Mining Sequences in Phone Recordings with Answer Set Programming

Francesca A. Lisi[1,*], Gioacchino Sterlicchio[2]

[1]Dipartimento di Informatica, University of Bari "Aldo Moro", Via E. Orabona 4, Bari, 70125, Italy
[2]Dept. Mechanical Engineering, Polytechnic University of Bari, Via G. Amendola 126/b - 70126 Bari, Italy

Abstract

The analysis of phone recordings is one of the activities typically performed in the Digital Forensics practice. It provides information such as the geographical position of some suspect, useful to reconstruct the network of contacts. In this work we have considered the problem of analyzing a dataset of anonymized phone recordings made available within the DigForASP project by looking for sequential patterns in the recordings. The approach followed leverages the expressive and inferential power of the Answer Set Programming paradigm. The experiments reported here concern the extraction of closed and maximal patterns (condensed representations), which significantly reduces the number of returned patterns without loss of information.

Keywords

Sequential Pattern Mining, Answer Set Programming, Digital Forensics.

1. Introduction

In the context of the forensic practice, material found in digital devices, often in relation to mobile devices and computer crime, is typically a subject of investigation. The identification, acquisition, preservation, analysis and presentation of this material, by means of specialized software, and according to specific regulations, are indeed the phases of the procedure usually followed by professionals of *Digital Forensics* (DF). In particular, the phase of so-called *Evidence Analysis* involves the examination and aggregation of evidence about possible crimes and crime perpetrators collected from various electronic devices in order to reconstruct events, event sequences and scenarios related to a crime. The results from this phase are then communicated to the other actors in the forensic context, such as law enforcement bodies, lawyers and judges.

Recently, it has been suggested that the phase of Evidence Analysis has particular requirements that make the use of logic-based AI techniques, especially from the areas of *Knowledge Representation* and *Automated Reasoning*, a much more promising approach, with the potential of becoming a breakthrough in the state-of-the-art of research in the field and of introducing

HYDRA - RCRA 2022: 1st International Workshop on HYbrid Models for Coupling Deductive and Inductive ReAsoning and 29th RCRA workshop on Experimental evaluation of algorithms for solving problems with combinatorial explosion

*Corresponding author, affiliated also to the *Centro Interdipartimentale di Logica e Applicazioni* (CILA) of the University of Bari.

FrancescaAlessandra.Lisi@uniba.it (F. A. Lisi); ninnisterlicchio@gmail.com (G. Sterlicchio)

http://www.di.uniba.it/~lisi/ (F. A. Lisi)

0000-0001-5414-5844 (F. A. Lisi); 0000-0002-2936-0777 (G. Sterlicchio)

disruptive innovations in the forensic practice [1, 2]. The ultimate goal of Evidence Analysis is indeed the formulation of verifiable evidence that can be rationally presented in a trial. Under this perspective, the results provided by, *e.g.*, ML classifiers or other types of "black box" AI systems can be regarded as having the same value as human witness' suspicions. Therefore, they could not be used as legal evidence. Conversely, logic-based AI methods provide a broad range of proof-based reasoning functionalities that can be implemented in a declarative framework where the problem specification and the computational program are closely aligned. This has the benefit that the correctness of the resulting systems can be formally verified. Moreover, new methods for visualising and explaining the computed answers (*e.g.*, based on graphical languages) have been recently proposed. So one can not only model and solve relevant problems, but also effectively communicate the conclusions (and their proofs) in a transparent, comprehensible and justified way. This approach is the one adopted by the COST Action "Digital forensics: evidence analysis via intelligent systems and practices" (DigForASP)[1], a large international cooperation network which aims at promoting formal and verifiable AI methods and techniques for Evidence Analysis [3].

A typical DF problem is the analysis of phone recordings. Indeed, during the investigation of a crime, it is common to analyze the communications of a particular suspect. Since nowadays mobile phones are objects owned by anyone, it can be useful for investigators to analyze the calls or messages exchanged. The telephone records contain all the traces of communications (calls, SMS, and all the data traffic) concerning a specific user over a certain period of time. Note that phone records do not trace important data such as the audio of calls sent or received. In fact, they only provide a trace of the communication that has taken place but not its content. The phone records can be requested by the Judicial Authority if deemed useful in order to carry out investigations involving the individual owner of the phone. Correctly analyzing the telephone records is essential to obtain useful hints. Depending on the analysis, different types of information can be extracted. The records are typically analyzed for comparing the geographical positions with respect to the declarations, and for reconstructing the network of contacts of a single user in order to trace which conversations (s)he has had with whom, where and when. In this paper we consider the problem of mining sequences in phone recordings in order to discover frequent sequential patterns. In particular, we propose a declarative approach based on *Answer Set Programming* (ASP) which adapts previous work in sequential pattern mining [4, 5] to the problem in hand. The proposed approach is in the spirit of DigForASP and appears a promising support to the analysis of events and sequences of events in scenarios of interest to DF experts. This paper extends the preliminary work reported in [6] with a significant refinement aimed at addressing some limits of pattern mining such as the size of the output. In particular, here we explore the use of condensed representations for sequential patterns.

The paper is organized as follows. In Section 2 we provide the necessary preliminaries on ASP and sequential pattern mining. In Section 3 we describe the specific case study for the analysis of mobile phone recordings. In Section 4 we report the details of our ASP-based approach, including the ASP encodings, and the experimental results obtained by comparing the behaviour of the encodings. In Section 5 we conclude by commenting the ongoing work and by outlining some promising directions for research.

[1] https://digforasp.uca.es/

2. Preliminaries

2.1. Answer Set Programming

In the following we give a brief overview of the syntax and semantics of disjunctive logic programs in ASP. The reader can refer to, e.g, [7] for a more extensive introduction to ASP.

Let U be a fixed countable set of (domain) elements, also called *constants*, upon which a total order $\prec$ is defined. An *atom* α is an expression $p(t_1, \ldots, t_n)$, where p is a predicate of arity $n \geq 0$ and each t_i is either a variable or an element from U (i.e, the resulting language is function-free). An atom is *ground* if it is free of variables. We denote the set of all ground atoms over U by B_U. A *(disjunctive) rule* r is of the form

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

with $n \geq 0$, $m \geq k \geq 0$, $n + m > 0$, where $a_1, \ldots, a_n, b_1, \ldots, b_m$ are atoms, or a count expression of the form $\#count\{l : l_1, \ldots, l_i\} \bowtie u$, where l is an atom and l_j is a literal (i.e, an atom which can be negated or not), $1 \geq j \geq i$, $\bowtie \in \{\leq, <, =, >, \geq\}$, and $u \in \mathbb{N}$. Moreover, "not" denotes *default negation*. The *head* of r is the set $head(r) = \{a_1, \ldots, a_n\}$ and the *body* of r is $body(r) = \{b_1, \ldots, b_k, not\ b_{k+1}, \ldots, not\ b_m\}$. Furthermore, we distinguish between $body^+(r) = \{b_1, \ldots, b_k\}$ and $body^-(r) = \{b_{k+1}, \ldots, b_m\}$. A rule r is *normal* if $n \leq 1$ and a *constraint* if $n = 0$. A rule r is *safe* if each variable in r occurs in $body^+(r)$. A rule r is *ground* if no variable occurs in r. A *fact* is a ground rule with $body(r) = \emptyset$ and $|head(r)| = 1$. An *(input) database* is a set of facts. A *program* is a finite set of rules. For a program Π and an input database D, we often write $\Pi(D)$ instead of $D \cup \Pi$. If each rule in a program is normal (resp. ground), we call the program normal (resp. ground).

Given a program Π, let U_Π be the set of all constants appearing in Π. $Gr(\Pi)$ is the set of rules $r\sigma$ obtained by applying, to each rule $r \in \Pi$, all possible substitutions σ from the variables in r to elements of U_Π. For count-expressions, $\{l : l_1, \ldots, l_n\}$ denotes the set of all ground instantiations of l, governed through $l_1, \ldots, l_n$. An interpretation $I \subseteq B_U$ satisfies a ground rule r iff $head(r) \cap I \neq \emptyset$ whenever $body^+(r) \subseteq I$, $body^-(r) \cap I = \emptyset$, and for each contained count-expression, $N \bowtie u$ holds, where $N = |\{l | l_1, \ldots, l_n\}|$, $u \in \mathbb{N}$ and $\bowtie \in \{\leq, <, =, >, \geq\}$. A ground program Π is satisfied by I, if I satisfies each $r \in \Pi$. A non-ground rule r (resp., a program Π) is satisfied by an interpretation I iff I satisfies all groundings of r (resp., $Gr(\Pi)$). A subset-minimal set $I \subseteq B_U$ satisfying the *Gelfond-Lifschitz reduct* $\Pi^I = \{head(r) \leftarrow body^+(r) | I \cap body^-(r) = \emptyset, r \in Gr(\Pi)\}$ is called an *answer set* of Π. We denote the set of answer sets for a program Π by $AS(\Pi)$.

The tools used in this work are part of the Potassco[2] collection [8]. The main tool of the collection is the *clingo* ASP solver [9].

2.2. Sequential Pattern Mining

Our terminology on sequence mining follows the one in [10]. Throughout this article, $[n] = \{1, \ldots, n\}$ denotes the set of the first n positive integers.

[2] https://potassco.org/

Table 1
An example of sequence database $\mathcal{D}$

Id	Sequence
1	$\langle d\ a\ b\ c\rangle$
2	$\langle a\ c\ b\ c\rangle$
3	$\langle a\ b\ c\rangle$
4	$\langle a\ b\, c\rangle$
5	$\langle a\ c\rangle$
6	$\langle b\rangle$
7	$\langle c\rangle$

Let Σ be the alphabet, i.e, the set of items. An *itemset* $A = \{a_1, a_2, \ldots, a_m\} \subseteq \Sigma$ is a finite set of items. The size of A, denoted $|A|$, is m. A *sequence* s is of the form $s = \langle s_1 s_2 \ldots s_n\rangle$ 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 $s = \langle s_1 \ldots s_m\rangle$ with $s_i \in \Sigma$ is contained in a sequence $t = \langle t_1 \ldots t_n\rangle$ 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 $(e_1 \ldots e_m)$ 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 p is the set of sequences in $\mathcal{D}$ that contain p: $cover(p, \mathcal{D}) = \{t \in D | p \sqsubseteq t\}$. The number of sequences in $\mathcal{D}$ containing p is called its *support*, that is, $supp(p, \mathcal{D}) = |cover(p, \mathcal{D})|$. For an integer th, the problem of *frequent sequence mining* is about discovering all sequences p such that $supp(p, \mathcal{D}) \geq th$. We often call p a (sequential) pattern, and th is also referred to as the (minimum) *support threshold*. For $th = 2$ 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.

The ASP encoding for sequential pattern mining considered in this paper follows the principles outlined in [11] and [4]. In particular, $\mathcal{D}$ is represented as a collection of ASP facts *seq(t, p, e)*, where the *seq* predicate says that an item *e* occurs at position *p* in a sequence *t*. Also, there are two parameters to be defined. Besides *th*, *maxlen* determines the maximum pattern length. The lower the value of *th* 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 frequent 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 object in increasing order, where *m* can vary, while *1* always indicates the first item in the pattern. For example the atoms *pat(1, a)*, *pat(2, b)* and *pat(3, c)* describe a frequent pattern $\langle a\ b\ c\rangle$ of the database in Table 1.

For a thorough discussion of the program the reader can refer to [5].

3. The DF case study

As a case study for our application of sequential pattern mining in DF, we have considered a dataset that has been made available by Prof. David Billard (University of Applied Sciences in Geneva) under NDA to DigForASP members for academic experimentation. In the following we provide details of the structure of the dataset and of the pre-processing which is necessary in order to put the dataset into a more suitable format for the application in hand.

3.1. The DigForASP dataset

The dataset consists of the telephone records of four users from a real-word investigative case. Each file in the dataset has the following schema:

- *Type*: what kind of operation the user has performed (e.g, incoming/outgoing call or SMS);
- *Caller*: who makes the call or sends an SMS;
- *Callee*: who receives the call or SMS;
- *Street*: where the operation has taken place;
- *Time*: when the operation has taken place (ISO format[3] HH: MM: SS);
- *Duration*: how long the operation has been (ISO format HH: MM: SS);
- *Date*: when the operation has taken place (format: day, month, year).

The type of the operation is one of the following cases: "config", "gprs", "redirect", "out_sms(SUB_TYPE)", "in_sms(SUB_TYPE)", "out_call(SUB_TYPE)", "in_call(SUB_TYPE)". Subtypes are: "simple", "ack", "foreign".

The dataset has undergone the mandatory anonymization process for reasons of privacy and confidentiality. Therefore it does not contain data that allows tracing back to the real people involved in the investigative case. For instance, there is no phone number for the caller/callee but only a fictitious name. The names and the sizes (# rows) of the four files in the dataset are the following: Eudokia Makrembolitissa (8,783), Karen Cook McNally (20,894), Laila Lalami (12,689), and Lucy Delaney (8,480).

3.2. Data pre-processing

The DigForASP dataset in its original format can not be considered as a set of sequences. Therefore, it must undergo an intermediate transformation. In short, each line of the original dataset is transformed into an ASP fact through the *seq_event* atom.

The procedure for transforming the original dataset into sequences of ASP facts is the following. Each row of the dataset has been transformed into a fact *seq_event(t, p, e)* (Listing 1), where e represents the item (in our case the event), p defines the position of e within the sequence t (identified by date). The term p is important as it allows you to define the order of events within a sequence. More specifically, e is made up of the following features: *Type*, *Caller*, *Callee*, *Street_a* and *Street_b* for the geo-location of the event, the *(hour, minute, seconds)*

[3]Format to describe dates and times: https://en.wikipedia.org/wiki/ISO_8601

triple, *Weekday* with range (0 = Monday, ..., 6 = Sunday), and *Duration* expressed in seconds. Notice that, with reference to the first two facts in Listing 1, the event e_1 is prior to e_2 since $(p_{e_1} = 1) < (p_{e_2} = 2)$. Also, it is possible to transform in sequence only certain days, months or years, so that the analysis can be carried out at different granularity levels.

```
seq_event((1,9,2040),1,(out_call(simple),eudokia_makrembolitissa,florence_violet_mckenzie,
    acheson_boulevard,acheson_boulevard,(0,12,9),5,10)).
seq_event((1,9,2040),2,(out_call(simple),eudokia_makrembolitissa,florence_violet_mckenzie,
    acheson_boulevard,ashcott_street,(0,12,50),5,39)).
.
.
seq_event((2,9,2040),1,(in_sms(simple),annie_dillard,eudokia_makrembolitissa,alder_road,
    none,(9,22,26),6,0)).
seq_event((2,9,2040),2,(out_call(simple),eudokia_makrembolitissa,irena_jordanova,
    alexander_muir_road,adenmore_road,(11,55,29),6,82)).
.
.
```

Listing 1: Some facts representing sequences of events in the DigForASP dataset.

Additional pre-processing is required to create simpler and easier to analyze sequences out of the *seq_event* atoms in Listing 1. The idea is to create sequences whose identifier refers to a particular day describing what events on that day happened. For instance, in *communication sequences* the event *e* refers to the *(Caller, Callee)* pair as shown in Listing 2.

```
seq((1,9,2040),1,(eudokia_makrembolitissa,florence_violet_mckenzie)).
seq((1,9,2040),2,(eudokia_makrembolitissa,florence_violet_mckenzie)).
.
.
seq((2,9,2040),1,(annie_dillard,eudokia_makrembolitissa)).
seq((2,9,2040),2,(eudokia_makrembolitissa,irena_jordanova)).
.
.
```

Listing 2: Communication sequences generated from the facts in Listing 1.

4. Our ASP-based Approach

For the purposes of law enforcement investigations, it is especially useful to understand what the extracted patterns are and what information they provide to the analyst. To this aim, we have modified the basic algorithm provided by [5] in such a way as to elaborate patterns whose items have a more complex structure, more precisely the structure of the DigForASP dataset (see Section 3.2). In Section 4.1, for the sake of self-containment of this paper, we briefly describe the base ASP encoding already presented in [6]. Then we introduce the ASP encoding for mining so-called *condensed representations* (closed and maximal patterns) in Section 4.2, and finally report the results of a comparative evaluation in Section 4.3.

4.1. Mining Communication Sequences with ASP

Listing 3 reports the base version of our adaptation of [5] to the problem first introduced in [6] of mining communication sequences. Here, we need to handle items with an internal structure (Lines 1, 11, and 13). Also, it is necessary to figure out in which and in how many daily sequences the patterns are found (Line 21). Patterns are then enriched with additional information such as the type of operation (*Type*) carried out between the two communicating entities (*CC*) and the precise time of day (*Time*) with the relative date (*T*) (Line 22). Furthermore, since the dataset contains rows with undefined values (indicated with *none*), two constraints have been added to eliminate all patterns with *none* (Lines 25-26). Finally, in addition to the *maxlen* parameter, often used in pattern mining, the *minlen* parameter has been introduced with relative constraint to filter out patterns not sufficiently long (Line 29).

```
1  item(I) :- seq(_, _,(I, _, _)).
2
3  % sequential pattern generation
4  patpos(1).
5  { patpos(X+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 % pattern embeddings
11 occ(T,1,P) :- seq(T,P,(I, _, _)), pat(1,I).
12 occ(T,L,P) :- occ(T, L,   P-1), seq(T,P,_).
13 occ(T,L,P) :- occ(T, L-1, P-1), seq(T,P,(C, _, _)), pat(L,C).
14
15 % frequency constraint
16 seqlen(T,L) :- seq(T,L,_), not seq(T,L+1,_).
17 supp(T) :- occ(T, L, LS), patlen(L), seqlen(T,LS).
18 :- { supp(T) } < th.
19
20 % pattern information
21 len_support(N) :- N = #count{T : supp(T)}.
22 pat_information(T, (Pos, CC) , Type, Time) :- supp(T), pat(Pos, CC), seq(T, P, (CC, Type,
       Time)), occ(T, Pos, P).
23
24 % constraint for specific db with none line
25 :- pat(_, (none, _)).
26 :- pat(_, (_, none)).
27
28 % constraint for minimum pattern length
29 :- #count{T : pat(T, _)} < minlen.
30
31 % atoms to print
32 #show pat/2.
33 #show len_support/1.
34 #show support/1.
35 #show pat_information/4.
```

Listing 3: Base ASP encoding for mining communication sequences [6].

Each answer set returned by Listing 3 is a sequential pattern represented by means of the $pat/2$ predicate. As an example, Listing 4 reports the first answer out of the 15 generated by Listing 3 for a run over 100 instances from the DigForASP dataset, with maximum pattern length equal to 3 and minimum support threshold equal to 25%. It represents the sequential pattern which consists of two communication events: The first is between Margaret Hasse and Karen Cook McNally (Line 2), while the second is between Karen Cook McNally and Lucie Julia (Line 3). The pattern occurs in the days 8, 9 and 12 of September 2040 (see Line 4), and can be graphically presented as shown in Figure 1.

```
1  Answer: 1
2  pat(1,(margaret_hasse,karen_cook_mcnally))
3  pat(2,(karen_cook_mcnally,lucie_julia))
4  support((8,9,2040)) support((9,9,2040)) support((12,9,2040))
5  pat_information((8,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_sms(simple),(1,0,55))
6  pat_information((8,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_sms(simple),(1,2,27))
7  pat_information((8,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_sms(simple),(8,55,9))
8  pat_information((8,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_sms(simple),(8,55,16))
9  pat_information((9,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_sms(simple),(1,33,29))
10 pat_information((9,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_call(simple),(10,24,9))
11 pat_information((12,9,2040),(1,(margaret_hasse,karen_cook_mcnally)),in_call(simple),(8,23,41))
12 pat_information((12,9,2040),(2,(karen_cook_mcnally,lucie_julia)),out_call(simple),(8,26,17))
13 len_support(3)
```

Listing 4: An example of answer returned by Listing 3.

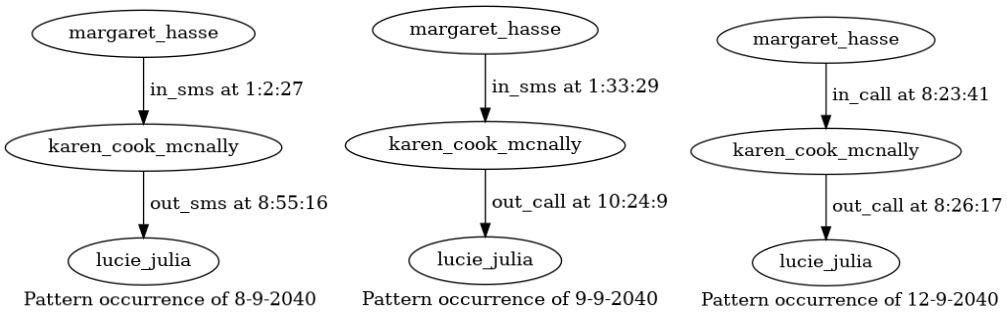

Figure 1: The three occurrences of the sequential pattern corresponding to Answer 1 in Listing 4.

4.2. Condensed representations for sequential pattern mining

One of the major problems in pattern mining is still the problem of pattern explosion, i.e., the large amounts of patterns produced by the mining algorithms when analyzing a database with a predefined minimum support threshold. To address this issue, condensed representations for patterns have been proposed in the literature **[rif?]**.

In this Section we will discuss how additional requirements can be encoded with ASP in order to extract maximal and closed sequential patterns as shown in [5]. Let $\mathcal{D}$ be a sequence database, $supp(s, \mathcal{D})$ the support evaluation function, and th the minimum support threshold. A pattern s is *maximal*, if there are no other patterns t such that $s \subseteq t$ and $supp(s, \mathcal{D}) \geq th$. A pattern s is *closed*, if no other pattern t exists such that $s \subseteq t$ and $supp(s, \mathcal{D}) = supp(t, \mathcal{D})$.

```
1  % embeddings in a reverse order
2  rocc(T,L,P) :- seq(T,P,I), pat(L,I), patlen(L).
3  rocc(T,L,P) :- rocc(T, L, P+1), seq(T,P,_).
4  rocc(T,L,P) :- rocc(T, L+1, P+1), seq(T,P,C), pat(L,C).
5
6  % insertable items
7  ins(T,1 ,I) :- seq(T,P,I), rocc(T,1, P+1).
8  ins(T,L+1,I) :- seq(T,P,I), occ(T,L,P-1), patlen(L).
9  ins(T,X, I) :- seq(T,P,I), rocc(T,X,P+1),
10 occ(T,X-1,P-1), patpos(X), X>1.
11
12 % Integrity constraint for maximal patterns
13 :- item(I), X = 1..maxlen+1, { ins(T,X,I) : support(T) } >= th.
14
15 % Integrity constraint for closed patterns
16 :- item(I), X = 1..maxlen+1, { ins(T,X,I) } >=th,
17 ins(T,X,I) : support(T).
```

Listing 5: ASP encoding for maximal and closed sequential patterns [5].

Listing 5 first describes how to define the set of items that can be inserted between successive items of an embedding (Lines 1-10). These itemsets are encoded by the atoms with predicate *ins(T, X, I)* where *I* is an item which can be inserted in an embedding of the current pattern in sequence *T* between items at position *X* and *X+1* in the pattern. Here, only the positions of the last and the first valid occurrences are required for any pattern item. It can be observed that the strategy provides the first valid occurrence of an item *X* as the first atom of the *occ(T, X, _)* sequence. Then, computing the last occurrence for each pattern item can be done in a similar way by considering an embedding represented in reverse order. Lines 2 to 4 represent *occ/3* and *rocc/3* (reverse order) occurrences.

The computation of insertable items (Lines 7-10) exploits the above remark. Line 7 defines the insertable region in a prefix using *rocc(T, 1, P)*. Since items are insertable if they are strictly before the first position, we consider the value of *rocc(T, 1, P+1)*. Line 8 uses *occ(T, L, P)* to identify the suffix region. Lines 9-10 combine both constraints for in-between cases.

Listing 5 includes also the (integrity) constraints for dealing with closed and maximal patterns. To extract only maximal patterns, the constraint at Line 13 denies patterns for which it is possible to insert an item which will be frequent within sequences that support the current pattern. The constraint at Line 16 concerns the extraction of closed-patterns. It specifies that for each insertion position (from 1, in the prefix, to *maxlen+1*, in the suffix), it is not possible to have a frequent insertable item *I* for each supported transaction.

Though interesting from a theoretical point of view, these encodings lead to more complex programs and should be more difficult to ground and to solve as stressed in [5].

4.3. Experiments

The goal of the experiments discussed in this Section is the comparative evaluation of the three versions of our ASP encoding for mining frequent, closed, and maximal sequential patterns, respectively. The comparison is done with respect to the number of patterns, the execution

time and the memory usage, by varying the key parameters. The experiments can be grouped into two categories:

1. Discovery of frequent, closed and maximal patterns by varying:
    - the minimum support threshold from 10% to 50% while leaving the maximum pattern length fixed to 5;
    - the maximum pattern length (1, 3, 5 and 8) while leaving the minimum support threshold fixed to 25%.
2. Scalability tests:
    - discovery of frequent patterns by varying the dataset size (100, 1K, 10K), while leaving the minimum support threshold unchanged (set to 25%) and the maximum pattern length fixed to 3.

All the experiments have been conducted over the largest available file of the DigForASP dataset (named Karen Cook McNally) made up of more than 20,000 instances. Given the size, a fairly long execution time for our ASP programs has been assumed. Therefore, the timeout has been set to 5 hours. As a solver, we have used the version 5.4.0 of clingo, with default solving parameters. 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.

In Table 2 and Figure 2 it is possible to observe how the use of condensed representations, in particular the maximal one, allows to reduce the number of extracted patterns. Considering the case of frequent patterns, the computation ends before the time limit only for the threshold set to 50%, while for the case of closed and maximal patterns the computation stopped at the time limit set at 5 hours. In the first case, the behavior is due to a more restrictive constraint, which allows to extract all the patterns in the dataset, while with a lower threshold values the patterns are many more. Viceversa, condensed representations require longer times to extract the patterns even at the same threshold values. As regards the memory usage (Table 2 and Figure 3), one can observe that it decreases, as the minimum support threshold decreases, and this holds for all condensed representations. In particular, the extraction of closed patterns uses more memory than the discovery of frequent and maximal patterns.

With a fixed threshold of 25% and the variation in the maximum pattern length (Table 3) we can see a generalized trend in which both the time taken and the memory used increase (Figure 5) as the length of the patterns increases. Notably, the memory usage is higher for closed patterns as already observed in the previous experiments. Except for the case of frequent patterns, the computation needed to extract patterns with a maximum length of 3 reaches the time limit. Once again, maximal patterns drastically reduce the size of the output (Figure 4).

For the scalability tests, values of 25% and 3 were chosen for the minimum support threshold and maximum pattern length, respectively. As can be seen in Table 4, the memory used increases as the size of the dataset increases (Figure 6). The time taken, except for frequent patterns, reaches the computation time limit for condensed representations when the size of the dataset is in the order of tens of thousands of rows (Figure 7). A general behavior concerns the closed

Table 2
Number of frequent (A), closed (B) and maximal (C) patterns generated as the minimum support threshold varies, while leaving the maximum pattern length set to 5.

(A) Frequent

Threshold	# Patterns	Execution Time	Solver Time	Memory usage (MB)
10%	3993	17,999.998s	17,995.37s	698.01
20%	1026	17,999.999s	17,996.94s	652.2
30%	413	17,999.999s	17,996.84s	471.54
40%	77	17,999.999s	17,996.93s	411.93
50%	78	14,819.002s	14,815.91s	306.06

(B) Closed

Threshold	# Patterns	Execution Time	Solver Time	Memory usage (MB)
10%	6691	17,999.998s	17,983.82s	935.66
15%	511	17,999.999s	17,989.97s	814.14
16%	2456	17,999.999s	17,990.00s	804.68
18%	213	17,999.999s	17,989.94s	962.55
20%	524	17,999.999s	17,990.16s	890.42
30%	310	18,000.001s	17,990.39s	767.39
40%	147	17,999.999s	17,989.64s	752.97
50%	11	17,999.999s	17,990.83s	748.29

(C) Maximal

Threshold	# Patterns	Execution Time	Solver Time	Memory usage (MB)
10%	186	17,999.998s	17,989.93s	623.21
15%	148	17,999.999s	17,993.43s	551.35
20%	229	17,999.999s	17,993.61s	568.65
30%	69	17,999.999s	17,993.62s	511.79
40%	42	17,999.999s	17,993.79s	476.52
50%	19	17,999.999s	17,993.72s	469.27

patterns: The used memory is always higher than for the other representations in all the experiments done.

5. Final remarks

Pattern mining problems typically involve a combinatorial explosion. Condensed representations for patterns are among the solutions proposed in the literature to address this issue. In this paper we have reported the results of a comparative evaluation of the ASP encodings for solving three variants of the problem of mining sequential patterns, one being the base version (discovery of frequent patterns) and the other two using condensed representations (discovery of closed and maximal patterns). As a case study we have considered the analysis of a real-world dataset of anonymised phone recordings. The results show that condensed representations have pros and cons. On one hand, they effectively reduce the number of patterns. On the other hand, they tend to consume more computational resources.

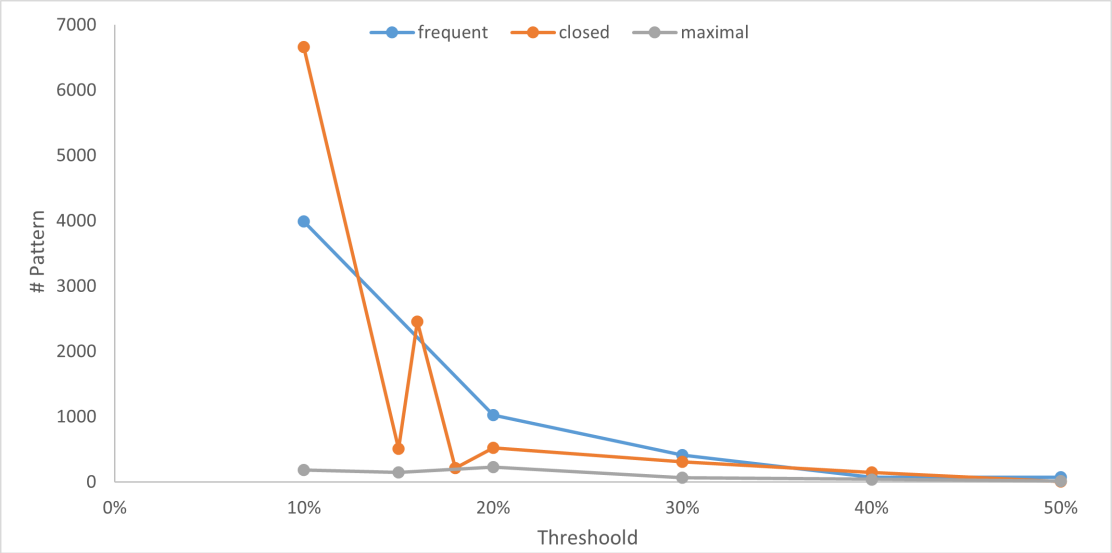

Figure 2: Comparison as regards the number of patterns while varying the minimum support threshold (Table 2).

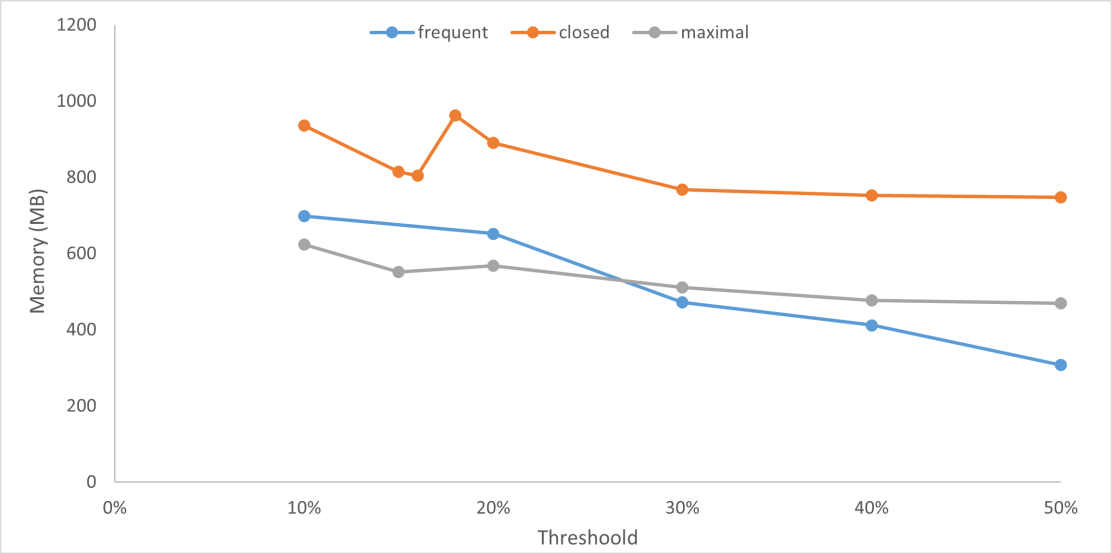

Figure 3: Comparison as regards memory usage while varying the minimum support threshold (Table 2).

For the future we intend to consider the case of localization events, so that evidence can be analysed from a spatio-temporal viewpoint. Furthermore, we expect that the solicited feedback from the DF experts involved in DigForASP will not only validate our work but will also suggest new interesting directions of research.

Table 3
Number of frequent (A), closed (B) and maximal (C) sequential patterns generated as the maximum pattern length varies from 1 to 8, while leaving the minimum support threshold fixed to 25%.

(A) Frequent

Max. length	# Patterns	Execution Time	Solver Time	Memory usage (MB)
1	47	6.127s	4.18s	95.1
3	769	10,562.276s	10,560.14s	306.89
5	200	17,999.999s	17,996.69s	294.23
8	1657	17,999.999s	17,989.19s	533.26

(B) Closed

Max. length	# Patterns	Execution Time	Solver Time	Memory usage (MB)
1	47	9.522s	5.52s	196.74
3	626	17,999.999s	17,994.01s	572.32
5	449	17,999.999s	17,990.46s	916.69
8	1474	17,999.999s	17,984.96s	1157.45

(C) Maximal

Max. length	# Patterns	Execution Time	Solver Time	Memory usage (MB)
1	15	75.903s	73.26s	115.03
3	309	17,999.999s	17,994.35s	394.69
5	207	17,999.999s	17,993.78s	613.96
8	141	17,999.999s	17,990.09s	764.98

Acknowledgments

This article is based upon work from COST Action 17124 “Digital forensics: evidence analysis via intelligent systems and practices (DigForASP)”, supported by COST (European Cooperation in Science and Technology). The work is also partially funded by the University of Bari “Aldo Moro” under the 2017-2018 grant “Metodi di Intelligenza Artificiale per l’Informatica Forense”.

References

[1] S. Costantini, G. De Gasperis, R. Olivieri, How answer set programming can help in digital forensic investigation, in: D. Ancona, M. Maratea, V. Mascardi (Eds.), Proceedings of the 30th Italian Conference on Computational Logic, Genova, Italy, July 1-3, 2015, volume 1459 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2015, pp. 53–65. URL: http://ceur-ws.org/Vol-1459/paper29.pdf.

[2] S. Costantini, G. De Gasperis, R. Olivieri, Digital forensics and investigations meet artificial intelligence, Ann. Math. Artif. Intell. 86 (2019) 193–229. URL: https://doi.org/10.1007/s10472-019-09632-y. doi:10.1007/s10472-019-09632-y.

[3] S. Costantini, F. A. Lisi, R. Olivieri, DigForASP: A European cooperation network for logic-based AI in digital forensics, in: A. Casagrande, E. G. Omodeo (Eds.), Proceedings of the 34th Italian Conference on Computational Logic, Trieste, Italy, June 19-21, 2019,

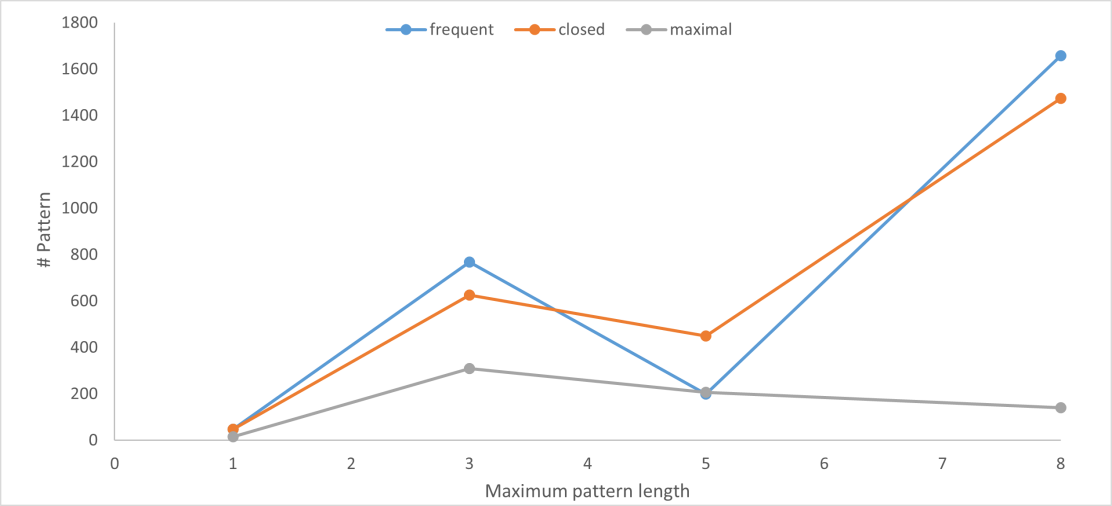

Figure 4: Comparison as regards the number of patterns by varying the maximum pattern length (Table 3).

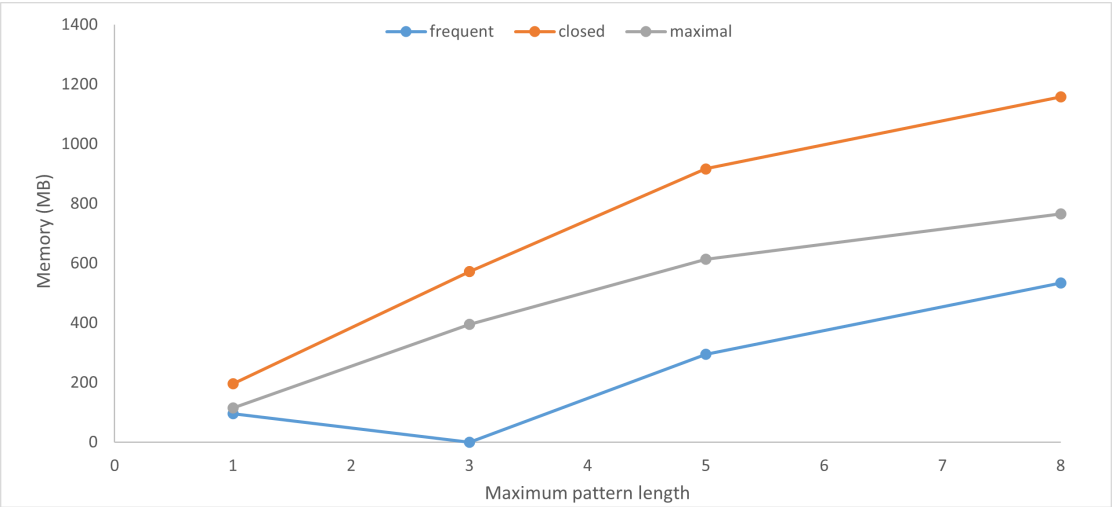

Figure 5: Comparison as regards memory usage by varying the maximum pattern length (Table 3).

volume 2396 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2019, pp. 138–146. URL: http://ceur-ws.org/Vol-2396/paper34.pdf.

[4] M. Gebser, T. Guyet, R. Quiniou, J. Romero, T. Schaub, Knowledge-based sequence mining with asp, in: IJCAI 2016-25th International joint conference on artificial intelligence, AAAI, 2016, p. 8.

[5] T. Guyet, Y. Moinard, R. Quiniou, T. Schaub, Efficiency analysis of asp encodings for sequential pattern mining tasks, in: Advances in Knowledge Discovery and Management, Springer, 2018, pp. 41–81.

[6] F. A. Lisi, G. Sterlicchio, Declarative pattern mining in digital forensics: Preliminary results,

Table 4
Number of frequent (A), closed (B) and maximal (C) sequential patterns generated as the dataset size varies (100, 1K, 10K), while leaving the minimum support threshold (25%) and maximum pattern length (3) unchanged.

(A) Frequent

# Rows	# Patterns	Execution Time	Solver Time	Memory usage (MB)	# Sequences
100	15	0.084ss	0.03s	14.52	6
1000	9821	81.993s	81.83s	72.81	9
10000	51	2978.229s	2976.73s	170.2	93

(B) Closed

# Rows	# Patterns	Execution Time	Solver Time	Memory usage (MB)	# Sequences
100	7	0.091s	0.03s	15.73	6
1000	171	13.446s	13.22s	36.50	9
10000	625	18,000.012s	17,997.07s	281.72	93

(C) Maximal

# Rows	# Patterns	Execution Time	Solver Time	Memory usage (MB)	# Sequences
100	6	0.099s	0.02s	15.33	6
1000	32	10.928s	10.60s	30.75	9
10000	153	18,000.001s	17,997.71s	265.33	93

in: R. Calegari, G. Ciatto, A. Omicini (Eds.), Proceedings of the 37th Italian Conference on Computational Logic, Bologna, Italy, June 29 - July 1, 2022, volume 3204 of *CEUR Workshop Proceedings*, CEUR-WS.org, 2022, pp. 232–246. URL: http://ceur-ws.org/Vol-3204/paper_23.pdf.

[7] G. Brewka, T. Eiter, M. Truszczynski, Answer set programming at a glance, Communications of the ACM 54 (2011) 92–103. URL: http://doi.acm.org/10.1145/2043174.2043195. doi:10.1145/2043174.2043195.

[8] M. Gebser, B. Kaufmann, R. Kaminski, M. Ostrowski, T. Schaub, M. Schneider, Potassco: The potsdam answer set solving collection, Ai Communications 24 (2011) 107–124.

[9] M. Gebser, R. Kaminski, B. Kaufmann, T. Schaub, Clingo= asp+ control: Preliminary report, arXiv preprint arXiv:1405.3694 (2014).

[10] B. Negrevergne, T. Guns, Constraint-based sequence mining using constraint programming, in: International Conference on Integration of Constraint Programming, Artificial Intelligence, and Operations Research, Springer, 2015, pp. 288–305.

[11] M. Järvisalo, Itemset mining as a challenge application for answer set enumeration, in: International Conference on Logic Programming and Nonmonotonic Reasoning, Springer, 2011, pp. 304–310.

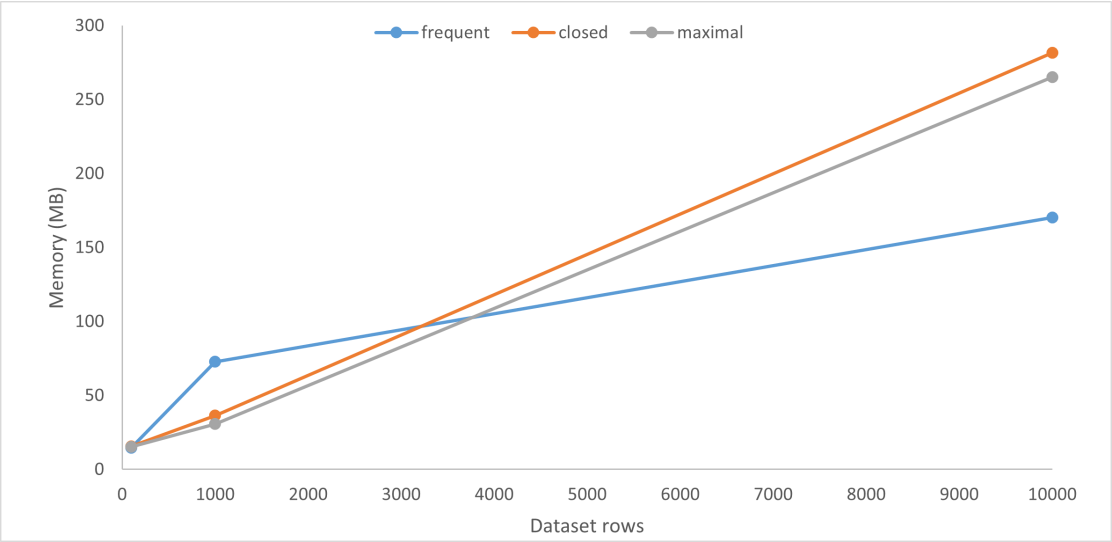

Figure 6: Comparison as regards memory usage at the variation of the dataset size (Table 4).

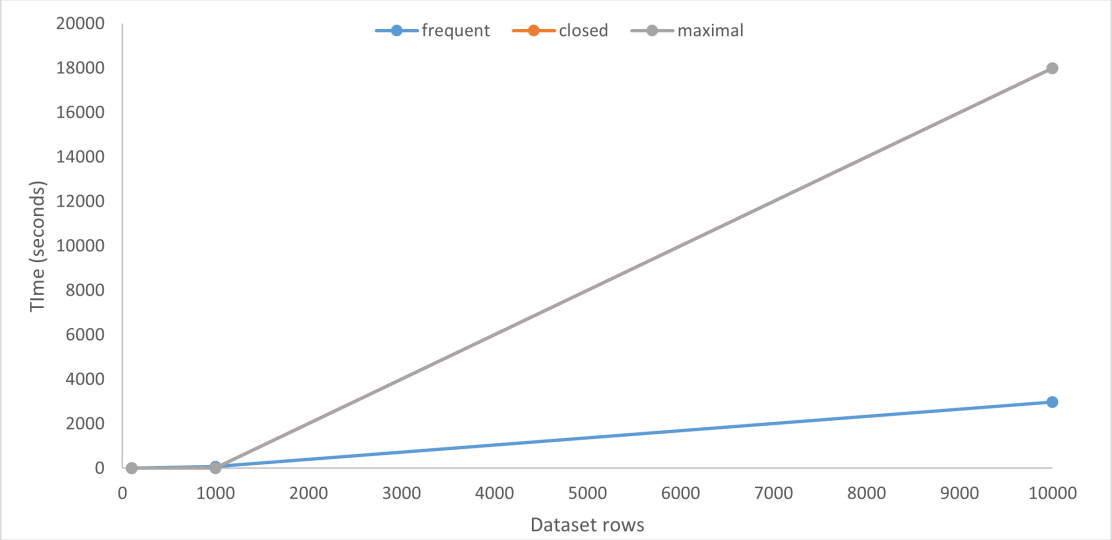

Figure 7: Comparison as regards execution time at the variation of the dataset size (Table 4).