Alignement et partitionnement de chaînes de caractères à l'échelle du motif

Maxime Raynal, Marc-Olivier Buob, Georges Quénot

Alignement et partitionnement de chaînes de caractères à l'échelle du motif

Maxime Raynal[1 2] [†] et Marc-Olivier Buob[1] et Georges Quénot[2]

[1] *Nokia Bell Labs, France*
[2] *Univ. Grenoble Alpes, CNRS, Grenoble INP, LIG, F-38000 Grenoble France*

Le traitement de logs réseaux passe souvent par le regroupement d'alarmes similaires afin d'en faciliter l'analyse. Ce travail présente un algorithme de partitionnement basé sur une nouvelle distance d'édition. Alors que les distances d'édition usuelles comparent deux chaînes caractère par caractère, notre approche les compare à l'échelle du motif. Pour cela, nous proposons un modèle mathématique alliant les domaines de la théorie des langages, de la programmation dynamique et des algèbres de chemins. Enfin, nous évaluons notre proposition sur des données synthétiques et réelles.

Mots-clefs : distance d'édition, partitionnement de log, algèbre de chemins, théorie des langages, automates

1 Introduction

Dans ce travail, nous cherchons à regrouper ensemble les lignes similaires d'un journal d'erreurs (*log*), typiquement produit par des équipements réseau. Ce traitement facilite le chargement et l'analyse des données qu'il contient, notamment pour identifier l'origine d'une panne [ZHL+19]. Dans ce but, nous nous intéressons aux distances d'éditions. Celles-ci sont communément utilisées en informatique pour la recherche approximative de chaîne de caractères et en bio-informatique pour comparer des séquences d'ADN. Les distances d'éditions sont construites sur des opérations d'édition, permettant de transformer localement un mot en un autre. Par exemple, la distance de LCS (Longest Common Distance) [Mai78] utilise l'ajout et la suppression de caractères. La distance de Levenshtein considère en plus le replacement de caractère [Lev66]. Le nombre minimal d'opérations requises pour transformer une chaîne en une autre détermine leur distance.

Par construction, les distances d'éditions opèrent à l'échelle du caractère. Or dans ce travail, nous cherchons à regrouper les lignes dont la structure est voisine. Nous proposons donc d'adapter les distances d'édition pour raisonner à l'échelle d'une collection restreinte de motifs prédéterminée. Nous montrons comment représenter une chaîne à cette échelle (automate de motifs), puis comment comparer deux chaînes (similarité par motifs). Enfin, nous montrons comment exploiter cette distance pour partitionner un fichier de log, et évaluons la performance de notre algorithme par simulation.

Certains travaux s'intéressent à des problèmes voisins. [Ars07] cherche le meilleur alignement entre deux séquences conforme à une expression rationnelle donnée. Mais dans notre cas, les expressions sont a priori inconnues. [JLMZ02] présente un modèle de distance d'édition impliquant des opérations opérant sur plusieurs caractères, mais ne raisonne pas à l'échelle de motifs prédéterminés.

2 Automate de motifs

Notations : Sauf indication contraire, tous les mots et automates sont définis sur un même alphabet Σ. On note ε le mot vide. Étant donné un mot w, on note w_j le $j^{\text{ième}}$ caractère de w et $w_{j:k}$ le sous-mot $w_j \dots w_{k-1}$. Si $j \geq k$, alors $w_{j:k} = \varepsilon$. L'étoile de Kleene est notée *, et pour tout langage L, nous posons $L^+ = LL^*$. Un automate fini non-déterministe (AFN) est un tuple $A = (\Sigma, Q, \delta, q_0, F)$ où Σ est son alphabet, Q l'ensemble de ses états, $\delta : Q \times \Sigma \to Q$ l'ensemble de ses transitions, $q_0 \in Q$ son état initial, $F \subseteq Q$ l'ensemble de ses états finaux. Enfin, pour $n \in \mathbb{N}$, on note $\mathbb{N}_n$ l'ensemble $\{0, \dots, n\}$.

[†] Une partie de ces travaux a été effectuée au LINCS (Laboratory of Information, Networking and Communication Sciences).

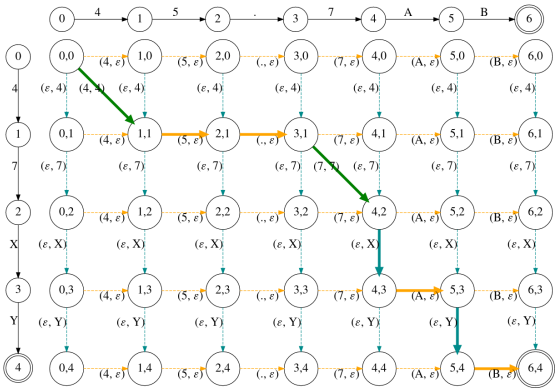

FIGURE 1: Graphe d'édition à l'échelle du caractère pour $w = \texttt{47XY}$ et $w' = \texttt{4.57AB}$. Le chemin en gras correspond à un alignement optimal (voir section 3).

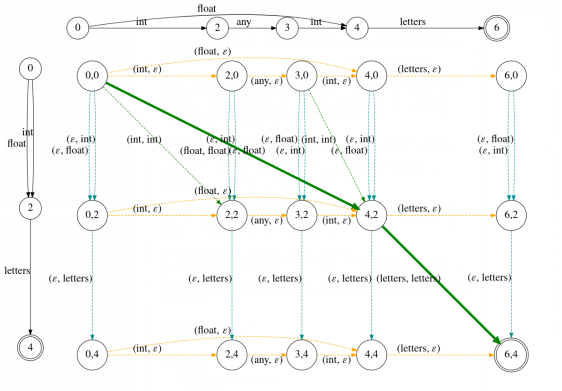

FIGURE 2: Graphe d'édition à l'échelle des motifs pour $\text{AM}(\texttt{47XY}, \mathscr{P})$ (à gauche) et $\text{AM}(\texttt{4.57AB}, \mathscr{P})$ (en haut), et avec $\mathscr{P}$ défini conformément à l'exemple 1.

Cette section montre comment représenter un mot arbitraire à l'échelle du motif à l'aide d'un automate. Nous appelons *motif* tout langage inclus dans Σ^* et *collection de motifs* tout ensemble $\mathscr{P} = \{P_0, \ldots, P_p\}$ tel que (i) $\forall P \in \mathscr{P}, P \subseteq \Sigma^+$ et (ii) $(\bigcup_{P \in \mathscr{P}} P)^+ = \Sigma^+$. Cette définition impose que tout mot non vide est décomposable en une suite de sous-mots non vides reconnus par un langage de $\mathscr{P}$.

Exemple 1 : Soit $\Sigma = \{\texttt{0}, \ldots, \texttt{9}, \texttt{A}, \ldots, \texttt{Z}, \texttt{.}\}$. L'ensemble $\mathscr{P}$ réunissant $P_{\text{int}} = \{\texttt{0}, \ldots, \texttt{9}\}^+$, $P_{\text{float}} = P_{\text{int}} \cup \{w.w' \mid w, w' \in P_{\text{int}}\}$, $P_{\text{letters}} = \{\texttt{A}, \ldots, \texttt{Z}\}^+$, $P_{\text{dot}} = \{\texttt{.}\}$ est une collection de motifs.

L'*automate de motifs* (AM) d'un mot w par rapport à la collection $\mathscr{P}$ est l'AFN noté $\text{AM}(w, \mathscr{P})$ et caractérisé par le tuple $(\mathscr{P}, \mathbb{N}_{|w|}, \delta, 0, \{|w|\})$, où $\delta(j, P) = k$ si et seulement si $w_{j:k} \in P$. Dans le cas général, cet automate est acyclique et non déterministe. Afin de simplifier l'AM, on filtre toute transition de la forme $\delta(j', P) = k'$ s'il existe une transition de type $\delta(j, P) = k$ avec $[j', k']$ strictement inclus dans $[j, k]$. On peut montrer que l'AM est alors déterministe. Si on l'émonde, il devient minimal et donc uniquement défini. Sans perte de généralité, on considère par la suite que les AM sont toujours ainsi construits.

3 Similarité par motif

Nous montrons à présent comment étendre l'algorithme de Needleman-Wunsch [NW70] aux automates de motifs. La figure 1 représente le *graphe d'édition* [Mai78] utilisé pour calculer la distance LCS à l'échelle du caractère. Chaque état correspond à la position de deux curseurs sur w et w'. Chaque arc horizontal (resp. vertical) revient à supprimer (resp. ajouter) un caractère de w (resp. de w') et a pour poids 1. Chaque arc diagonal signifie que le caractère de w correspond au (resp. est remplacé par) le caractère courant de w' et de poids 0 (resp. 1). Le poids d'un chemin comptabilise le nombre d'opérations d'édition appliquées. Tout chemin de poids minimal (en gras sur la figure 1) de $(0,0)$ à (w, w') correspond à un jeu minimal d'opérations d'édition pour transformer w en w' et est communément appelé *alignement* de w et w'.

Pour passer à l'échelle du motif, nous adaptons les opérations d'éditions par le remplacement d'un sous-mot de $P \in \mathscr{P}$ par un autre mot de P , la suppression et l'insertion d'un sous mot d'un motif de $\mathscr{P}$. Comme le montre la figure 2, le graphe d'édition et l'algorithme de Needleman-Wunsch s'étendent naturellement en raisonnant sur les AMs des deux mots comparés. Plus formellement, soient deux mots w et w' et une collection $\mathscr{P}$. Leurs AMs $A = (Q, \mathscr{P}, \delta, 0, \{|w|\})$ et $A' = (Q, \mathscr{P}, \delta, 0, \{|w'|\})$ induisent le graphe d'édition $\mathscr{E}(A, A')$ correspondant à l'AFN $(Q \times Q', (\mathscr{P} \cup \{\varepsilon\})^2, \delta_{\mathscr{E}}, (0,0), \{(|w|, |w'|)\})$, où :

— $\delta_{\mathscr{E}}((j, j'), (P, \varepsilon) = (k, j')$ si $w_{j:k} \in P$ (suppression) ;
— $\delta_{\mathscr{E}}((j, j'), (\varepsilon, P') = (j, k')$ si $w'_{j':k'} \in P'$ (insertion) ;
— $\delta_{\mathscr{E}}((j, j'), (P, P') = (k, k')$ si $w_{j:k} \in P$ et $w'_{j':k'} \in P'$ (modification ou concordance) ;

avec $k = \delta(j, P)$ et $k' = \delta'(j', P)$.

On peut montrer que si A et A' sont déterministes (resp. minimaux), alors $\mathscr{E}(A, A')$ l'est également.

Il ne reste plus qu'à pondérer les arcs du graphe d'édition à l'aide d'une fonction c à valeur dans un espace de poids noté E. On peut par exemple utiliser la fonction de coût, suivante, où $k = \delta(j,P)$ et $k' = \delta(j,P)$:

$$c : ((j,j),(P,P')) \mapsto \begin{cases} 0 & \text{si } P = P' \wedge w_{j:k} = w'_{j':k'} \\ \rho(P) & \text{si } P = P' \wedge w_{j:k} \neq w'_{j':k'} \\ 1 & \text{si } P = \varepsilon \vee P' = \varepsilon \\ 1 & \text{sinon} \end{cases} \text{ avec } \rho(P) = \sum_{i \in \mathbb{N}^*} \frac{|P|_i}{2^i \cdot |\Sigma|^i}$$

Intuitivement, $\rho(P)$ reflète la densité d'un langage P : elle tend vers 0 si P tend vers $\emptyset$ et vers 1 si P tend vers Σ^*. Par exemple, $\rho(P_{\text{int}}) < \rho(P_{\text{float}})$. Ainsi, c favorise l'alignement des motifs les plus stricts.

Pour le meilleur alignement, on peut utiliser la programmation dynamique. Si l'on voit le problème comme un calcul de plus courts chemins, on peut définir une structure algébrique $(E, \oplus, \otimes)$, où $\oplus$ est l'opérateur binaire qui sélectionne le meilleur point et où $\otimes$ concatène deux poids. Si $(E, \oplus, \otimes)$ un semi-anneau, un alignement optimal peut être calculé plus efficacement avec l'algorithme de Dijkstra généralisé [GM08]. Par la suite, nous choisissons le semi-anneau $(\mathbb{R}^+ \cup \{+\infty\}, \min, +)$. Nous obtenons ainsi notre *similarité par motifs* (SM), notée d. Il est utile de préciser qu'une distance d'édition n'est pas toujours une distance au sens algébrique. C'est le cas de d, mais aussi des distances de Damerau-Levenshtein et de Jaro.

4 Partitionnement par motifs

Soit W un ensemble de mots arbitraires (e.g. les lignes d'un fichier de log) et $D \in E$ un seuil paramètre de l'algorithme. En s'inspirant de l'algorithme des plus proches voisins, la similarité par motif nous permet de dériver l'algorithme de partitionnement glouton suivant. Au cours de cet algorithme, chaque partition C est représentée par l'AM du premier mot qu'on y insère, noté $r(C)$. À l'initialisation, on assigne un élément arbitraire de W à une première partition. À chaque itération, on choisit un mot w qui n'a pas encore été assigné à une partition. On calcule $\text{AM}(w, \mathscr{P})$ et on détermine la partition C dont le représentant minimise $d(r(C), A)$ et est inférieur à D au sens de $\oplus$. Si C existe, on ajoute w à C. Sinon, on assigne w à une nouvelle partition. Lorsque tous les mots de W sont traités, on obtient une partition de W.

Si $(E, \oplus, \otimes)$ est un semi-anneau, nous proposons l'optimisation suivante. Soit G le graphe obtenu : (i) en réunissant les graphes d'éditions $\mathscr{E}(r(C), w)$ pour chaque partition C actuellement formée ; (ii) en ajoutant un sommet s et en connectant s aux états initiaux des graphes d'éditions par une ε-transition de poids nul ; (iii) en définissant s comme l'état initial de l'automate ainsi obtenu. Nous exécutons l'algorithme de Dijkstra sur G depuis s, jusqu'à traiter un état final ou un état dont la distance depuis s dépasse D. Si un tel état final est trouvé, son graphe d'édition $\mathscr{E}(A, r(C))$ permet de retrouver la partition C à laquelle ajouter w. Sinon, on assigne w à une nouvelle partition. Pour économiser de la mémoire et des calculs inutiles, nous construisons G à la volée, au cours de son exploration par l'algorithme de Dijkstra.

5 Résultats expérimentaux

Nous commençons par évaluer le partitionnement par motif sur un jeu de données synthétique, pour lequel on génère N lignes conformes à un patron de ligne parmi $K = 40$, et où chaque patron implique jusqu'à $L = 20$ motifs parmi ceux de l'exemple 1.

La figure 3 montre le temps d'exécution obtenu pour l'algorithme de Needleman-Wunsch avec une implémentation à l'aide de tableaux (NW) et la similarité par motifs (SM). NW-FW montre les résultats qu'on obtiendrait si, comme pour SM, on utilisait la même librairie de graphe. L'écart entre NW et NW-FW montre l'impact de l'implémentation. SM est sensiblement plus rapide que NW-FW, car le graphe d'édition traité est plus petit et n'est exploré que partiellement. D'autres expériences montrent que la performance du partitionnement par motif est en $O(K.N.L^2)$, où $K.N$ résulte de l'algorithme de partitionnement et L^2 de la similarité par motif. La figure 4 compare la *pattern accuracy* [ZHL+19] et l'index de Rand ajusté obtenus. La connaissance de $\mathscr{P}$ permet à SM d'avoir des résultats bien meilleurs que NW.

Enfin, nous comparons le partitionnement basé sur SM sur les fichiers de logs réels `OpenStack` and `openssh` de LogHub[‡]. Chaque fichier comporte 2 000 lignes est accompagné d'une vérité terrain qui

‡. `https://github.com/logpai/logparser`

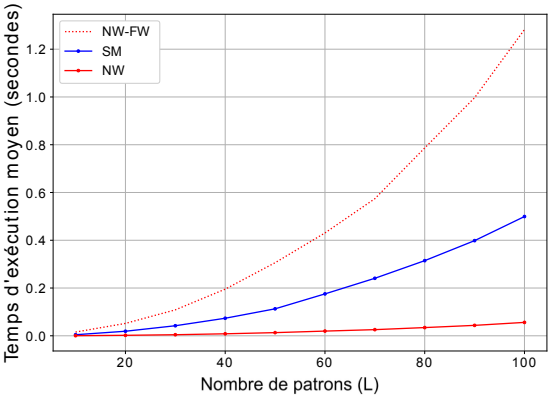

FIGURE 3: Temps d'exécution de la similarité par motifs.

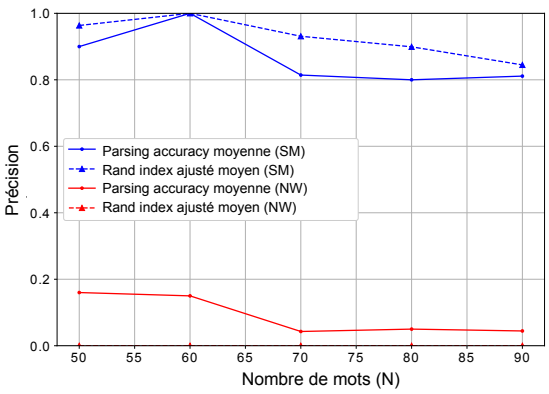

FIGURE 4: Précision du partitionnement par motifs.

associe à chaque ligne une partition. Nous obtenons des résultats au-dessus de la moyenne par rapport aux approches concurrentes présentées dans [ZHL+19]. Le temps d'exécution de notre approche est cependant beaucoup plus élevé (de l'ordre d'un facteur 100). En effet, les approches concurrentes se basent sur des algorithmes plus rapides, mais qui font en contrepartie bien plus d'hypothèses sur le format du fichier.

6 Conclusion

Nous avons présenté une distance d'édition permettant de regrouper des chaînes de caractères similaires à l'échelle du motif en étendant l'algorithme de Needleman-Wunsch sur des automates. Nous l'avons ensuite exploité pour en tirer un algorithme de partitionnement par motifs. Nous avons montré sous quelles conditions et comment accélérer leurs calculs à l'aide de l'algorithme de Dijkstra. Nos expérimentations montrent que l'on aboutit à un partitionnement de log précis et ne nécessitant qu'une connaissance préalable limitée sur la nature des données traitées. Ce partitionnement facilite le chargement et l'analyse de données et réseau, et laisse donc espérer à terme l'émergence d'outils génériques pour la surveillance et l'orchestration d'équipements mis en réseau.

Références

[Ars07] Abdullah N. Arslan. Regular expression constrained sequence alignment. *Journal of Discrete Algorithms*, 5(4) :647–661, 2007.

[GM08] Michel Gondran and Michel Minoux. *Graphs, dioids and semirings : new models and algorithms*, volume 41. Springer Science & Business Media, 2008.

[JLMZ02] Tao Jiang, Guohui Lin, Bin Ma, and Kaizhong Zhang. A general edit distance between rna structures. *Journal of computational biology*, 9(2) :371–388, 2002.

[Lev66] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet physics doklady*, volume 10, pages 707–710, 1966.

[Mai78] David Maier. The complexity of some problems on subsequences and supersequences. *Journal of the ACM (JACM)*, 25(2) :322–336, 1978.

[NW70] Saul B. Needleman and Christian D Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of molecular biology*, 48(3) :443–453, 1970.

[ZHL+19] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R Lyu. Tools and benchmarks for automated log parsing. In *2019 IEEE/ACM 41st International Conference on Software Engineering : Software Engineering in Practice (ICSE-SEIP)*, pages 121–130. IEEE, 2019.