Mining contrast sequential patterns with ASP

Francesca Alessandra Lisi[1][0000−0001−5414−5844] and Gioacchino Sterlicchio[2][0000−0002−2936−0777]

[1] DIB and CILA, University of Bari Aldo Moro, Italy
[2] DMMM, Polytechnic University of Bari, Italy
FrancescaAlessandra.Lisi@uniba.it, g.sterlicchio@phd.poliba.it

Abstract. In this paper we address an extension of the sequential pattern mining problem which aims at detecting the significant differences between frequent sequences with respect to given classes. The resulting problem is known as contrast sequential pattern mining, since it merges the two notions of sequential pattern and contrast pattern. For this problem we present a declarative approach based on Answer Set Programming (ASP). The efficiency and the scalability of the ASP encoding are evaluated on two publicly available datasets, iPRG and UNIX User, by varying parameters, also in comparison with a hybrid ASP-based approach.

Keywords: Declarative Pattern Mining · Contrast Sequential Pattern Mining · Answer Set Programming.

1 Introduction

In recent times there is an increasing availability of data that contain sequences of events, items, or tokens organized according to an ordered metric space. The requirement to detect and analyze frequent subsequences has therefore become a common problem. Sequential Pattern Mining (SPM) arose as a subfield of pattern mining just to address this need (see, e.g., [18] for a survey). More precisely, the typical task in SPM consists in finding frequent and non-empty temporal sequences, called *sequential patterns*, from a dataset of sequences. Another interesting class of pattern mining problems goes under the name of Contrast Pattern Mining [4]. Here, the typical task is about detecting statistically significant differences/similarities, called *contrast patterns*, between two or more disjoint datasets (or portions of the same dataset). Sequential and contrast pattern mining are known to have a higher complexity than, e.g., itemsets mining. However, they have broad applications, e.g., the analysis of patient care pathways, education traces, digital logs (web access for client profiling, intrusion detection from network logs), customer purchases (rules for purchases recommendations), text and bioinformatic sequences. In this paper we consider to merge the concepts of sequential pattern and contrast pattern in order to find significant differences between frequent sequences with respect to given classes. This gives rise to the concept of *contrast sequential pattern*. The resulting problem of Contrast Sequential Pattern Mining (CSPM) is not new. However, it has been little addressed so far (see [3] for a recent survey).

In this paper we address the CSPM task by means of a declarative approach. In particular, we resort to Answer Set Programming (ASP) [14,2], a well-established computational logic paradigm for declarative problem solving. Declarative approaches are generally desirable in application domains where the requirements of transparency, verifiability and explainability of the AI techniques employed are of paramount importance, such as bio-informatics. For this reason a novel stream of research called Declarative Pattern Mining (DPM) has been proposed which can be more useful and appropriate in such contexts. To the best of our knowledge, no DPM approach exists that supports intensive knowledge-based contrast sequence mining. We have developed a concise and versatile ASP encoding for CSPM and for managing complex preferences on patterns. In particular, we have extended previous work on ASP-based SPM with the necessary code for checking which frequent sequential patterns out of the discovered ones highlight significant differences with respect to two classes. We have evaluated the encoding on two real-world public datasets of sequences (iPRG and UNIX User) which are enriched with information about the class of reference for each sequence. For a comparative evaluation, we have chosen a hybrid ASP-based approach that combines a first step with a traditional algorithm for SPM and a second step with ASP.

The paper is organized as follows. In Section 2 we overview the current research in CSPM and DPM. In Section 3 we provide the necessary background on ASP and CSPM. In Section 4 we describe our ASP enconding for the CSPM task and in Section 5 we report the experimental results obtained on the chosen datasets. Section 6 concludes the paper with final remarks.

2 Related works

Sequential and Contrast Pattern Mining are challenging tasks in data mining and play an important role in many applications. Notably, *PrefixSpan* is an optimized algorithm for mining sequences [11]. The notion of contrast is deeply discussed by Dong in [4]. *Chen et al.* [3] provide an up-to-date comprehensive and structured overview of the research in Contrast Pattern Mining which includes also the case of contrast sequential patterns. In particular, *Zheng et al.* [23] present a CSPM method for taxpayer behaviour analysis, and *Wu et al.* [22] propose a top-k self-adaptive CSPM solution.

DPM covers many pattern mining tasks such as sequence mining [19,5] and frequent itemset mining [12,7]. In [19], the authors organize the constraints on sequential patterns in three categories: 1) constraints on patterns, 2) constraints on patterns embeddings, 3) constraints on pattern sets. These constraints are provided by the user and capture his background knowledge. Then, they introduce two formulations based on Constraint Programming (CP). *Jabbour et al.* [12] propose SAT based encodings of itemset mining problems to overcome space complexity issue behind the competitiveness of new declarative and flexible models. In [7], *MiningZinc* is presented as a declarative framework for constraint-based data mining.

Besides SAT and CP, ASP is also widely used in DPM. The first proposal is described by *Guyet et al.* [9]. The authors explore a first attempt to solve the SPM problem with ASP and compare their method with a dedicated algorithm. Next, in [5] *Gebser et al.* use ASP for extracting condensed representation of sequential pattern. They focus on closed, maximal and skyline patterns. *Samet et al.* in [21] show a method for mining meaningful rare sequential patterns with ASP, whereas in [8] *Guyet et al.* propose to apply an ASP-based DPM approach to investigate the possible association between hospitalization for seizure and antiepileptic drug switch from a french medico-administrative database. *Guyet et al.* [10] present the use of ASP to mine sequential patterns within two representations of embeddings (fill-gaps vs skip-gaps) and various kinds of patterns: frequent, constrained and condensed. *Besnard and Guyet* [1] address the task of mining negative sequential patterns in ASP. A negative sequential pattern is specified by means of a sequence consisting of events to occur and of other events, called negative events, to be absent. In [15,16] Guyet's ASP encodings for SPM are adapted in order to address the requirements of an application in the digital forensics domain: The analysis of anonymised mobile phone recordings. Motivated by the same application, *Lisi and Sterlicchio* present an ASP-based approach to contrast pattern mining in [17].

Whereas all the works mentioned so far are pure ASP-based DPM solutions, particularly interesting is the hybrid ASP-based approach proposed by *Paramonov et al.* [20] which combines dedicated algorithms for pattern mining and ASP. The authors show that such two-step approach outperforms one-shot ones.

3 Preliminaries

3.1 Answer Set Programming

ASP is a declarative and expressive programming language to resolve difficult research problems (e.g. security analysis, planning, configuration, semantic web, etc.) introduced at the end of the 90s [2]. Every ASP programs is made up of atoms, literals and logic rules. Atoms can be true or false and a literal is an atom a or its negation $not\, a$. An ASP general rule has the following form:

$$a_1 \vee \ldots \vee a_n \leftarrow b_1, \ldots, b_k, not\, b_{k+1}, \ldots, not\, b_m \tag{1}$$

where all a_i and b_j are atoms. The previous rule says that if $b_1, \ldots, b_k$ are true and there is not reason for believing that $b_{k+1}, \ldots, b_m$ are true then at least one of the $a_1, \ldots, a_n$ is believed to be true. The *not* statement is not the standard logical negation but it is used to derive $not\, p$ (i.e. p is assumed not to hold) from failure to derive p. The left side of the $\leftarrow$ is called *head* while the right side *body*. Rules of the form "$a \leftarrow$" are called *facts* and they have no body. The head is unconditionally true and the arrow is usually omitted. Rules of the form "$\leftarrow b_1, \ldots, b_k$" are called *constraints*. Adding a constraint in a program deletes answer sets that satisfy the constraint body. There are different ASP systems, the most important are Clingo [6] and DLV [13].

3.2 Contrast Sequential Pattern ¡mining

Sequential Pattern Mining [18] aims at identifying frequent subsequences within a sequences database $\mathcal{D}$. In the following, we briefly formalize the SPM problem. Throughout this article, $[n] = \{1, \dots, n\}$ denotes the set of the first n positive integers. Let Σ be the alphabet, i.e, the set of items. An *itemset* $A = \{a_1, a_2, \dots, a_m\} \subseteq \Sigma$ is a finite set of items. The size of A, denoted $|A|$, is m. A *sequence* s is of the form $s = \langle s_1 s_2 \dots s_n \rangle$ where each s_i is an itemset, and n is the length of the sequence. A *database* $\mathcal{D}$ is a multiset of sequences over Σ. A sequence $s = \langle s_1 \dots s_m \rangle$ with $s_i \in \Sigma$ is contained in a sequence $t = \langle t_1 \dots t_n \rangle$ with $m \leq n$, written $s \sqsubseteq t$, if $s_i \subseteq t_{e_i}$ for $1 \leq i \leq m$ and an increasing sequence $(e_1 \dots e_m)$ of positive integers $e_i \in [n]$, called an *embedding* of s in t. For example, we have $\langle a(cd) \rangle \sqsubseteq \langle ab(cde) \rangle$ relative to embedding $(1, 3)$. Here, (cd) denotes the itemset made of items c and d. Given a database $\mathcal{D}$, the *cover* of a sequence s is the set of sequences in $\mathcal{D}$ that contain s: $cover(s, \mathcal{D}) = \{t \in D | s \sqsubseteq t\}$. The number of sequences in $\mathcal{D}$ containing s is called its *support*, that is, $supp(s, \mathcal{D}) = |cover(s, \mathcal{D})|$. For an integer *minsup* (that is often referred to as the *minimum support threshold*), the problem of *frequent sequence mining* is about discovering all sequences s such that $supp(s, \mathcal{D}) \geq minsup$. Each sequence that satisfies this requirement is called a *(sequential) pattern*. For $minsup = 2$ we can see how $\langle a \rangle$, $\langle b \rangle$, $\langle c \rangle$, $\langle a\ b \rangle$, $\langle a\ c \rangle$, $\langle b\ c \rangle$ and $\langle a\ b\ c \rangle$ are common patterns in the database $\mathcal{D}$ reported in Table 1 without considering the reference classes.

Table 1. Example of a sequence dataset, in tabular format (left) and encoded with ASP (right). Each sequence has a class label, that is used in CSPM.

ID	**Sequence**	**Class**
1	$\langle d\ a\ a\ c \rangle$	C_1
2	$\langle a\ c\ b\ c \rangle$	C_1
3	$\langle a\ c \rangle$	C_1
4	$\langle b \rangle$	C_1
5	$\langle a\ b\ c \rangle$	C_2
6	$\langle a\ b\ c \rangle$	C_2
7	$\langle c \rangle$	C_2

```
cl(1,c1). seq(1,1,d). seq(1,2,a). seq(1,3,a). seq(1,4,c).
cl(2,c1). seq(2,1,a). seq(2,2,c). seq(2,3,b). seq(2,4,c).
cl(3,c1). seq(3,1,a). seq(3,2,c).
cl(4,c1). seq(4,1,b).
cl(5,c2). seq(5,1,a). seq(5,2,b). seq(5,3,c).
cl(6,c2). seq(6,1,a). seq(6,2,b). seq(6,3,c).
cl(7,c2). seq(7,1,c).
```

A *contrast sequential pattern* is defined as a sequential pattern that occurs frequently in one sequence dataset but not in the others [3]. We start by introducing the concept of *growth rate*. Given two sequence datasets, $\mathcal{D}_1$ labelled with class C_1 and $\mathcal{D}_2$ labelled with class C_2, the growth rate from $\mathcal{D}_2$ to $\mathcal{D}_1$ of a sequential pattern s is defined as:

$$GR_{C_1}(s) = \frac{supp(s, \mathcal{D}_1)/|\mathcal{D}_1|}{supp(s, \mathcal{D}_2)/|\mathcal{D}_2|} \tag{2}$$

If $supp(s, \mathcal{D}_2) = 0$ and $supp(s, \mathcal{D}_1) \neq 0$ then $GR_{C_1}(s) = \infty$.

In the same way, the growth rate from $\mathcal{D}_1$ to $\mathcal{D}_2$ of s is defined as:

$$GR_{C_2}(s) = \frac{supp(s, \mathcal{D}_2)/|\mathcal{D}_2|}{supp(s, \mathcal{D}_1)/|\mathcal{D}_1|} \tag{3}$$

If $supp(s, \mathcal{D}_1) = 0$ and $supp(s, \mathcal{D}_2) \neq 0$ then $GR_{C_2}(s) = \infty$.

The *contrast rate* of s is denoted as:

$$CR(s) = max\{GR_{C_1}, GR_{C_2}\} \tag{4}$$

If $GR_{C_1}(s) = 0$ and $GR_{C_2}(s) = 0$ then $CR(s) = \infty$.

A sequence s in a sequences dataset is said to be a *contrast sequential pattern* if its contrast rate is not lower than the given threshold: $CR(s) \geq mincr$. Unlike frequent sequential pattern mining, contrast sequential pattern mining can discover the characteristics of different classes in sequences datasets, which has been widely used in sequential data analysis, such as protein/DNA dataset analysis, anomaly detection, and customer behavior analysis.

With reference to the dataset in Table 1, we split $\mathcal{D}$ into $\mathcal{D}_1$ and $\mathcal{D}_2$ according to the classes C_1 and C_2, respectively. For example, the sequence $s = \langle a\ c \rangle$ has $supp(s, \mathcal{D}_1) = 3$, $supp(s, \mathcal{D}_2) = 2$, $GR_{C_1}(s) = 1.125$, $GR_{C_2}(s) = 0.89$, and $CR(s) = 1.125$. Given $t = \langle a\ b\ c \rangle$, it has $supp(t, \mathcal{D}_1) = 1$, $supp(t, \mathcal{D}_2) = 2$, $GR_{C_1}(t) = 0.375$, $GR_{C_2}(t) = 2.67$, and $CR(t) = 2.67$. If, *e.g.*, $mincr = 1$, we can conclude that s and t are contrast sequential patterns for C_1 and C_2 resp., because $CR(s) \geq mincr$ and $CR(t) \geq mincr$.

4 ASP encoding for contrast sequence mining

In this section we describe the proposed ASP encoding and discuss the rationale behind. Since CSPM merges the two notions of sequential pattern and contrast pattern, it is necessary to first extract the frequent sequential patterns from the input sequences (see Section 4.1) and then check which of these regularities are actually contrast sequential patterns (see Section 4.2).

4.1 From sequences to sequential patterns

The sub-problem of mining frequent sequential patterns is encoded according to the principles outlined in [5]. However, in our case, we need to consider also the reference class. In particular, $\mathcal{D}$ is represented as a collection of ASP facts of the kind *seq(s, p, i)* and *cl(s, c)*, where the *seq* predicate says that an item i occurs at position p in a sequence s while the *cl* predicate says that s is labelled with class c. Table 1 shows the ASP encoding of a sequence dataset.

Besides *minsup* and *mincr*, *maxlen* and *minlen* are introduced to denote the maximum and the minimum pattern length, respectively. Also, *c1* and *c2* stand for the classes C_1 and C_2, respectively. The lower the value of *minsup* and *mincr* the more patterns will be extracted; the lower the value of *maxlen*, the smaller the ground program will be. Therefore the parameters allow a tuning for the program efficiency. Finally, each answer set comprises a single pattern of interest. More precisely, an answer set represents a (contrast) sequential pattern $s = \langle s_i \rangle_{i \leq th \leq m}$ such that $1 \leq m \leq maxlen$ from atoms $pat(1, s_1), ..., pat(m, s_m)$. The first argument expresses the position of the item inside the pattern. For example the atoms *pat(1, a)*, *pat(2, b)* and *pat(3, c)* describe a contrast sequential pattern $\langle a\ b\ c\rangle$ for the database in Table 1.

For this sub-problem the basic "fill-gaps" encoding provided by [10] can be used with little modifications.

4.2 From frequent sequences to contrast patterns

The sub-problem of filtering contrast patterns out of the frequent sequences discovered by the encoding reported in Section 4.1 requires additional ASP rules.

```
1  card(Card, c1) :- Card = #count{T : cl(T, c1)}.
2  card(Card, c2) :- Card = #count{T : cl(T, c2)}.
3
4  sup(Sup, c1) :- Sup = #count{T : support(T), seq(T, _, _), cl(T, c1)}.
5  sup(Sup, c2) :- Sup = #count{T : support(T), seq(T, _, _), cl(T, c2)}.
6
7  gr_rate("inf", c1) :- sup(Sup1, c1), Sup1 != 0, sup(0, c2).
8  gr_rate("inf", c2) :- sup(Sup2, c2), Sup2 != 0, sup(0, c1).
9  gr_rate(@gr(Sup1, Card1, Sup2, Card2), c1) :- sup(Sup1, c1),
10   card(Card1, c1), sup(Sup2, c2), card(Card2, c2), Sup1 > 0, Sup2 > 0.
11 gr_rate(@gr(Sup2, Card2, Sup1, Card1), c2) :- sup(Sup1, c1),
12   card(Card1, c1), sup(Sup2, c2), card(Card2, c2), Sup1 > 0, Sup2 > 0.
13
14 contr_pat(yes, Class) :- gr_rate("inf", Class).
15 contr_pat(@csp(Cr, mincr), Class) :- gr_rate(Cr, Class),  Cr != "inf".
16
17 :- contr_pat(no, c1), contr_pat(no, c2).
```

In the code above Lines 1-2 compute the cardinality of the datasets $\mathcal{D}_1$ and $\mathcal{D}_2$ whereas Lines 4-5 compute the support of a pattern s in $\mathcal{D}_1$ and $\mathcal{D}_2$ respectively. Lines 9-10 calculate $GR_{C_1}(s)$ in accordance with the formula in Eq. (2), while Line 6 capture the case of $GR_{C_1}(s) = \infty$. ASP does not support the computation of formulas that return decimal values. For this reason, a Python script has been developed which can be called from within ASP (with the @ command followed by the function name). The result will no longer be treated in ASP as a constant but rather as a string. Analogously, Lines 12-13 encode the computation of GR_{C_2} according to Eq. (3) and Line 7 concerns the infinite case for GR_{C_2}. Finally, Lines 14-15 check if the sequence s in hand is a contrast pattern for either C_1 or C_2 by means of a Python function because it compares decimal numbers. If the growth rate is less than *mincr*, a constant *no* is returned, *yes* otherwise. Lines 14-15 set the first term of the *contrast_pattern* to *yes* in accordance with the formulas in Eq. (2) and (3), respectively. The constraint at Line 17 discards all answer sets that do not represent contrast patterns for any of the two classes.

Below, the same example of CSPM reported at the end of Section 3.2 is solved by running our ASP encoding over the ASP facts in Table 1 with *minsup = 20%* and *mincr = 1*.

```
pat(1,a) pat(2,c) gr_rate("0.89",c2) contr_pat(no,c2) gr_rate("1.125",c1) contr_pat(yes,c1)
pat(1,a) pat(2,b) pat(3,c) gr_rate("2.67",c2) contr_pat(yes,c2) gr_rate("0.375",c1) contr_pat(no,c1)
```

Table 2. Features of iPRG and UNIX User sub-datasets: The number of distinct symbols, the number of sequences, the total number of symbols in the dataset, the maximum sequence length, the average sequence length, and the density (calculated by $\frac{||D||}{|\Sigma||D|}$.)

Dataset	$\|\Sigma\|$	$\|\mathbf{D}\|$	$\|\|\mathbf{D}\|\|$	**max**$\|\mathbf{T}\|$	**avg**$\|\mathbf{T}\|$	**density**
iPRG	21	8628	111,743	12	11.95	0.62
iPRG_25_25	20	50	657	12	11.88	0.64
iPRG_100_100	20	200	2591	12	11.83	0.64
iPRG_500_500	21	1000	12,933	12	11.92	0.62
iPRG_1000_1000	21	2000	25,841	12	11.91	0.61
UNIX	2672	9099	165,748	1256	18.22	0.01
UNIX_25_25	70	50	365	55	7.3	0.10
UNIX_100_100	178	200	2281	175	11.41	0.06
UNIX_500_500	420	1000	13,289	187	13.29	0.03
UNIX_755_755	540	1510	20,234	214	13.4	0.02

5 Experiments

In this section we examine the computational behavior of the ASP encoding described in Section 4. In pattern mining, it is usual to evaluate the effectiveness (number of extracted patterns) and the time and space efficiency of an algorithm. Moreover in ASP-based DPM approaches it is important to know the solver and grounder time. To this end, we conducted experiments on the following datasets:

– iPRG: each transaction is a sequence of peptides that is known to cleave in presence of a Trypsin enzyme,[3]
– UNIX User: each transaction is a sequence of shell commands executed by a user during one session.[4]

We have chosen these datasets because (i) they are suitable for the task considered in this paper (classified sequences), (ii) they have been already used in the DPM literature, in particular in [10,19] although for a different task, and (iii) they are publicly available. Notably, transactions in both datasets are labelled with one of two classes, *pos* and *neg*.

In the following we report and discuss the results obtained from scalability tests on iPRG (Section 5.1) and UNIX User (Section 5.2). As a solver, we have used the version 5.4.0 of clingo, with default solving parameters. The timeout (TMO) has been set to 5 hours. The ASP programs were run on a laptop computer with Windows 10 (with Ubuntu 20.04.4 subsystem), AMD Ryzen 5 3500U @ 2.10 GHz, 8GB RAM without using the multi-threading mode of clingo. Multi-threading reduces the mean runtime but introduces variance due to the random allocation of tasks. Such variance is inconvenient for interpreting results with repeated executions.

[3] https://dtai.cs.kuleuven.be/CP4IM/cpsm/datasets.html
[4] https://archive.ics.uci.edu/ml/datasets/UNIX+User+Data

As regards the comparison with alternative solutions to the CSPM problem in hand, it has not been possible to run experiments with dedicated algorithms like [23,22] since their code is not available. Thus, we have considered a hybrid ASP-based approach inspired by [20]. The results from the comparative evaluation are reported and discussed in Section 5.3.

Full code and detailed experimental results are available online.[5].

5.1 Scalability tests on iPRG

In order to study the behaviour of the ASP encoding over iPRG, we have created several subsets of this dataset of increasing size (Table 2).

For each sub-dataset, we show runtime and memory behaviour of the encoding in two settings (see Table 3). On the left the minimum support threshold (*minsup*) varies from 10% to 50% while *mincr = 3*, *minlen = 2*, and *maxlen = 5* remain fixed, and on the right the minimum contrast rate (*mincr*) varies from 1 to 5 while *minsup = 20%*, *minlen = 2*, and *maxlen = 5* remain fixed. When we increase *minsup* and/or *mincr*, the number of patterns and the runtime decrease. The minimum support threshold at 20% represents a cut point as regards the number of patterns found. Obviously, as the size of the dataset increases, the runtime and memory parameters grow up but the grounding phase ($time - solv.\ t.$) not significantly. In the case of Table 3(e, f, g, h) the execution is interrupted as it exceeds the time limit of 5 hours. Also, the choice of $minsup = 10\%$ influences the dimension of the program and consequently the execution time as shown in Table 3(e, g). We recall that these tests were conducted using a single thread for the reasons mentioned before. With more threads the total execution time will decrease. As regards memory usage, this grows up in proportion to the size of the input dataset but remains stable as *minsup* or *mincr* increases. Low *minsup* values allow to find more patterns but at a higher runtime. In fact, the thresholds *minsup* and *mincr* have the pruning function, the former for frequent sequences and the latter for contrast sequences.

Below, as an illustrative example, we report some contrast sequential patterns found in iPRG_100_100 for $minsup = 10\%$ and $mincr = 3$.

```
1   pat(1,4) pat(2,5) pat(3,9) gr_rate("0.05",neg) contr_pat(no,neg) gr_rate("19.0",pos) contr_pat(yes,pos)
2   pat(1,9) pat(2,9) pat(3,2) pat(4,11) gr_rate("inf") contr_pat(yes,pos)
3   pat(1,11) pat(2,16) pat(3,8) gr_rate("12.0") contr_pat(yes,neg) gr_rate("0.08") contr_pat(no,pos)
4   pat(1,2) pat(2,16) pat(3,8) gr_rate("inf") contr_pat(yes,neg)
```

Line 1 describes the contrast pattern $\langle 4\ 5\ 9 \rangle$ that represents the sequence $\langle Q\ \ P\ \ N \rangle$ of peptides.[6] This pattern has high contrast rate for the *pos* class. Conversely, Line 3 shows a contrast pattern ($\langle 11\ 16\ 8 \rangle = \langle L\ D\ K \rangle$) for the *neg* class. The patterns $\langle 9\ 9\ 2\ 11 \rangle = \langle N\ N\ I\ L \rangle$ and $\langle 2\ 16\ 8 \rangle = \langle I\ D\ K \rangle$ at Lines 2 and 4, respectively, have an important property. They occur only in the *pos* and *neg* classes respectively because the growth rate value is infinite.

[5] https://github.com/mpia3/Contrast-Sequential-Pattern-Mining.git

[6] The meaning of each item can be found at the link where the dataset is published.

Table 3. Number of patterns, runtime (seconds), solver time (seconds) and memory consumption (MB) on all iPRG sub-datasets. TMO means that the execution has exceeded the imposed 5-hour timeout.

(a) iPRG_25_25, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	712	3.065	2.96	25.96
20%	24	1.550	1.35	23.89
30%	2	1.050	0.86	23.89
40%	0	0.480	0.34	23.89
50%	0	0.243	0.08	23.89

(b) iPRG_25_25, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	0	0.085	0.00	22.31
2	0	0.076	0.00	21.93
3	0	0.086	0.00	21.67
4	0	0.086	0.00	22.31
5	0	0.086	0.00	22.18

(c) iPRG_100_100, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	561	47.553	46.82	87.05
20%	15	22.585	21.43	60.05
30%	0	10.279	9.02	60.04
40%	0	5.474	4.34	60.00
50%	0	3.488	2.23	60.00

(d) iPRG_100_100, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	72	20.290	19.12	59.16
2	37	21.855	20.74	61.77
3	15	21.974	20.73	60.05
4	9	18.289	17.15	60.04
5	8	18.338	17.11	59.98

(e) iPRG_500_500, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	71	TMO	TMO	852.24
20%	12	3543.002	3524.08	852.24
30%	0	1712.463	1692.43	852.24
40%	0	140.521	120.08	852.24
50%	0	98.535	79.49	852.24

(f) iPRG_500_500, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	71	TMO	TMO	852.24
2	20	403.259	383.63	859.90
3	12	3440.552	3421.60	852.24
4	8	606.101	586.90	851.83
5	4	TMO	TMO	1447.55

(g) iPRG_1000_1000, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	12	TMO	TMO	3258.44
20%	3	TMO	TMO	3242.55
30%	0	7375.746	7284.13	3253.90
40%	0	2061.364	1972.19	3232.85
50%	0	TMO	TMO	3433.55

(h) iPRG_1000_1000, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	12	TMO	TMO	3258.44
2	18	TMO	TMO	3166,26
3	3	TMO	TMO	3207.61
4	0	TMO	TMO	3172.99
5	0	TMO	TMO	3118.20

5.2 Scalability tests on UNIX User

As regards the UNIX User dataset we have created several subsets of the same size as iPRG (see Table 2), except for one (namely, UNIX_755_755) where the rationale behind the size of 755 is the fact that the positive sequences are only 755 in the original dataset and we wanted to keep the two classes balanced.

Analogously to the experiments conducted with iPRG, we report runtime and memory usage for two batches of tests (see Table 4). One concerns the variation of the minimum support threshold (*minsup*) from 10% to 50%, while keeping *mincr = 3*, *minlen = 2*, and *maxlen = 5* fixed. The other concerns the variation of the minimum contrast rate (*mincr*) from 1 to 5 while leaving unchanged *minsup = 20%*, *minlen = 2*, and *maxlen = 5*. The particularity of the dataset lies in the size of its alphabet, clearly higher than iPRG. Such a size affects sequences with a higher average length. In fact the largest sequence, whatever the size of the dataset considered, is clearly larger than iPRG. This also affects the number of patterns found, lower than for iPRG because the single sequence has much more variance. Moreover, the alphabet size affects the overall time. From a comparison between Tables 3(e, f) and 4(e, f) it is clear the difference in magnitude of the time taken. All tables show the same behavior in memory as iPRG. When the data size is high, the overall time exceeds the timeout only when support threshold is less or equal than 20% (see Table 4(e, g, h)).

Table 4. Number of patterns, runtime (seconds), solver time (seconds) and memory consumption (MB) on all UNIX User sub-datasets. TMO means that the execution has exceeded the imposed 5-hour timeout.

(a) UNIX_25_25, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	335	0.414	0.26	23.5
20%	1	0.105	0.02	22.93
30%	0	0.095	0.01	22.93
40%	0	0.086	0.01	22.93
50%	0	0.087	0.00	21.38

(b) UNIX_25_25, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	1	0.132	0.01	22.94
2	1	0.103	0.02	22.94
3	1	0.099	0.02	22.93
4	1	0.104	0.02	22.93
5	1	0.103	0.02	21.21

(c) UNIX_100_100, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	18	3.679	2.69	59.58
20%	0	1.792	1.02	59.57
30%	0	1.342	0.56	59.57
40%	0	0.973	0.24	37.48
50%	0	0.886	0.14	37.35

(d) UNIX_100_100, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	0	2.409	1.4	59.4
2	0	1.841	1.10	59.55
3	0	0.818	1.03	59.57
4	0	1.789	1.04	59.53
5	0	1.753	1.04	59.56

(e) UNIX_500_500, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	9	TMO	TMO	850.79
20%	1	129.941	110.99	850.79
30%	0	59.270	37.88	850.79
40%	0	50.735	30.39	850.79
50%	0	35.419	15.68	850.79

(f) UNIX_500_500, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	1	74.553	53.96	855.96
2	1	123.825	104.54	846.29
3	1	123.167	103.68	848.51
4	1	125.537	106.22	850.38
5	1	128.747	109.14	850.38

(g) UNIX_755_755, mincr = 3

minsup	#pat	time	solv. t.	memory
10%	1	TMO	TMO	2653.12
20%	0	TMO	TMO	2642.25
30%	0	235.760	188.53	1849.92
40%	0	146.888	99.52	1850.01
50%	0	120.360	72.95	1850.00

(h) UNIX_755_755, minsup = 20%

mincr	#pat	time	solv. t.	memory
1	0	286.354	238.35	1848.77
2	0	13,709.513	13,661.11	1851.55
3	0	TMO	TMO	2642.25
4	0	TMO	TMO	1848.83
5	0	TMO	TMO	3766.38

Some contrast sequential patterns mined from UNIX_25_25 are shown below as an illustrative example. Each item represents a UNIX command.[7]

```
1 pat(1,12) pat(2,14) pat(3,15) pat(4,13) pat(5,12) gr_rate("inf",neg) contr_pat(yes,neg)
2 pat(1,103) pat(2,2611) pat(3,29) pat(4,2812) gr_rate("inf",pos) contr_pat(yes,pos)
```

Line 1 shows the sequence $\langle 12\ 14\ 15\ 13\ 12\rangle = \langle fg \mid more\ finger\ fg\rangle$ that is a contrast pattern only for the *neg* class while Line 2 a sequence found only for the *pos* class: $\langle 103\ 2611\ 29\ 2812\rangle = \langle quota\ emacs - nw\ netscape\ assoc.out\rangle$.

[7] The reader can find the conversion table at the link where the dataset is published.

5.3 Comparison with a hybrid ASP-based approach

As a baseline for a comparative evaluation we have considered a two-step approach that features an ASP filtering on top of a dedicated algorithm for SPM. In the first step, *PrefixSpan* [11] is applied to discover frequent sequential patterns, while in the second step, the patterns are post-processed by using the ASP rules reported in Section 4.2 to find the constrasting ones. The resulting hybrid *PrefixSpan+ASP* approach has been applied on the same datasets (see Table 2) and with the same parameters used in the scalability tests reported in the previous two sections. The results obtained with the pure ASP and the hybrid method as regards the time and memory dimensions are graphically presented in Figure 1 in a comparative way.

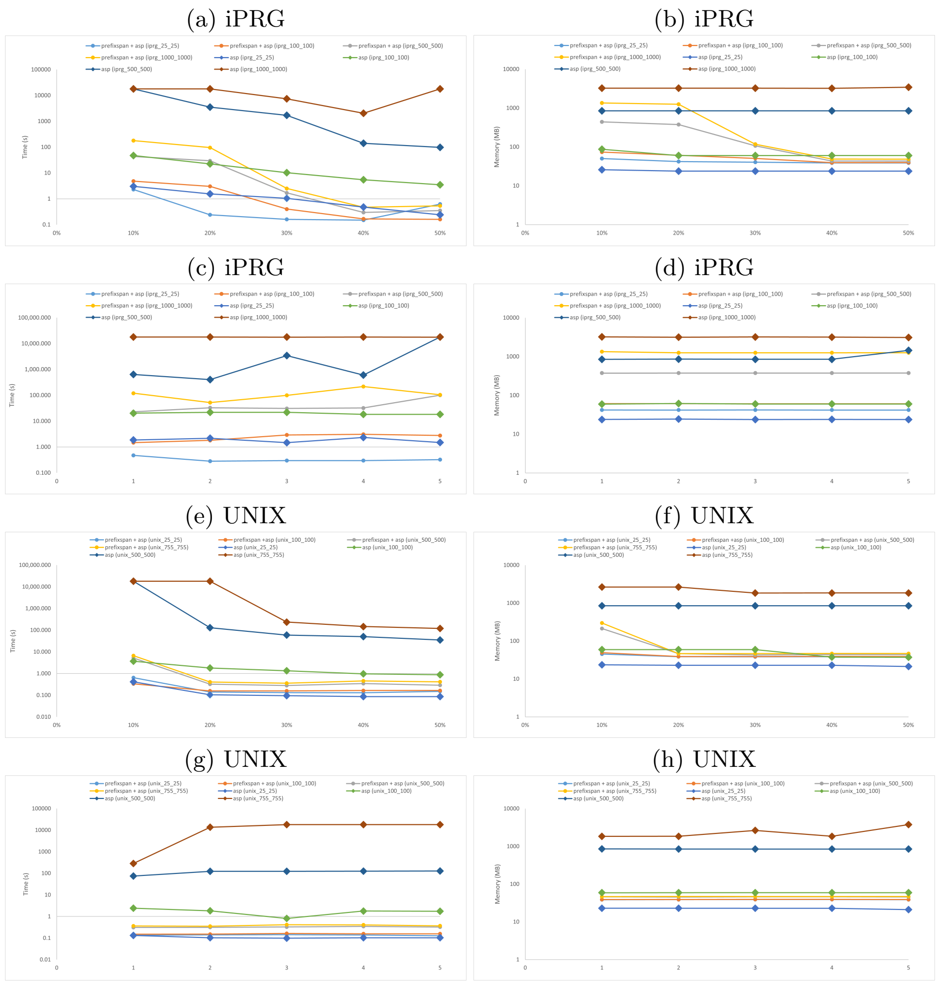

Fig. 1. Comparison between pure and hybrid ASP-based approaches.

It is interesting to note the behavior of the two approaches. For both iPRG and UNIX User, the one-shot approach performs slightly better than the two-step approach when the dataset size is not large (up to 100-100 sequences). This can be observed especially for memory usage (see Figure 1 (b, d, f, h)).

6 Conclusions and future work

This article has presented a declarative approach to the CSPM task which is based on ASP. To the best of our knowledge, this is the first proposed ASP encoding for this task. It takes advantage of the Python interface for Clingo to design more complex ASP programs, e.g., numerical computation as in our case. The encoding has been extensively evaluated on real-world (publicly available) datasets to draw conclusions about the efficiency of the approach. Low *minsup* and *mincr* values allow to find more patterns. However, this comes with an increasing runtime. So, the results from the scalability tests are promising although they can not be considered conclusive about the validity of the approach. For this reason, a comparison with two-step approaches is particularly interesting. The results obtained with the hybrid PrefixSpan+ASP approach confirm and complement the conclusions of [20] concerning the advantages of hybrid approaches over pure approaches as regards time performance. Our contribution is the empirical evidence for a DPM task (namely, CSPM) that was not covered by [20]. In particular, the analysis of memory usage provides new hints on the behaviour of ASP-based DPM solutions.

Much work needs to be done for the future. In order to improve the performance, we intend to explore the extraction of condensed representations (e.g. maximal and closed patterns) in the context of CSPM. Also, further experiments are needed to complete the efficiency analysis of our ASP encoding, such as the ones aimed at studying the interplay between memory usage and pattern length as done in [10]. Another direction for future development of the present work is to consider other forms of contrast pattern as described in [3]. Our proposal is indeed general enough to enable the encoding of several types of constraints. The addition/deletion of constraints allows the modeling of problem variants. Overall, an advantage of DPM is that for most well-specified tasks, the development effort is significantly lower than for procedural approaches. We do not expect DPM to be competitive with dedicated algorithms, but to take advantage of the versatility of declarative frameworks to propose pattern mining tools that could exploit background knowledge during the mining process to extract less but meaningful patterns. To this aim, we plan to enrich the datasets for future experiments with domain knowledge, *e.g.*, chemistry or biology related constraints in the iPRG dataset.

Acknowledgments This work was partially supported by the project FAIR - Future AI Research (PE00000013), under the NRRP MUR program funded by the NextGenerationEU.

References

1. Besnard, P., Guyet, T.: Declarative mining of negative sequential patterns. In: DPSW 2020-1st Declarative Problem Solving Workshop. pp. 1–8 (2020)
2. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011). https://doi.org/10.1145/2043174.2043195, `http://doi.acm.org/10.1145/2043174.2043195`
3. Chen, Y., Gan, W., Wu, Y., Yu, P.S.: Contrast pattern mining: A survey. arXiv preprint arXiv:2209.13556 (2022)
4. Dong, G., Bailey, J.: Contrast data mining: concepts, algorithms, and applications. CRC Press (2012)
5. Gebser, M., Guyet, T., Quiniou, R., Romero, J., Schaub, T.: Knowledge-based sequence mining with asp. In: IJCAI 2016-25th International joint conference on artificial intelligence. p. 8. AAAI (2016)
6. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo= ASP + control: Preliminary report. arXiv preprint arXiv:1405.3694 (2014)
7. Guns, T., Dries, A., Nijssen, S., Tack, G., De Raedt, L.: Miningzinc: A declarative framework for constraint-based mining. Artificial Intelligence **244**, 6–29 (2017)
8. Guyet, T., Happe, A., Dauxais, Y.: Declarative sequential pattern mining of care pathways. In: Artificial Intelligence in Medicine: 16th Conference on Artificial Intelligence in Medicine, AIME 2017, Vienna, Austria, June 21-24, 2017, Proceedings 16. pp. 261–266. Springer (2017)
9. Guyet, T., Moinard, Y., Quiniou, R.: Using answer set programming for pattern mining. arXiv preprint arXiv:1409.7777 (2014)
10. Guyet, T., Moinard, Y., Quiniou, R., Schaub, T.: Efficiency analysis of ASP encodings for sequential pattern mining tasks. In: Advances in Knowledge Discovery and Management, pp. 41–81. Springer (2018)
11. Han, J., Pei, J., Mortazavi-Asl, B., Pinto, H., Chen, Q., Dayal, U., Hsu, M.: PrefixSpan: Mining sequential patterns efficiently by prefix-projected pattern growth. In: proceedings of the 17th international conference on data engineering. pp. 215–224. IEEE (2001)
12. Jabbour, S., Sais, L., Salhi, Y.: Decomposition based SAT encodings for itemset mining problems. In: Pacific-Asia Conference on Knowledge Discovery and Data Mining. pp. 662–674. Springer (2015)
13. Leone, N., Allocca, C., Alviano, M., Calimeri, F., Civili, C., Costabile, R., Fiorentino, A., Fuscà, D., Germano, S., Laboccetta, G., Cuteri, B., Manna, M., Perri, S., Reale, K., Ricca, F., Veltri, P., Zangari, J.: Enhancing DLV for large-scale reasoning. In: Balduccini, M., Lierler, Y., Woltran, S. (eds.) Logic Programming and Nonmonotonic Reasoning - 15th International Conference, LPNMR 2019, Philadelphia, PA, USA, June 3-7, 2019, Proceedings. Lecture Notes in Computer Science, vol. 11481, pp. 312–325. Springer (2019). https://doi.org/10.1007/978-3-030-20528-7_23, `https://doi.org/10.1007/978-3-030-20528-7_23`
14. Lifschitz, V.: Answer sets and the language of answer set programming. AI Magazine **37**(3), 7–12 (2016)
15. Lisi, F.A., Sterlicchio, G.: Declarative pattern mining in digital forensics: Preliminary results. In: Calegari, R., Ciatto, G., Omicini, A. (eds.) Proceedings of the 37th Italian Conference on Computational Logic, Bologna, Italy, June 29 - July 1, 2022. CEUR Workshop Proceedings, vol. 3204, pp. 232–246. CEUR-WS.org (2022), `http://ceur-ws.org/Vol-3204/paper_23.pdf`

16. Lisi, F.A., Sterlicchio, G.: Mining sequences in phone recordings with answer set programming. In: Bruno, P., Calimeri, F., Cauteruccio, F., Maratea, M., Terracina, G., Vallati, M. (eds.) Joint Proceedings of the 1st International Workshop on HYbrid Models for Coupling Deductive and Inductive ReAsoning (HYDRA 2022) and the 29th RCRA Workshop on Experimental Evaluation of Algorithms for Solving Problems with Combinatorial Explosion (RCRA 2022) co-located with the 16th International Conference on Logic Programming and Non-monotonic Reasoning (LPNMR 2022), Genova Nervi, Italy, September 5, 2022. CEUR Workshop Proceedings, vol. 3281, pp. 34–50. CEUR-WS.org (2022), `http://ceur-ws.org/Vol-3281/paper4.pdf`
17. Lisi, F.A., Sterlicchio, G.: A declarative approach to contrast pattern mining. In: Dovier, A., Montanari, A., Orlandini, A. (eds.) AIxIA 2022 - Advances in Artificial Intelligence - XXIst International Conference of the Italian Association for Artificial Intelligence, AIxIA 2022, Udine, Italy, November 28 - December 2, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13796, pp. 17–30. Springer (2023). https://doi.org/10.1007/978-3-031-27181-6_2, `https://doi.org/10.1007/978-3-031-27181-6_2`
18. Mooney, C.H., Roddick, J.F.: Sequential pattern mining–approaches and algorithms. ACM Computing Surveys (CSUR) **45**(2), 1–39 (2013)
19. Negrevergne, B., Guns, T.: Constraint-based sequence mining using constraint programming. In: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research. pp. 288–305. Springer (2015)
20. Paramonov, S., Stepanova, D., Miettinen, P.: Hybrid ASP-based approach to pattern mining. Theory Pract. Log. Program. **19**(4), 505–535 (2019). https://doi.org/10.1017/S1471068418000467, `https://doi.org/10.1017/S1471068418000467`
21. Samet, A., Guyet, T., Negrevergne, B.: Mining rare sequential patterns with ASP. In: ILP 2017-27th International Conference on Inductive Logic Programming (2017)
22. Wu, Y., Wang, Y., Li, Y., Zhu, X., Wu, X.: Top-k self-adaptive contrast sequential pattern mining. IEEE Transactions on Cybernetics **52**(11), 11819–11833 (2022). https://doi.org/10.1109/TCYB.2021.3082114
23. Zheng, Z., Wei, W., Liu, C., Cao, W., Cao, L., Bhatia, M.: An effective contrast sequential pattern mining approach to taxpayer behavior analysis. World Wide Web **19**, 633–651 (2016)