DIG-DAG: stockage et recherche de motifs dans un flux d'événements

Anne Bouillard, Marc-Olivier Buob, Achille Salaün, Maxime Raynal

DIG-DAG: stockage et recherche de motifs dans un flux d'événements

Anne Bouillard [†1], Marc-Olivier Buob[1], Achille Salaün[12] et Maxime Raynal[13]

[1]*Nokia Bell Labs, France*
[2]*Telecom Sud Paris, France*
[3]*Université de Grenoble, France*

La recherche de motifs dans une chaîne de caractères se réalise facilement avec un outil tel que `grep`. Dans cet article, nous considérons un flux d'événements caractérisés par une date de début et de fin. La recherche de motifs dans une telle structure devient alors plus complexe. Afin de réaliser efficacement cette opération, nous proposons une nouvelle structure, appelée DIG-DAG (*Directed Interval Graph - Directed Acyclic Graph*). Nous montrons comment construire de manière compacte cette structure lorsque le flux est découvert à la volée. Nous expliquons ensuite comment extraire du DIG-DAG les motifs conformes à la requête d'un utilisateur. Enfin, nous illustrons l'utilisation de ces algorithmes sur des traces réelles issues de réseaux GSM. Cet article synthétise deux articles publiés dans CNSM'2018 et MLN'2019.

Mots-clefs : Recherche de motifs, algorithmes en ligne, analyse de traces

1 Introduction

La recherche de motifs est un problème courant lorsqu'on manipule du texte, notamment pour détecter ou extraire certaines sous-chaînes de caractères d'un grand fichier. Par exemple, un outil tel que `grep` cherche les sous-chaînes conformes à une expression rationnelle dans un flux de caractères.

Dans cet article, nous nous intéressons à la recherche de motifs dans un flux d'événements temporisés, dans lequel la dimension temporelle définit une relation de causalité. Plusieurs chaînes de causalité peuvent alors coexister entre deux événements. Dès lors, la recherche de motifs dans une telle structure devient plus complexe que dans un flux de caractères.

Plus concrètement, ce besoin apparaît quand on souhaite retrouver la cause racine d'un incident, ou lorsqu'on veut compter le nombre d'occurrences d'un motif dans un flux.

Le but de ce travail est de proposer un algorithme capable de retrouver efficacement les chemins de causalité conformes à un motif donné. Pour cela, nous proposons de stocker de manière compacte les motifs observés dans le flux à l'aide une structure construite en ligne, appelée DIG-DAG (*Directed Interval Graph - Directed Acyclic Graph*). Cet article apporte les contributions suivantes :

— après avoir défini notre structure de causalité, nous montrons comment construire en ligne le DIG-DAG correspondant ;
— ensuite, nous expliquons comment exploiter le DIG-DAG pour extraire les chemins de causalités conformes à un motif exprimé par l'utilisateur ;
— enfin, nous évaluons notre proposition sur des traces réelles issues de réseaux GSM.

2 État de l'art

Dans [LVM18], Latapy et al. présentent une structure appelée *stream graph*, dans laquelle sommets et arcs peuvent apparaître et disparaître au cours du temps. On peut voir un *stream graph* orienté comme un

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

moyen de représenter une structure de causalité. [LVM18] explore comment généraliser certaines primitives de la théorie des graphes, mais ne montre pas comment chercher un chemin dans une telle structure.

L'algorithme d'Ukkonnen [Ukk95] construit en temps linéaire un arbre de suffixes, à partir d'une chaîne de caractères découverte à la volée. Il permet donc de mémoriser tous les suffixes observés. Ce travail a servi de point de départ à l'algorithme de construction du DIG-DAG afin de garder trace des chemins de causalités observés dans un flux.

L'algorithme d'Aho-Corasick [AC75] permet de rechercher dans un texte les occurrences d'un ensemble fini de mots. L'algorithme de `grep` [Tho68] permet de chercher des facteurs conformes à une expression rationnelle. Dans ces deux travaux, le motif cherché est traduit sous la forme d'une structure proche d'un automate fini déterministe. Celle-ci permet d'extraire les facteurs désirés en une passe sur le texte.

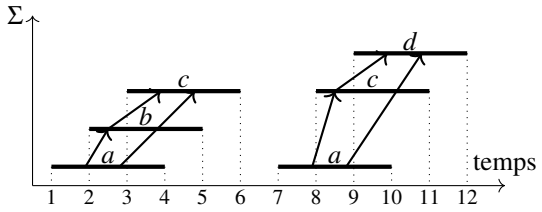
3 Représentation d'un flux d'événements par un DIG

Dans ce travail, nous supposons que chaque événement e est caractérisé par une étiquette $a \in \Sigma$, une date de début $s \in \mathbb{R}^+$ et une date de fin $t \in]s, +\infty[$. Un tel événement est noté $e = (a, [s,t])$ et $\mathcal{E}$ désigne l'ensemble des événements observés dans le flux. On suppose que pour tous $e = (a, [s,t]), e' = (a', [s',t']) \in \mathcal{E}$ les valeurs de s, t, s' et t' sont deux à deux distinctes (H1). De plus, si $a = a'$, nous imposons que $[s,t] \cap [s',t'] = \emptyset$ (H2). On note la fonction $\lambda : \mathcal{E} \to \Sigma$ la fonction qui permet d'extraire l'étiquette d'un événement ($\lambda(e) = a$).

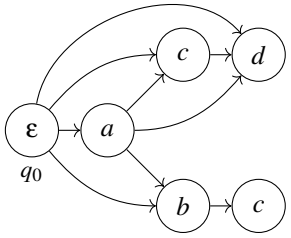
On peut maintenant définir une relation de causalité $\to$ entre événements, par exemple $(a, [s,t]) \to (a', [s',t'])$ si et seulement si $s < s'$ et $[s,t] \cap [s',t'] \neq \emptyset$. La structure ainsi obtenue est acyclique.

Le triplet $(\mathcal{E}, \to, \lambda)$ définit un DIG (*Directed Interval Graph*), dans lequel chaque sommet correspond à un événement et chaque arc à une causalité potentielle. Notre approche requiert que le DIG soit acyclique (H3), ce qui est toujours vrai selon notre définition de $\to$.

Sous réserve des hypothèses (H1), (H2) et (H3), le DIG peut être converti sous forme d'un mot w, plus adapté pour traiter un flux d'événements découverts à la volée. Ce mot liste, dans l'ordre chronologique, les débuts et fins d'événements. À chaque caractère $a \in \Sigma$, nous associons un caractère $\bar{a} \in \overline{\Sigma}$, avec $\Sigma \cap \overline{\Sigma} = \emptyset$. La présence d'un caractère a correspond au début d'un événement e tel que $\lambda(e) = a$. La présence d'un caractère $\bar{a}$ correspond à la fin d'un événement e tel que $\lambda(e) = a$. La figure (a) représente un DIG et sa conversion sous forme de mot.



(a) Le DIG représenté peut être traduit par le mot $w = abc\bar{a}\bar{b}\bar{c}acd\bar{a}\bar{c}\bar{d}$



(b) Le DIG-DAG correspondant à w. Tout chemin du DIG correspond à un chemin partant de la racine q_0 du DIG-DAG.

4 Stockage des motifs (DIG-DAG)

Les chaînes de causalité correspondent aux mots formés par les étiquettes des chemins du DIG. L'objectif de cette partie est de stocker ces chaînes de causalité dans une structure proche d'un automate fini déterministe appelée DIG-DAG. Formellement, un DIG-DAG est un quadruplet $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ construit à partir d'un DIG jusqu'à une date τ tel que :

— (V, E) est un graphe acyclique orienté connexe et possédant un unique sommet source q_0 ;

— $\lambda : V \to \Sigma \cup \{\varepsilon\}$ associe à chaque sommet de V le caractère des événements associés (automate), où ε désigne l'absence d'événement. La racine q_0 est le seul sommet étiqueté par ε ;

— tout sommet du DIG-DAG possède au plus un successeur par étiquette (déterminisme). On appelle a-successeur d'un sommet son successeur étiqueté par a ;
— $\mathcal{A} \subseteq V$ est l'ensemble des sommets du DIG-DAG associés à un événement en cours. Ces sommets sont dits actifs ;
— l'ensemble des mots formés par les chemins du DIG antérieurs à la date τ coïncide avec l'ensemble des mots formés par les chemins du DIG-DAG partant du sommet q_0.

La construction d'un DIG-DAG $\mathcal{D}$ se fait en ligne, à partir d'un mot $w \in (\Sigma \cup \overline{\Sigma})^*$ découvert caractère par caractère. Intuitivement, l'ensemble $\mathcal{A}$ correspond aux sommets à partir desquels la structure va évoluer (soit grossir, soit être modifiée).

À la réception d'un caractère $a \in \Sigma$, pour chaque sommet $q \in \mathcal{A}$, on lui ajoute un a-successeur s'il n'en existe pas déjà, puis on l'ajoute à l'ensemble $\mathcal{A}$ (*extension*). Toutefois, si un tel a-successeur r existe, afin de garder la structure cohérente, il faut parfois scinder r en deux (*scission*) : l'un sera relié aux prédécesseurs inactifs de r, l'autre sera relié à ses prédécesseurs actifs et ajouté à $\mathcal{A}$. Tous deux conservent les mêmes successeurs. Afin de garder la structure compacte, toutes les feuilles actives étiquetées par a sont agrégées en une seule feuille (*fusion*).

À la réception d'un caractère $\bar{a} \in \overline{\Sigma}$, on retire de $\mathcal{A}$ l'ensemble des noeuds q tels que $\lambda(q) = a$.

Notons qu'un DIG ne définit pas un DIG-DAG de manière unique. Un extrême serait de construire un arbre, l'autre extrême serait de construire l'automate minimal [Rev92]. Notre construction du DIG-DAG offre un bon compromis entre compacité et efficacité, tout en permettant un traitement en ligne.

De plus, on peut compter le nombre d'activations de chaque sommet ou chaque arc. Ces métriques peuvent par la suite être exploitées pour extraire des motifs en fonction de leur fréquence. Par exemple, si l'on note n_a le nombre de fois que a a été observé dans w, et si l'on note $c_{(q,r)}$ le nombre de fois qu'un arc (q,r) tel que $\lambda(r) = a$ a été activé, on peut définir le degré de corrélation de l'arc (q,r) par $c_{(q,r)}/n_a$.

La figure (b) représente le DIG-DAG obtenu à partir du DIG de la figure (a). Pour plus de détails, le lecteur curieux peut se référer à [BBRS18].

5 Recherche des motifs

Un motif du DIG-DAG $\mathcal{D} = (V, E, \lambda, \mathcal{A})$ est représenté par un sous-DIG-DAG $\mathcal{D}' = (V', E', \lambda_{|V'}, \mathcal{A} \cap V')$ où (V', E') est un sous-graphe de (V, E) et où les étiquettes et sommets actifs sont ceux de $\mathcal{D}$ restreints à V'.

Parmi les motifs stockés dans $\mathcal{D}$, nous souhaitons extraire ceux conformes à une requête spécifiée par l'utilisateur. On appelle requête $\mathcal{R}(S, T, \mathcal{M}, P_v, P_e)$ l'application qui à un DIG-DAG $\mathcal{D}$ associe le plus petit sous-DIG-DAG $\mathcal{D}'$ tel que ses sommets (resp. arcs) vérifient les propriétés P_v (resp. P_e), ses sources (resp. cibles) soient inclues dans S (resp. T) et tel que tout chemin du DIG-DAG $\mathcal{D}$ reconnu par l'automate fini $\mathcal{M}$ appartienne à $\mathcal{D}'$.

L'algorithme présenté dans [SBB19] permet d'extraire d'un DIG-DAG le sous-DIG-DAG correspondant à une requête. Après avoir déterminé un ordre topologique sur les sommets de $\mathcal{D}$, la requête est traitée en deux étapes.

Tout d'abord, on parcourt les sommets dans l'ordre topologique (*aller*). Ainsi, on détermine l'ensemble des chemins partant de S, passant par des sommets (resp. arcs) vérifiant P_v (resp. P_e) qui sont des préfixes de mots reconnus par $\mathcal{M}$. Lors de ce parcours, chaque sommet de V est associé à l'ensemble des états de $\mathcal{M}$ qui lui correspondent. Ensuite, on parcourt le DIG-DAG dans l'ordre topologique inverse (*retour*) en partant des sommets de T découverts dans la phase aller et associés à un état final de $\mathcal{M}$. On raffine alors l'ensemble des chemins trouvés dans la phase aller.

Les détails, la preuve, ainsi qu'un exemple complet de cet algorithme sont donnés dans [SBB19].

6 Evaluation

Nous évaluons la performance de nos algorithmes sur trois traces réelles d'alarmes issues de réseaux GSM. Les résultats sont reportés dans la Table 1.

Un DIG-DAG construit naïvement sous forme d'arbre explose complètement en mémoire. À titre indicatif, il dépasse les 500 000 sommets après avoir traité seulement 21 (resp. 47, resp. 77) sommets de DIG1

	DIG 1	DIG 2	DIG 3
Nombre de sommets du DIG	268	266	251
Nombre d'arcs du DIG	9857	4864	937
Temps de construction du DIG-DAG (ms)	3080	298	141
Nombre de sommets du DIG-DAG	1838	304	201
Nombre d'arcs du DIG-DAG	53001	5734	1385
Nombre de sommets du DIG-DAG minimal	1810	293	156
Nombre d'arcs du DIG-DAG minimal	52703	5590	1179
Temps d'exécution de la requête (ms)	1010	407	78
Nombre d'arcs du sous-DIG-DAG	29	97	2
Nombre de sommets du sous-DIG-DAG	18	99	3

TABLE 1: Tailles et temps de construction pour trois traces GSM

(resp. DIG2, resp. DIG3). On pourrait minimiser *a posteriori* chaque DIG-DAG grâce à un algorithme de minimisation [Rev92] au détriment de la performance. Pour chaque trace, on construit donc le DIG et le DIG-DAG conformément à l'algorithme présenté en Section 4. On remarque que les DIG-DAGs ainsi construits sont déjà presque minimaux, ce qui rend la minimisation superflue.

Afin de démontrer notre capacité à trouver des motifs d'erreurs pertinents, nous avons extrait les sous-DIG-DAGs correspondant à la requête telle que les chemins de causalité dont les degrés de corrélation des arcs ($c_{(q,r)}/n_a$) sont supérieurs à 0.7 (P_e), et dont les extrémités correspondent à des alarmes sévères (S et T). On peut observer que les sous-DIG-DAGs obtenus sont significativement plus petits que le DIG et le DIG-DAG initiaux.

7 Conclusion

Cet article montre comment on peut extraire des motifs conformes à la requête d'un utilisateur dans un flux d'événements. Pour cela, nous avons proposé une nouvelle une structure de données, appelée DIG-DAG, construite en ligne et capable de mémoriser les chemins de causalité observés dans un flux de données. Il est ainsi possible d'interroger cette structure à volonté pour en extraire des motifs. Cette approche s'avère particulièrement pertinente et efficace pour l'analyse de longues traces d'événements.

Références

[AC75] Alfred V. Aho and Margaret J. Corasick. Efficient string matching : An aid to bibliographic search. *Commun. ACM*, 18(6) :333–340, June 1975.

[BBRS18] Anne Bouillard, Marc-Olivier Buob, Maxime Raynal, and Achille Salaün. Log analysis via space-time pattern matching. In *2018 14th International Conference on Network and Service Management (CNSM)*, pages 303–307. IEEE, 2018.

[LVM18] Matthieu Latapy, Tiphaine Viard, and Clémence Magnien. Stream graphs and link streams for the modeling of interactions over time. *Social Network Analysis and Mining*, 8(1) :61, 2018.

[Rev92] Dominique Revuz. Minimisation of acyclic deterministic automata in linear time. *Theoretical Computer Science*, 92(1) :181–189, 1992.

[SBB19] Achille Salaün, Anne Bouillard, and Marc-Olivier Buob. Space-time pattern extraction in alarm logs for network diagnosis. *2nd IFIP International Conference on Machine Learning for Networking*, 2019.

[Tho68] Ken Thompson. Programming techniques : Regular expression search algorithm. *Communications of the ACM*, 11(6) :419–422, 1968.

[Ukk95] Esko Ukkonen. On-line construction of suffix trees. *Algorithmica*, 14(3) :249–260, 1995.