombre moyen de tirages, noté $\mu_{[m..M]}(\mathcal{S})$, pour obtenir une première occurrence d'une sous-séquence de norme comprise entre m et M. La complexité de tirage d'une occurrence de motif de norme comprise entre m et M étant en $O(M^2)$, donc le coût d'exécution local e est $O(N \times (\log(\frac{|\mathcal{S}|}{k}) + M^2 \times \mu_{[m..M]}(\mathcal{S})))$ vu que les N motifs souhaités peuvent concerner un unique SN au pire des cas. Toutefois, aucun message ne sera envoyé pendant ce dernier *superstep* ($h = 0$).

5 Conclusion

Cet article propose une version parallélisée de l'algorithme NUSSAMPLING pour l'échantillonnage de motifs séquentiels selon une mesure d'utilité fondée sur la norme, et distribuée sur différents sites ou unités de calcul. Par conséquent, plus nous en avons, plus notre solution est rapide. Nous avons démontré qu'elle est exacte et estimé son efficacité en fonction du nombre

d'unités de traitement utilisés. Notre approche montre que la méthode de programmation BSP est très efficace pour la parallélisation des traitements. En effet, elle est très parcimonieuse en coût de nombre de *supersteps* qui est égale à 3 indépendamment de tout jeu de données.

En perspective, nous voudrions appliquer notre méthode sur l'échantillonnage d'HUI (High Utility Itemset) où la phase de prétraitement ainsi que la phase de tirage d'un motif risquent d'être très coûteuses sur les grosses bases de données transactionnelles.

Références

Agrawal, R. et R. Srikant (1995). Mining sequential patterns. In *Proc. of ICDE 95*, pp. 3–14.

Al Hasan, M. et M. J. Zaki (2009). Output space sampling for graph patterns. *Proc. of the VLDB Endowment 2*(1), 730–741.

Ba, C. et A. Gueye (2020). A BSP based approach for nfas intersection. In *Algorithms and Architectures for Parallel Processing - 20th International Conference, ICA3PP 2020, New York City, NY, USA*.

Boley, M., C. Lucchese, D. Paurat, et T. Gärtner (2011). Direct local pattern sampling by efficient two-step random procedures. In *Proc. of KDD*, pp. 582–590.

Dean, J. et S. Ghemawat (2008). Mapreduce : Simplified data processing on large clusters. *Commun. ACM 51*(1), 107–113.

Diop, L., C. T. Diop, A. Giacometti, D. L. Haoyuan, et A. Soulet (2018). Sequential pattern sampling with norm constraints. In *Proc. of ICDM 2018*.

Diop, L., C. T. Diop, A. Giacometti, D. Li, et A. Soulet (2019). Sequential pattern sampling with norm-based utility. *Knowledge and Information Systems 62*, 2029–2065.

Diop, L., C. T. Diop, A. Giacometti, et A. Soulet (2020). Pattern sampling in distributed databases. In *Advances in Databases and Information Systems*, Cham, pp. 60–74. Springer International Publishing.

Fournier Viger, P., C.-W. Lin, U. Rage, Y. S. Koh, et R. Thomas (2017). A survey of sequential pattern mining. *Data Science and Pattern Recognition 1*, 54–77.

Gonzalez, J. E., Y. Low, H. Gu, D. Bickson, et C. Guestrin (2012). Powergraph : Distributed graph-parallel computation on natural graphs. In *OSDI 2012, Hollywood, CA, USA, October 8-10*, pp. 17–30.

Gonzalez, J. E., R. S. Xin, A. Dave, D. Crankshaw, M. J. Franklin, et I. Stoica (2014). Graphx : Graph processing in a distributed dataflow framework. In *OSDI '14, Broomfield, CO, USA, October 6-8*.

Malewicz, G., M. H. Austern, A. J. C. Bik, J. C. Dehnert, I. Horn, N. Leiser, et G. Czajkowski (2010). Pregel : a system for large-scale graph processing. In *Proceedings of the ACM SIGMOD International Conference on Management of Data, Indianapolis, Indiana, USA, June 6-10*, pp. 135–146.

Valiant, L. G. (1990). A bridging model for parallel computation. *Commun. ACM 33*(8), 103–111.

Summary

In recent years, the field of data mining has seen significant works on pattern discovery by output sampling. Very recently, these sampling methods have been applied to sequential data which is of a complex nature. The complexity of these data lies in their structure which has a notorious impact on the speed of computation and in particular on the preprocessing. In this paper, we have shown how to take advantage of the BSP (Bulk Synchronous Parallel) programming model to improve the efficiency of output sampling methods on sequential data.