Fouille de motifs séquentiels avec ASP

Thomas Guyet[*,**], Yves Moinard[**], René Quiniou[**], Torsten Schaub[**,***]

[*]AGROCAMPUS-OUEST/IRISA UMR 6074
65 rue de Saint Brieuc, 35042 Rennes
[**]Inria – Centre de Rennes Bretagne Atlantique
[***]Université de Potsdam, Allemagne

Résumé. Cet article présente l'utilisation de la programmation par ensembles réponses (ASP) pour répondre à une tâche de fouille de motifs séquentiels. La syntaxe de l'ASP, proche du Prolog, en fait un langage très pertinent pour représenter des connaissances de manière aisée et ses mécanismes de résolution, basés sur des solveurs efficaces, en font une solution alternative aux approches de programmation par contraintes pour la fouille déclarative de motifs. Nous proposons un premier encodage de la tâche classique d'extraction de motifs séquentiels et de ses variantes (motifs clos et maximaux). Nous comparons les performances calculatoires de ses encodages avec une approche de programmation par contraintes. Les performances obtenues sont inférieures aux approches de programmation par contraintes, mais l'encodage purement déclaratif offre plus de perspectives d'intégration de connaissances expertes.

1 Introduction

L'extraction de motifs (*pattern mining*) consiste à identifier les motifs « intéressants » dans une base de données structurée. Dans le cas de l'extraction de motifs séquentiels (Shen et al., 2014), la base de données est structurée sous la forme d'un ensemble de séquences d'itemsets (par exemple, des séquences d'achats). La tâche d'extraction de motifs consiste à identifier l'ensemble de sous-séquences qui apparaissent fréquemment dans les séquences de la base. Une fréquence suffisante, définie par un seuil f_{min} est la mesure d'intérêt classiquement utilisée pour les motifs, car elle permet de concevoir des algorithmes efficaces.

Les recherches en algorithmique ont conduit à proposer des méthodes d'extraction de motifs séquentiels de plus en plus efficaces si bien qu'elles sont en mesure d'extraire des motifs dans de très grandes bases de données et pour des seuils de fréquences très bas. Ces méthodes se heurtent maintenant à une profusion de motifs qui est devenu un problème pour les analystes qui, à la place d'être noyés sous les données, se trouvent noyés sous des motifs.

Le nouveau défi des méthodes de fouille de motifs est d'extraire moins de motifs mais plus pertinents. Il faut pour cela être en mesure d'intégrer des connaissances expertes au sein même du processus d'extraction de motifs. De telles connaissances ont été exprimées sous forme de contraintes portant, par exemple, sur la forme des motifs à extraire (*e.g.* contraintes de taille de motifs), sur la forme de leurs occurrences (*e.g.* contraintes de *max-gap* sur des séquences (Pei

et al., 2004)) ou sur l'ensemble des motifs (*e.g.* sky-patterns (Ugarte et al., 2013)). L'intégration de ces contraintes nécessite de modifier, parfois significativement, l'algorithme de fouille. L'utilisation d'une nouvelle contrainte de motifs peut ainsi conduire à un effort et un temps de développement important pour l'analyste. En pratique, l'analyste se limitera aux outils existants et sera fortement limité dans l'expression de connaissances du domaine.

Notre objectif est de développer une approche qui permette à un expert du domaine d'intégrer des connaissances riches, sans connaissance des algorithmes d'extraction. Des connaissances riches du domaine portent, par exemple, sur le fonctionnement des systèmes sous-jacents aux données. Flouvat et al. (2014) illustrent ce principe en proposant un système pour extraire des motifs appliqué à l'érosion des sols. Un modèle physique de l'érosion est introduit afin d'évaluer l'intérêt des motifs extraits. Un tel système est très riche mais requiert une conception dédiée pour assurer le fonctionnement des algorithmes. L'approche que l'on souhaite mettre en œuvre doit être suffisamment expressive pour représenter une grande variété de contraintes pour une grande variété de systèmes.

La solution envisagées dans cet article vise à utiliser une approche déclarative et plus particulièrement la programmation logique, pour intégrer des connaissances dans le processus d'extraction de motifs. Les travaux portant sur l'utilisation d'approches de programmation par contraintes ont en partie été développées dans cet objectif. La plupart de ces travaux ont été développés dans le cadre de la fouille d'itemsets (Guns et al., 2013), et des travaux récents se sont intéressés à la fouille de séquences (Negrevergne et Guns, 2015; Coquery et al., 2012; Métivier et al., 2013). Leur applicabilité réside dans l'efficacité de l'encodage du problème de fouille. Ces travaux ont montré qu'une telle approche pouvait approcher les performances d'algorithmes dédiées.

Les outils de programmation logique ont été conçus pour représenter et raisonner sur des connaissances. Ils offrent une expressivité importante et une syntaxe plus naturelle que celle de la programmation par contraintes. Le paradigme de programmation logique ASP (*Answer Set Programming* ou programmation par ensembles réponses) dispose de cette grande expressivité et s'appuie sur des solveurs de contraintes pour résoudre les programmes de manière efficace (Lifschitz, 2008). Järvisalo (2011) a montré que cette approche pouvait être viable pour la fouille de motifs.

Dans cet article, nous mettons en évidence que le problème d'extraction de motifs séquentiels et ses variantes (motifs clos, maximaux) peuvent être encodés de manière purement déclarative à l'aide d'ASP. Nous analysons les performances computationnelles de nos approches en les comparant avec celles de CPSM, une approche de programmation par contraintes.

2 ASP – Answer Set Programming

2.1 Principes de la programmation par ensembles réponses

L'ASP est une forme de programmation déclarative basée sur la sémantique des modèles stables. De manière générale, un programme se compose, d'une part, de la spécification générale du problème et, d'autre part, de la description d'une instance de ce problème, *i.e.* des données d'entrée qui serviront à sa résolution concrète.

La spécification du problème s'effectue au moyen d'un ensemble de formules logiques du premier ordre (règles similaires aux clauses d'un programme Prolog) (Lallouet et al., 2013). Un modèle ASP est une assignation d'une valeur de vérité à chacun des atomes du programme. La solution d'un programme est un ensemble de modèles minimaux, appelés *ensembles réponses*, qui ne contredisent aucune des règles du programme. Les modèles sont dits *minimaux* dans le sens où seul ce qui doit être vrai l'est effectivement.

Exemple 1. *L'exemple ci-dessous illustre les syntaxes de règles rencontrées dans un programme ASP résolvant un problème de coloration de graphe. Les lignes 1 à 6 décrivent l'instance du problème (un graphe dont les nœuds sont numérotés de 1 à 6, des arcs entre ces nœuds et 3 couleurs :* **a**, **b** *et* **c**).

Les lignes 8 à 10 décrivent le problème de la coloration d'un graphe. La ligne 9 indique que pour un nœud **X**, *un modèle doit contenir 1 et 1 seul atome de la forme* **color**(**X**,**C**) *où* **C** *est une couleur. Ces atomes décrivent la coloration. Cette règle peut être vue comme une* ***règle de choix*** *qui engendre l'espace de recherche du problème. La ligne 10 (règle sans tête) exprimer une* ***contrainte d'intégrité****. Elle impose que, dans un modèle, deux nœuds voisins ne peuvent avoir la même couleur. Le symbole " ;" dénote une conjonction en ASP.*

```
1  %description de l'instance du problème
2  node(1). node(2). node(3). node(4). node(5). node(6).
3  edge(1,2). edge(1,3). edge(1,4). edge(2,4). edge(2,5). edge(2,6).
4  edge(3,1). edge(3,4). edge(3,5). edge(4,1). edge(4,2). edge(5,3).
5  edge(5,4). edge(5,6). edge(6,2). edge(6,3). edge(6,5).
6  col(r). col(b). col(g).
7
8  %résolution du problème de coloration (3 couleurs)
9  1 { color(X, C) : col(C) } 1 :- node(X).
10 :- edge(X, Y); color(X, C); color(Y, C).
```

L'ensemble **color**(2,**r**) **color**(1,**b**) **color**(3,**r**) **color**(4,**g**) **color**(5,**b**) **color**(6,**g**) *est une solution du programme ci-dessous.*

Dans le principe, l'ensemble des solutions possibles est exprimé par les règles de choix du programme. La résolution consiste donc à explorer l'ensemble de ces modèles et d'évaluer les contraintes sur celui-ci. Les modèles qui respectent l'ensemble des contraintes sont donnés comme ensembles réponses. La résolution du problème est assurée efficacement grâce à des solveurs spécifiques (Lifschitz, 2008). Ils présentent des analogies fortes avec les mécanismes de la programmation par contraintes.

2.2 La suite de logiciels `Potassco`

Les moteurs actuels de résolution des programmes ASP s'appuient sur des solveurs de contraintes efficaces. En pratique, nous utilisons l'outil `clingo` [1] (Gebser et al., 2011) dont les performances sont remarquables et dont la syntaxe respecte le standard d'ASP [2].

L'outil `clingo` décompose la résolution d'un programme ASP en deux parties distinctes :

— l'instanciation (ou *grounding*) qui consiste à propositionnaliser le programme ASP écrit dans une syntaxe du premier ordre. Le programme est transformé en clauses proposi-

1. Potassco : `http://potassco.sourceforge.net/`
2. Standard ASP : `https://www.mat.unical.it/aspcomp2013/ASPStandardization`

tionnelles et en clauses d'optimisation exprimant des contraintes sur les modèles solutions.

— la résolution qui consiste à trouver de une à toutes les solutions du programme instancié. `clasp`, le solveur de la suite `Potassco`, est un solveur SAT ou MaxSAT. La résolution utilise une approche dirigée par les conflits. L'algorithme s'appuie sur l'apprentissage de *nogood* et utilise, par défaut, une heuristique de type *vsids* qui peut être adaptée aux besoins du problème.

La résolution peut être contrôlée à l'aide de langages procéduraux, en particulier `Python`. Cette fonctionnalité est utile pour automatiser des traitements et pour récupérer des statistiques sur la résolution des problèmes.

L'ensemble des outils disponibles ainsi que l'efficacité reconnue du solveur nous ont conduit à préférer nous appuyer sur cette suite logicielle plutôt que sur l'un des outils alternatifs tels que DLV (Leone et al., 2006) ou ASPeRiX (Lefèvre et Nicolas, 2009).

3 Extraction de motifs séquentiels

Dans cette section, on rappelle les définitions classiques de l'extraction de motifs séquentiels à partir d'une base de séquences. On note $[n] = \{1, \ldots, n\}$, l'ensemble des n premiers entiers.

Soit $(\mathcal{E}, =, <)$ l'ensemble de tous les types d'items muni d'un ordre total (*e.g.* lexicographique). Un *itemset* $A = (a_1, a_2, ..., a_n),\ \forall i \in [n],\ a_i \in \mathcal{E}$ est un ensemble ordonné, *i.e.* $\forall i \in [n-1],\ a_i < a_{i+1}$, d'items distincts, *i.e.* $\forall i, j \in [n],\ i \neq j \Rightarrow a_i \neq a_j$.

Un itemset $\beta = (b_i)_{i \in [m]}$ est un sous-itemset de $\alpha = (a_i)_{i \in [n]}$, noté $\beta \sqsubseteq \alpha$, ssi il existe une suite de m entiers $1 \leq i_1 < i_2 < ... < i_m \leq n$ telle que $\forall k \in [m], b_k = a_{i_k}$.

Une *séquence* S est une suite ordonnée d'itemsets $S = \langle s_1, s_2, ..., s_n \rangle$. La longueur d'une séquence S, notée $|S|$ est le nombre d'itemsets qui composent la séquence.

$T = \langle t_1, t_2, ..., t_m \rangle$ est une *sous-séquence* de $S = \langle s_1, s_2, ..., s_n \rangle$, noté $T \preceq S$, ssi il existe une suite d'entiers $1 \leq i_1 < i_2 < ... < i_m \leq n$ telle que $\forall k \in [m],\ t_k \sqsubseteq s_{i_k}$.

$T = \langle t_1, t_2, ..., t_m \rangle$ est une *préfixe* de $S = \langle s_1, s_2, ..., s_n \rangle$, noté $T \preceq_b S$, ssi $t_m \sqsubseteq s_m$ et $\forall i \in [k-1],\ t_i = s_i$. On a alors $T \preceq_b S \Rightarrow T \preceq S$.

Exemple 2. *Soient $\mathcal{E} = \{a, b, c\}$ muni de l'ordre lexicographique ($a < b$, $b < c$) et la séquence $S = \langle a(bc)(abc)cb \rangle$. Pour simplifier l'écriture, les parenthèses sont omises autour des itemsets ne comprenant qu'un seul item. S est de longueur 5. Les séquences $\langle abb \rangle$, $\langle (bc)(ac) \rangle$ ou $\langle a(bc)(abc)cb \rangle$ sont des sous-séquences de S. $\langle a(bc)a \rangle$ et $\langle a(bc) \rangle$ sont des préfixes de S.*

Propriété 1. *La relation $\preceq$ est un ordre partiel sur l'ensemble des séquences.*

Soit $\mathcal{D} = \{S_1, S_2, \ldots, S_N\}$ une **base de séquences**. Le **support** d'une séquence S, noté $supp(S)$, est le nombre de séquences de $\mathcal{D}$ qui supportent S :

$$supp(S) = |\{S_i \in \mathcal{D} | S \preceq S_i\}|$$

Propriété 2. *$supp$ est une mesure anti-monotone sur l'ensemble des séquences muni de la relation $\preceq$ ou $\preceq_b$.*

Cette propriété implique que pour toute paire de séquences P, Q, nous avons : $P \preceq Q \Rightarrow supp(P) \geq supp(Q)$, et $P \preceq_b Q \Rightarrow supp(P) \geq supp(Q)$.

Soit σ le support minimum défini par l'analyste. Pour une séquence S, si $supp(S) \geq \sigma$, on dit que S est une **séquence fréquente** ou un **motif séquentiel fréquent**. L'**extraction de motifs séquentiels** consiste à extraire tous les motifs séquentiels fréquents dans la base de séquences $\mathcal{D}$.

La propriété d'anti-monotonie est une propriété usuellement utilisée par les algorithmes d'extraction de motifs pour élaguer efficacement l'espace de recherche. En effet, si un motif P n'est pas fréquent, il est inutile d'explorer les motifs Q tels que $P \preceq Q$.

La définition de la tâche d'extraction de tous les motifs fréquents s'accompagne généralement de deux tâches d'extraction : les motifs maximaux et les motifs clos.

Un motif fréquent est **maximal** (resp. **backward-maximal**) vis-à-vis de la relation $\preceq$ (resp. $\preceq_b$) ssi il n'y a pas de motif fréquent S' tel que $S \preceq S'$ (resp. $S \preceq_b S'$).

Un motif fréquent S est **clos** (resp. **backward-clos**) vis-à-vis d'une relation $\preceq$ (resp. $\preceq_b$) ssi il n'y a aucun autre motif S' tel que $S \preceq S'$ (resp. $S \preceq_b S'$) et $supp(S) = supp(S')$. Extraire l'ensemble de motifs clos réduit significativement le nombre de motifs sans perdre d'informations sur l'ensemble de tous les motifs fréquents.

4 Programme ASP pour l'extraction de motifs séquentiels

De manière similaire à Järvisalo (2011), les programmes ASP ci-dessous se basent sur l'idée qu'un modèle est un motif, et qu'un ensemble-réponse est un motif fréquent. Pour être complet et correct, le programme ASP doit donc explorer tous les motifs possibles et les contraintes exprimées doivent être vérifiées si et uniquement si le motif est fréquent.

Dans la suite, on présente et on évalue des programmes pour une tâche d'extraction de motifs séquentiels qui sont des séquences d'items (et non d'itemsets) dans une base contenant toujours des séquences d'itemsets. Cette limitation permet une présentation plus pédagogique des programmes. L'extension au cas général des itemsets est aisée.

4.1 Représentation de la base de séquences

La base de séquences $\mathcal{D}$ est représentée au moyen d'atomes du prédicat **seq**/3. Par exemple, l'atome **seq**(1,3,5) déclare que l'item 5 est présent dans la séquence 1 à la position 3. De manière similaire, le motif associé au modèle est décrit à l'aide d'atomes du prédicat **pat**/2 : **pat**(2,4) indique que le motif contient l'item 4 à la position 2.

On définit également deux constantes au programme :

— **#const th**=23. : seuil de fréquence **th** (en nombre d'occurrences),

— **#const maxlen**=10. : longueur maximale des motifs.

4.2 Encodage de l'extraction de motifs séquentiels

Le programme 1 présente un programme pour l'extraction de motifs séquentiels. La première ligne effectue une projection de la base de séquences pour extraire l'ensemble des items qu'elle contient.

Les lignes 3 à 8 engendrent l'espace de recherche, c'est-à-dire qu'elles permettent de décrire tous les motifs séquentiels possibles (de longueur inférieure à **maxlen**). Dans cette génération, l'ensemble des atomes du prédicat **patpos**/1 représentent une liste d'entiers consécutifs dont la longueur est inférieure à **maxlen**. Toute longueur inférieure à **maxlen** est possible (exprimée par l'absence de nombres de part et d'autre de l'accolade). Le prédicat **patlen**/1 détermine cette longueur (ligne 6). La ligne 8 engendre les motifs proprement dits (génération des atomes de **pat**/2). Pour chaque position **X**, un unique atome **pat**(**X**,**I**) est choisi parmi l'ensemble des items possibles. Ces lignes garantissent la complétude de l'exploration des motifs de taille inférieure ou égale à **maxlen**.

```
1 item(I) :- seq(_, _, I). %génération de la liste des items dans la base
2
3 %génération des motifs séquentiels
4 patpos(1).
5 0 { patpos(X+1) } 1 :- patpos(X); X<maxlen.
6 patlen(L) :- patpos(L); not patpos(L+1).
7
8 1 { pat(X, I): item(I) } 1 :- patpos(X).
9
10 %occurrences du motif
11 occ(T, 1, P) :- seq(T, P, I):pat(1, I); seq(T, P, _).
12 occ(T, Pos+1, P) :- occ(T, Pos, Q); Q<P; seq(T, P, _); seq(T, P, I):pat(Pos+1, I).
13
14 %contrainte de fréquence
15 support(T) :- occ(T, L, _); patlen(L).
16 :- #count{ T: support(T) } < th.
```

Listing 1 – *Programme pour l'extraction de motifs séquentiels (occurrences skip-gaps)*

La seconde partie du programme, lignes 10 à 12, recherche les occurrences du motif séquentiel dans une séquence **T** de la base. Le prédicat **occ**(**T**,**N**,**P**) sert à représenter une occurrence où **T** est l'identifiant de la séquence, **N** est une position dans la séquence et **P** est une position dans le motif. La liste d'atomes du prédicat **occ**/3 décrit les correspondances entre items du motif et itemsets de la séquence. La construction de telles listes s'écrit récursivement pour être efficace :

— initialement (ligne 11), on cherche une correspondance entre le premier item du motif et une position dans la séquence.

— lorsqu'on a identifié une correspondance entre l'item du motif à la position **N** avec un itemset à la position **P** dans la séquence **T**, on cherche toutes les possibilités de trouver l'item de la position **N**+1 dans le motif avec un itemset à une position postérieure (**Q**>**P**).

Cette représentation des occurrences, nommée ici *skip-gap*, identifie toutes les occurrences possibles d'un motif dans une unique séquence (et pas uniquement la première).

On peut noter qu'une occurrence de la séquence a été trouvée dans la séquence **T** si et seulement si l'atome **occ**(**T**,**L**,_) a été généré dans le modèle où **L** est la longueur du motif. Cette propriété assure la correction de l'approche.

Finalement, les lignes 14 à 16, ajoutent la contrainte de fréquence sur les motifs. L'atome **support**(**T**) est vérifié si une occurrence du motif est bien trouvée dans la séquence **T** et, ligne 16, on ajoute une contrainte spécifiant que le modèle n'est pas conservé si le nombre d'atomes du prédicat **support**/1 est inférieur à **th** (constante du programme, *cf.* section 4.1).

4.3 Une recherche alternative d'occurrences

Contrairement aux approches CP qui utilisent des méthodes dédiées pour vérifier qu'une séquence supporte un motif, l'approche proposée est purement déclarative et fait réaliser cette tâche par le solveur ASP. Nous proposons un encodage alternatif en vue d'améliorer les performances globales de nos programmes.

```
1 occ(T, 1, P) :- pat(1, I); seq(T, P, I).
2 occ(T, L, P+1) :- occ(T, L, P); seq(T, P, _).
3 occ(T, L+1, P+1) :- occ(T, L, P); seq(T, P+1, I); pat(L, I).
4
5 support(T) :- occ(T, L, LS); patlen(L); seqlen(T, LS).
```

Listing 2 – *Recherche d'occurrences « fill gaps »*

Dans le programme 2, on propose une méthode alternative pour réaliser la recherche des occurrences. Les atomes **occ**/3 sont toujours utilisés pour créer une correspondance entre les items du motif et les itemsets de la séquence. Néanmoins, leur sémantique est différente : **occ** (**T**,**L**,**P**) indique que lorsqu'on a « lu » les **P** premiers itemsets de la séquence **T**, on a reconnu **L** items du motif séquentiel. Si, à la fin de la séquence, position **LS**, on a lu **L** items, *i.e.* la longueur du motif, alors c'est que la séquence supporte le motif (ligne 5). Cet encodage évite l'énumération de toutes les occurrences possibles.

4.4 Extraire des motifs maximaux ou clos

Dans cette tâche, la difficulté réside principalement dans le fait que le processus de résolution utilisé par `clingo` traite les modèles de manière indépendante. Les mécanismes classiques de résolution ne permettent pas comparer deux motifs [3]. L'encodage des motifs clos et maximaux doit donc s'appuyer sur des propriétés propres au motif.

4.4.1 Motifs maximaux

La définition de la section 3 implique de comparer deux motifs. On reformule ici ces définitions pour exprimer la notion de motif maximal dans nos programmes ASP (un modèle par motif et un ensemble-réponse est un motif fréquent maximal).

Soit un motif $T = \langle t_1, \ldots, t_m \rangle$ et une séquence $S = \langle s_1, \ldots, s_n \rangle$ telle que S supporte T, $T \preceq S$. Donc il existe un ensemble de suites d'entiers, $\sigma_p = 1 \leq i_1^p \leq i_2^p \leq ... \leq i_m^p \leq n$ telles que $\forall k \in [m], t_k \sqsubseteq s_{i_k^p}$. Le suffixe de la séquence S pour T est la séquence $St = \langle s_{q+1}, \ldots, s_n \rangle$ telle que $q = argmin_p(i_m^p)$. Le suffixe est la partie de la séquence qui se situe après la position du dernier item de la première occurrence du motif. Un motif séquentiel fréquent $T = \langle t_1, \ldots, t_n \rangle$ est *backward*-maximal si pour tout item $a \in \mathcal{E}$, $T' = \langle t_1, \ldots, t_n, a \rangle$ n'est pas fréquent. De plus, par anti-monotonie, on sait que si T' est supporté par une séquence alors T doit l'être aussi. On en déduit que pour avoir T *backward*-maximal, il faut qu'il y ait moins f_{min} séquences supportées dont le suffixe contienne un même item a et, d'autre part, que ce a ne peut être que dans l'intersection des suffixes des séquences supportées par le motif.

Pour l'encodage ASP, on introduit un prédicat **suffix**(**T**,**I**) qui indique que l'item **I** se trouve dans le suffixe d'une séquence supportée. La première ligne du listing ci-dessous construit les

3. `clingo` permet des résolutions incrémentales pouvant tenir compte des contraintes monotones entre modèles. ASPRIN (Brewka et al., 2015), une sur-couche de `clingo`, permet d'écrire des préférences entre modèles.

atomes de ce prédicat dans le cas de l'encodage des occurrences du programme 1. La seconde ligne définit la contrainte de maximalité des motifs.

```
suffix(T, I) :- occ(T, L, Q); patlen(L); seq(T, P, I); P>Q.
:- #count{ T: support(T), suffix(T, I)} >= th; item(I).
```

Dans le cas de la construction des occurrences avec le programme 2, les atomes du prédicat **suffix**/2 sont construites par la règle ci-dessous. Dans cette règle, **P** désigne une position pour laquelle un dernier élément d'une occurrence a déjà été trouvé avant, en incluant la position de ce dernier élément. Un item est dans le suffixe lorsqu'il se trouve à une position **P**+1.

```
suffix(T, I) :- occ(T, L, P); patlen(L); seq(T, P+1, I).
```

4.4.2 Motifs clos

La définition d'un motif clos fait référence également à d'autres motifs. De même que pour la contrainte de maximalité, une nouvelle définition doit être utilisée pour exprimer cette contrainte. Un motif n'est pas clos si un motif plus grand (au sens de $\preceq_b$ ici) a exactement le même support. Si c'est le cas, par la contrainte d'anti-monotonie, on doit forcément avoir un item qui apparaît dans les suffixes de toutes les séquences supportées.

La règle ci-dessous ajoutée aux règles de production des atomes **suffix** (voir section précédente) extrait uniquement les motifs clos. Elle reprend la propriété précédente en éliminant les modèles pour lesquels un nouvel item se trouve dans le suffixe de toutes les séquences couvertes. Cette règle, bien que simple d'écriture, utilise une expression usuelle d'ASP pour vérifier une propriété pour un ensemble d'atomes. Ici, pour un **I** donné, le programme contraint à ne pas avoir les atomes **suffix**(**T**,**I**) vrais pour toutes les séquences **T** supportées par le motif.

```
:- suffix(T,I): support(T); item(I).
```

5 Expérimentations et résultats

L'objectif de nos expérimentations est de comparer les performances et les limites, en temps de calcul et en mémoire, de nos différents programmes ASP et de les comparer à l'approche de programmation par contraintes CPSM (Negrevergne et Guns, 2015). La tâche résolue par CPSM – similaire à la nôtre – et son approche semi-déclarative – la recherche d'occurrences est réalisée par un propagateur dédié – en font un point de référence naturel. CPSM ayant été comparé aux algorithmes dédiés, le lecteur pourra comparer les performances des approches dédiées, CP et ASP en se reportant à ce travail.

5.1 Expérimentations sur données réelles

Les jeux de données utilisés dans les expérimentations sont ceux utilisés par CPSM [4] : Unix, JMLR, iPRG et FIFA (voir les caractéristiques des jeux de données dans le tableau 1. Pour chacun de ces jeux de données, des résolutions ont été lancées avec différents seuils (2,5%, 5%, 10%, 15%, 20%, 30% et 40%) et avec les deux alternatives de recherche d'occurrences (`sg` pour *skip-gaps* et `fg` pour *fill-gaps*). Cinq types de résolution ont été testés :

4. `https://dtai.cs.kuleuven.be/CP4IM/cpsm`

TAB. 1 – *Caractéristiques des jeux de données : nombre de symboles, nombre de séquences et d'items, longueurs maximale et moyenne des séquences et densité de la base.*

Dataset	Σ	$\|D\|$	$\|\|D\|\|$	$max\|T\|$	$avg\|T\|$	$density$
Unix user	265	484	10935	1256	22.59	0.085
JMLR	3847	788	75646	231	96.00	0.025
iPRG	21	7573	98163	13	12.96	0.617
FIFA	20450	2990	741092	100	36.239	0.012

— extraction de motifs séquentiels,
— extraction de motifs séquentiels maximaux,
— extraction de motifs séquentiels clos,
— extraction de motifs séquentiels maximaux avec utilisation de l'heuristique `subset minimal`,
— extraction de motifs séquentiels clos avec l'heuristique `subset minimal`.

L'heuristique `subset minimal` indique au solveur que si un motif P a été trouvé, alors il est inutile d'explorer les motifs plus généraux que P (*i.e.* $\{Q|P \preceq Q\}$). L'utilisation de cette heuristique n'a pas de conséquence sur la complétude de l'extraction des motifs maximaux, en revanche son utilisation sur l'extraction de tous les motifs fréquents ou sur l'extraction des clos conduit à n'en extraire qu'un sous-ensemble.

Chaque exécution a été lancée sur un serveur disposant d'une quantité de mémoire suffisante (environ 20Go requis pour le jeu de données `FIFA`) avec 4 *threads* sur un unique processeur. Un *timeout* a été fixé à $3600s$. Si ce *timeout* est atteint, il est toujours possible de comparer les performances en temps, en comparant le nombre de motifs qui sont extraits dans ce laps de temps. Finalement, on évalue l'usage mémoire de nos programmes en évaluant la quantité de règles et d'atomes qui ont été nécessaires dans l'instanciation du programme.

5.2 Résultats

L'ensemble des résultats est illustré dans la Figure 1. On constate d'abord que l'approche ASP est de 1 à 2 ordre de grandeurs plus lente que l'approche de CPSM. On constate de plus que la différence est d'autant plus importante que les séquences sont longues. L'approche purement déclarative que nous utilisons n'est pas compétitive avec l'utilisation de propagateurs pour l'identification des occurrences d'un motif dans une séquence. Cette différence sera d'autant plus importante que ce test sera difficile, c'est à dire dans le cas des séquences longues (en particulier pour JMLR).

Au sein des résolutions ASP, on constate que l'utilisation de la recherche *fill-gaps* donne de meilleurs résultats. Cela s'explique par la limitation de l'évaluation multiple des occurrences d'une même séquence et aussi par une réduction significative de la taille du problème encodé. En particulier, pour un jeu de données dense comme IPRG, on constate que l'utilisation de cette stratégie est très efficace.

La différence entre les temps de calcul des différentes tâches d'extraction (tous, maximaux et clos) n'est pas très nette. On peut globalement indiquer que l'extraction des motifs maximaux est plus efficace que celle des clos et encore plus que celle de tous les items. Une partie de l'explication de ces résultats réside dans le nombre d'ensemble réponse : il y a moins de motifs maximaux que de motifs clos. En revanche, l'efficacité se paye par un besoin de mémoire

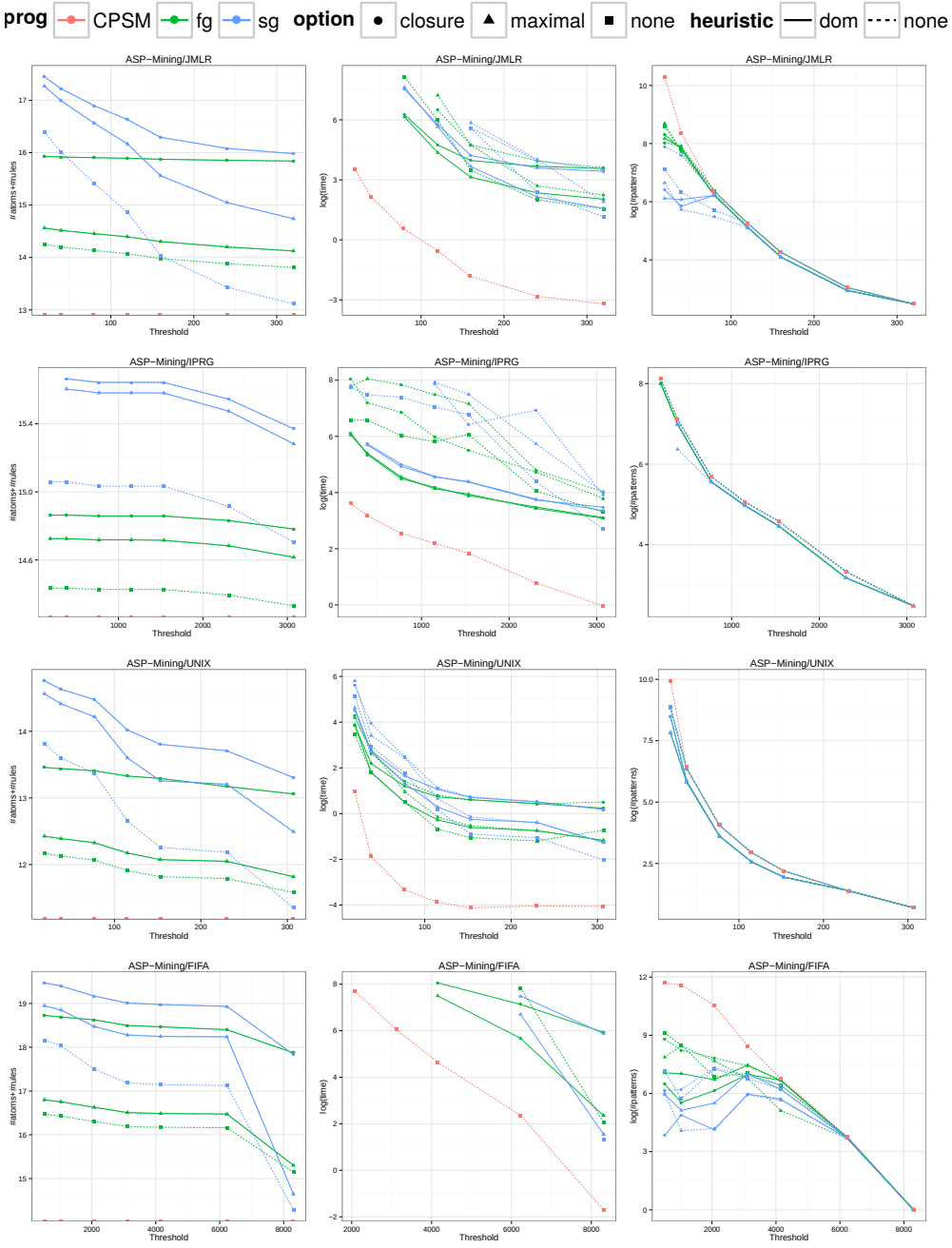

FIG. 1 – *De gauche à droite : taille du problème en mémoire, temps de calcul et nombre de motifs extraits. Les temps de calcul ne sont donnés que pour les résolutions complètes avant le timeout. En revanche, le nombre de motifs extrait est donné pour tous les résolutions. De haut à en bas : JMLR, IPRG, UNIX et FIFA. Pour chaque graphique, les courbes comparent les résultats obtenus pour les différents programmes (CPSM,* `skip-gap` *et* `fill-gap`*) et approches (voir légende).*

supplémentaire dû à une augmentation de la taille du problème encodé. On peut également noter, que les maximaux ont un coût mémoire moindre que ces clos.

Finalement, on peut noter que l'utilisation de l'heuristique *subset minimal* se montre très efficace. Cette heuristique est d'autant plus intéressante que la résolution reste complète pour l'extraction des motifs maximaux.

Les constatations similaires peuvent être faites à partir des courbes représentants le nombre de motifs. CPSM extrait quasiment tous les motifs séquentiels fréquents (seul FIFA avec 2.5% et 5% n'est pas finalisé avant le *timeout*). Si, pour des seuils élevés, les courbes des différentes approches se suivent (indiquant que les ensembles de motifs sont les mêmes), on constate que les courbes s'approchent de CPSM pour les seuils bas. On peut finalement constater que l'usage mémoire est relativement indépendant du seuil dans le cas d'utilisation de la recherche *fill-gaps*. Il n'y aura pas plus de problème d'instanciation du programme, étape qui charge en mémoire le programme pour le solveur, pour les seuils bas que pour les seuils élevés.

6 Conclusions et perspectives

Cet article présente une solution pour effectuer la fouille de motifs séquentiels en ASP. Les expériences montrent tout d'abord que l'approche de résolution ASP permet de s'attaquer à des données réelles. L'efficacité réduite de notre approche provient principalement d'un encodage purement déclaratif de la recherche des occurrences. Les programmes présentés constituent des premières approches qu'il est possible d'améliorer. Les perspectives d'amélioration visent à encoder des propriétés additionnelles de la tâche de fouille pour faciliter sa résolution.

L'acception générale de la tâche de fouille de séquences à laquelle nous nous sommes confrontés est réputée calculatoire. Des améliorations pratiques des performances sont attendues avec une tâche plus simple. En particulier, l'extraction de motifs séquentiels avec contraintes temporelles (Pei et al., 2004) tels que les *max-gap* diminue de manière importante les calculs nécessaires. L'ajout de telles contraintes aux programmes présentés est aisé.

L'étape suivante de notre travail sera de montrer que ces programmes peuvent intégrer des connaissances expertes riches, facilement exprimables, pour aller vers l'extraction de motifs pertinents.

Références

Brewka, G., J. P. Delgrande, J. Romero, et T. Schaub (2015). asprin : Customizing answer set preferences without a headache. In *proceedings of National Conference on Artificial Intelligence (AAAI)*, pp. 1467–1474.

Coquery, E., S. Jabbour, L. Saïs, et Y. Salhi (2012). A SAT-Based approach for discovering frequent, closed and maximal patterns in a sequence. In *proceedings of European Conference on Artificial Intelligence (ECAI)*, pp. 258–263.

Flouvat, F., J. Sanhes, C. Pasquier, N. Selmaoui-Folcher, et J.-F. Boulicaut (2014). Improving pattern discovery relevancy by deriving constraints from expert models. In *Proceedings of the European Conference on Artificial Intelligence (ECAI)*, pp. 327–332.

Gebser, M., R. Kaminski, B. Kaufmann, M. Ostrowski, T. Schaub, et M. Schneider (2011). Potassco : The Potsdam answer set solving collection. *AI Communications 24*(2), 107–124.

Guns, T., A. Dries, G. Tack, S. Nijssen, et L. De Raedt (2013). MiningZinc : A modeling language for constraint-based mining. In *Proceedings of the international joint conference on Artificial Intelligence (IJCAI)*, pp. 1365–1372.

Järvisalo, M. (2011). Itemset mining as a challenge application for answer set enumeration. In *Proceedings of the conference on Logic Programming and Nonmonotonic Reasoning*, pp. 304–310.

Lallouet, A., Y. Moinard, P. Nicolas, et I. Stéphan (2013). Programmation logique. In P. Marquis, O. Papini, et H. Prade (Eds.), *Panorama de l'intelligence artificielle Ses bases méthodologiques, ses développements*, Volume 2. Cépaduès.

Lefèvre, C. et P. Nicolas (2009). The first version of a new ASP solver : ASPeRiX. In *Proceedongs of the conference on Logic Programming and Nonmonotonic Reasoning*, pp. 522–527.

Leone, N., G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, et F. Scarcello (2006). The DLV system for knowledge representation and reasoning. *ACM Trans. Comput. Logic 7*(3), 499–562.

Lifschitz, V. (2008). What is answer set programming ? In *Proceedings of the Conference on Artificial Intelligence (AAAI)*, pp. 1594–1597.

Métivier, J.-P., S. Loudni, et T. Charnois (2013). A constraint programming approach for mining sequential patterns in a sequence database. *CoRR abs/1311.6907*.

Negrevergne, B. et T. Guns (2015). Constraint-based sequence mining using constraint programming. In *Proceedings of International Conference on Integration of AI and OR Techniques in Constraint Programming, CPAIOR*, pp. 288–305.

Pei, J., J. Han, B. Mortazavi-Asl, J. Wang, H. Pinto, Q. Chen, U. Dayal, et M.-C. Hsu (2004). Mining sequential patterns by pattern-growth : the PrefixSpan approach. *IEEE Transactions on Knowledge and Data Engineering 16*(11), 1424–1440.

Shen, W., J. Wang, et J. Han (2014). Sequential pattern mining. In C. C. Aggarwal et J. Han (Eds.), *Frequent Pattern Mining*, pp. 261–282. Springer International Publishing.

Ugarte, W., P. Boizumault, S. Loudni, B. Crémilleux, et A. Lepailleur (2013). Découverte des soft-skypatterns avec une approche PPC. In *Actes des Journées Francophones Extraction et Gestion de Connaissances (EGC)*, pp. 217–228.

Summary

This article presents the use of answers set programming (ASP) to mine sequential patterns. ASP is an alternative to constraint programming approaches to mine patterns in a declarative manner. Indeed, the syntax of ASP, close to Prolog, is very relevant for representing knowledge and its solving process is efficient. We propose a first encoding of sequential (closed and maximal) patterns mining. We compare computational performances of these encodings with respect to a constraint programming approach (CPSM). The performances obtained are lower than the former but our purely declarative encoding offers more perpectives for expert knowledge integration.