Declarative Pattern Mining in Digital Forensics: Preliminary Results

Francesca Alessandra Lisi[1,*], Gioacchino Sterlicchio[1]

[1]University of Bari "Aldo Moro", Via E. Orabona 4, Bari, 70125, Italy

Abstract

This paper proposes the application of ASP-based sequential pattern mining techniques in the analysis of evidence collected according to the practice of digital forensics. In particular, it reports preliminary results concerning the analysis of anonymised mobile phone recordings, which highlight the sequences of events in a given time span.

Keywords

Sequential Pattern Mining, Answer Set Programming, Digital Forensics

1. Introduction

Digital Forensics (DF) is a branch of criminalistics which deals with the identification, acquisition, preservation, analysis and presentation of the information content of computer systems, or in general of digital devices, by means of specialized software, and according to specific regulations. In particular, the phase of *Evidence Analysis* involves examining and aggregating 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. Evidence Analysis results are made available to law enforcement, investigators, intelligence agencies, public prosecutors, lawyers and judges.

Unlike the phase of Identification, where the application of Machine Learning (ML) techniques can be useful for the analysis of big data, the phase of Evidence Analysis has particular requirements that make the use of techniques from *Knowledge Representation* (KR) and *Automated Reasoning* (AR) a much more promising approach, potentially becoming a breakthrough in the state-of-the-art. 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 ML classifiers or other types of "black box" AI systems do not have more value than human witness' suspicions and cannot be used as legal evidence. Logical 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.

CILC 2022: 37th Italian Conference on Computational Logic, June 29 – July 1, 2022, Bologna, Italy

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

FrancescaAlessandra.Lisi@uniba.it (F. A. Lisi); g.sterlicchio2@studenti.uniba.it (G. Sterlicchio)

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

0000-0001-5414-5844 (F. A. Lisi)

This has the benefit that the correctness of the resulting systems can be formally verified. Moreover, recent research has led to new methods for visualising and explaining the results of computed answers (*e.g.*, based on argumentation schemes). So one can not only represent and solve relevant problems, but also provide tools to explain the conclusions (and their proofs) in a transparent, comprehensible and justified way. This approach to DF was first explored by Costantini *et al.* [1, 2], and subsequently adopted by the COST Action "Digital forensics: evidence analysis via intelligent systems and practices" (DigForASP)[1] which aims at promoting formal and verifiable AI methods and techniques for Evidence Analysis [3].

Pattern mining [4] is a class of data mining tasks that consist of extracting interesting structured patterns from a set of structured examples. These tasks encompass itemset mining, sequence mining and graph mining. The interestingness measure of a pattern is, in most of the algorithms, the number of its occurrences in the set of examples. Given a threshold k, interesting patterns are those that occur at least in k examples. In this case, the task is known as *frequent pattern mining* for which many algorithms have been proposed. Most of the efficient algorithmic solutions rely on an antimonotonicity property of the support: the larger the pattern, the fewer it occurs. Declarative pattern mining (DPM) aims at encoding pattern tasks in a declarative framework, and more specifically the frequent pattern mining tasks. Declarative pattern mining addressed the tasks of frequent itemset mining [5, 6], frequent sequential patterns [7, 8]. Different declarative frameworks have been explored: SAT [5], CP [9, 6], and ASP [8, 10]. 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. In this paper we will consider the case of sequential patterns, which turn out to be promising as a support to the analysis of events and sequences of events in scenarios of interest to DF experts.

The paper is organized as follows. In Section 2 we provide the necessary preliminaries on ASP, sequential pattern mining and the ASP encoding used in our work. In Section 3 we describe the application to a typical DF problem: the analysis of mobile phone recordings. In Section 4 we report some preliminary experimental results. In Section 5 we conclude by commenting the ongoing work and by outlining some promising directions for research.

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.*, [11] 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

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

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 [12]. The main tool of the collection is the *clingo* ASP solver [13].

2.2. Sequential Pattern Mining

Our terminology on sequence mining follows the one in [7]. Throughout this article, $[n] = \{1, \ldots, n\}$ denotes the set of the first n positive integers.

Let Σ be the alphabet, *i.e.*, the set of items. An *itemset* $A = \{a_1, a_2, \ldots, a_m\} \subseteq \Sigma$ 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 $(1, 3)$. (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*,

[2]Potassco: 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	$\langle c\rangle$

that is, $supp(p, \mathcal{D}) = |cover(p, \mathcal{D})|$. For an integer k, the problem of *frequent sequence mining* is about discovering all sequences p such that $supp(p, \mathcal{D}) \geq k$. We often call p a (sequential) pattern, and k is also referred to as the (minimum) *support threshold*. For $k = 2$ we can see how $\langle a\rangle$, $\langle b\rangle$, $\langle c\rangle$, $\langle a\ b\rangle$, $\langle a\ c\rangle$, $\langle b\ c\rangle$ e $\langle a\ b\ c\rangle$ are common patterns in the database $\mathcal{D}$ reported in Table 1.

2.3. Mining sequential patterns with ASP

The sequence database $\mathcal{D}$ is represented in terms of ASP facts *seq(t, p, e)*, where the *seq* predicate says that an item *e* occurs at position *p* in a sequence *t*. For example, Listing 1 represents the seven sequences of Table 1 in ASP format.

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

Listing 1: ASP encoding for the sequence database $\mathcal{D}$ reported in Table 1.

The ASP encoding for sequential pattern mining follows the principles outlined in [14] and [8]. In particular, there are two parameters to be defined: *maxlen* determines the maximum length of the patterns of interest and *th* specifies the minimum support threshold. The lower the value of *th* the more patterns will be extracted; the lower the *maxlen* parameter, the smaller the ground program will be. Therefore the parameters allow a tuning for the program efficiency. Also, 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 $pat(m, s_1), ..., pat(1, s_m)$. 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.

```
1  item(I) :- seq(_, _,I).
2
3  %sequential pattern generation
4  patpos(1).
5  0 { patpos(Ip+1) } 1 :- patpos(Ip), Ip<maxlen.
6  patlen(L) :- patpos(L), not patpos(L+1).
7
8  1 { pat(Ip,I): item(I) } 1 :- patpos(Ip).
9
10 %pattern embeddings
11 occ(T,1,Is) :- seq(T,Is,I), pat(1,I).
12 occ(T,Ip,Is) :- occ(T, Ip, Is-1), seq(T,Is,_).
13 occ(T,Ip,Is) :- occ(T, Ip-1, Is-1), seq(T,Is,I), pat(L,I).
14
15 %frequency constraint
16 seqlen(T,L) :- seq(T,L,_), not seq(T,L+1,_).
17 support(T) :- occF(T, L, LS), patlen(L), seqlen(T,LS).
18 :- { support(T) } < th.
```

Listing 2: Basic ASP encoding for sequential pattern mining [10].

Listing 2 reports the ASP program for sequential pattern mining according to [10]. Line 1 defines a new predicate that provides all items from the database. Lines 4 to 8 of the program encode the pattern generation. Lines 11 to 13 encode pattern embedding search. Finally, Lines 16 to 18 are dedicated to assess the pattern frequency constraint. For a thorough discussion of the program the reader can refer to [10].

3. Sequence Mining in Mobile Phone Records with ASP

During the investigation of a crime, it is common to analyze the communications of a particular suspect. Given that, nowadays the mobile phone or smartphone is an object owned by anyone, it can be useful for investigators to analyze the calls or messages exchanged. The telephone records are a set of data inserted in tables that contain the data relating to the external communications of the devices. In other words, the telephone records contain all the traces of communications relating to a specific user over a certain period of time, therefore they contain traces of telephone calls, SMS, and all the data traffic of the mobile phone.

Telephone records concern various pieces of information, such as:

- the telephone number of the caller
- the telephone number of the recipient
- the type of communication, *e.g.*: call, sms, missed call;
- the duration of the communication, indicated in minutes and seconds in the event of a call. On the other hand, in the case of text messages or missed calls, this value will be equal to 0 seconds.

In addition to the information indicated above, telephone records can report a series of additional information usually referred to mobile users and therefore related to communications via mobile phones.

Through a telephone records it is not possible to trace a series of important data such as the audio of calls sent or received, the list of SMS messages, the content of the e-mails received or sent, and the list of the web sites visited. In fact, through a telephone records it is possible to have a trace of the communication that has taken place but not to obtain its content. The telephone records can be requested by the Judicial Authority if it deems it useful to get hold of them in order to carry out investigations on the individual owner of the user.

Correctly analyzing the telephone records is essential to obtain useful data. Depending on the analysis, different types of information can be extracted. As a rule, the records are analyzed for comparing the geographical positions with respect to the declarations, and for reconstructing the network of contacts with reference to a single user in order to trace which conversations he/she has had with which people and at what times.

3.1. The DigForASP dataset

For our experiments we have considered a dataset which consists of the telephone records of four users from a real-word investigative case. The dataset has been made available by Prof. David Billard (University of Applied Sciences in Geneva) under NDA to DigForASP members for academic experimentation.

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). An excerpt of the file containing the phone recordings of Eudokia Makrembolitissa is reported in Figure 1.

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

Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Alder Road		07:58:33.000	00:00:00.000	2040-12-29
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Alexander Muir Road		20:00:51.000	00:00:00.000	2041-01-01
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Alhart Drive		22:04:29.000	00:00:00.000	2041-01-02
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Assiniboine Road		19:11:43.000	00:00:00.000	2041-01-05
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Assiniboine Road		12:52:13.000	00:00:00.000	2041-01-06
Incoming SMS	Andrea Levy	Eudokia Makrembolitissa	Assiniboine Road		13:02:11.000	00:00:00.000	2041-01-06
Incoming SMS (/	Andrea Levy	Eudokia Makrembolitissa	Athletic Avenue		12:57:29.000	00:00:00.000	2041-01-06
Incoming call	Andrea Levy	Eudokia Makrembolitissa	3420 St Clair Avenue East	3420 St Clair Aven	14:35:05.000	00:00:25.000	2040-12-28
Incoming call	Andrea Levy	Eudokia Makrembolitissa	Amarillo Drive	Amarillo Drive	12:12:34.000	00:00:20.000	2041-01-01
Incoming call	Angela Rawlings	Eudokia Makrembolitissa	Alder Road	Alder Road	23:01:30.000	00:00:02.000	2041-01-11
Incoming SMS	Angela Topping	Eudokia Makrembolitissa	Abilene Drive		22:13:08.000	00:00:00.000	2041-01-18
Incoming call	Anita Brookner	Eudokia Makrembolitissa	21st Street	31st Street	20:49:30.000	00:00:10.000	2040-12-18
Incoming call	Anita Brookner	Eudokia Makrembolitissa	Abbottswood Road	Abbotsfield Gate L	19:34:40.000	00:00:48.000	2040-12-18
Incoming SMS	Ann Kiessling	Eudokia Makrembolitissa	Alexander Muir Road		10:20:44.000	00:00:00.000	2040-12-07
Incoming call	Ann Taylor	Eudokia Makrembolitissa	Alanbury Crescent	Alanbury Crescent	14:30:26.000	00:00:13.000	2041-02-11
Incoming SMS	Anna Bijns	Eudokia Makrembolitissa	Alcorn Avenue		12:57:32.000	00:00:00.000	2041-02-05
Incoming SMS	Anna Eliza Bray	Eudokia Makrembolitissa	Alder Road		19:48:47.000	00:00:00.000	2040-10-21
Incoming SMS	Anna Eliza Bray	Eudokia Makrembolitissa	Aldergrove Avenue		15:46:00.000	00:00:00.000	2040-10-20
Incoming call	Anna Eliza Bray	Eudokia Makrembolitissa	Alexander Muir Road	Alder Road	15:12:01.000	00:00:19.000	2040-10-20
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Addison Crescent		23:46:45.000	00:00:00.000	2040-11-01
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Adelaide Street East		23:37:16.000	00:00:00.000	2040-11-01
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Advance Road		23:42:33.000	00:00:00.000	2040-11-01
Incoming SMS	Anna Zahorska	Eudokia Makrembolitissa	Alder Road		11:31:11.000	00:00:00.000	2040-11-04
Incoming SMS	Anne Askew	Eudokia Makrembolitissa	Alder Road		15:11:59.000	00:00:00.000	2040-11-02
Incoming SMS	Anne Askew	Eudokia Makrembolitissa	Alder Road		22:56:36.000	00:00:00.000	2040-11-18
Incoming SMS	Anne Bishop	Eudokia Makrembolitissa	Alder Road		21:52:27.000	00:00:00.000	2040-09-05
Incoming SMS (/	Anne Bradstreet	Eudokia Makrembolitissa	Assiniboine Road		16:59:47.000	00:00:00.000	2041-01-06
Incoming SMS	Anne de Marquets	Eudokia Makrembolitissa	Alder Road		10:01:47.000	00:00:00.000	2040-09-21
Incoming SMS	Anne Elliot	Eudokia Makrembolitissa	Adler Street		14:49:41.000	00:00:00.000	2040-10-19
Incoming SMS	Anne Hébert	Eudokia Makrembolitissa	27 S Eglinton E Ramp		23:02:31.000	00:00:00.000	2041-02-01
Incoming SMS	Anne Hébert	Eudokia Makrembolitissa	Alameda Avenue		22:33:46.000	00:00:00.000	2041-01-20

Figure 1: Some rows of the DigForASP dataset. The columns reflect the schema (type, caller, callee, street_a, street_b, time, duration, date).

3.2. Data pre-processing

The dataset can not be used as is to mine sequences with ASP. So, data is pre-processed to lead the dataset to a suitable ASP encoding.

As described in Section 2.2, the problem of sequential pattern mining consists in finding frequent and non-empty sequences s, called sequential patterns, from a database of sequences $\mathcal{D}$. The dataset of interest is the one described in Section 3.1. Obviously, in its original state it cannot be considered as a set of sequences but must undergo an intermediate transformation that leads it to be like the databases described in Section 2.3. In short, each line of the original dataset will be 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 3), 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*: type of event ("in_sms", "redirect", "out_call", etc.);
- *Caller*: the name of the caller;
- *Callee*: the name of the callee;
- *Street_a*: the geo-location of the event;
- *Street_b*: the geo-location of the event;
- the *(hour, minute, seconds)* triple: indicates the moment in time when the event occurred;
- *Weekday*: the day of the week (0 = Monday, ..., 6 = Sunday);
- *Duration*: duration, expressed in seconds, of the operation described by *Type*.

Depending on the analyst's needs, it is also possible to transform in sequence only certain days, months or years so as to subsequently carry out a more granular analysis.

```
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((1,9,2040),3,(in_sms(simple),florence_violet_mckenzie,eudokia_makrembolitissa,
   acheson_boulevard,ashby_place,(1,12,8),5,0)).
.
.
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)).
seq_event((2,9,2040),3,(out_call(simple),eudokia_makrembolitissa,irena_jordanova,
   alder_road,abigail_place,(12,17,57),6,39)).
.
.
```

Listing 3: Some facts representing sequences of events in the dataset.

Notice that, with reference to the first two facts in Listing 3, the event e_1 is prior to e_2 since $(p_{e_1} = 1) < (p_{e_2} = 2)$.

The *seq_event* atoms in Listing 3, in this form, are useless for discovering recurring patterns without first making a more granular choice of which patterns to look for. Additional pre-processing is required to create simpler and easier to analyze sequences. The idea is to create sequences whose identifier refers to a particular day describing what events on that day happened. Two types of sequences have been identified:

Communication sequences The event *e* refers to the *(Caller, Callee)* pair.

Localization sequences The event *e* refers to the *(Street_a, Street_b)* pair.

Listing 4 creates sequences of events in the format shown in Listing 5). The input to this script is sequences like the ones shown in 3. Line 6 allows the creation of sequences via the *seq* predicate. Line 8 contains a rule for calculating the number of sequences, whereas Line 10 generates *len_sequence* facts which denote for each sequence its length, *i.e.*, the number of events. The rule at Line 11 calculates the average length of all sequences, which is the average number of events for each sequence. The rule at Line 13 calculates the sequence with the greatest number of events. Finally, at Lines 15-18, the atoms describing the previously mentioned statistics are shown on an output standard (*e.g.*, terminal, screen) respectively.

```
1  % from
2  % seq_event(Date, Seq_position, (Type_op, Caller, Callee, Street_a, Street_b, Time,
     Weekday, Duration))
3  % to
4  % seq(Date, Seq_position, (Caller, Callee))
5
6  seq(Date, Seq_position, (Caller, Callee)) :- seq_event(Date, Seq_position, (_, Caller,
     Callee, _, _, _, _, _)).
7
```

```
number_of_sequences(N) :- N = #count{D : seq(D, _, _)}.

len_sequence(D, L) :- L = #max{P : seq(D, P, _), seq(D1, _, _), D != D1}, seq(D, _, _).
avg_len_sequences(A) :- S = #sum{L, D : len_sequence(D, L)}, number_of_sequences(N), A =
  S/N.

max_len_sequences(D, N) :- N = #max{L : len_sequence(_, L)}, len_sequence(D, N).

#show number_of_sequences/1.
#show max_len_sequences/2.
#show avg_len_sequences/1.
#show seq/3.
```

Listing 4: Generation of communication sequences with ASP.

```
avg_len_sequences(53).
number_of_sequences(164).
max_len_sequences((1,2,2041),129).
seq((1,9,2040),1,(eudokia_makrembolitissa,florence_violet_mckenzie)).
seq((1,9,2040),2,(eudokia_makrembolitissa,florence_violet_mckenzie)).
seq((1,9,2040),3,(florence_violet_mckenzie,eudokia_makrembolitissa)).
.
.
seq((2,9,2040),1,(annie_dillard,eudokia_makrembolitissa)).
seq((2,9,2040),2,(eudokia_makrembolitissa,irena_jordanova)).
seq((2,9,2040),3,(eudokia_makrembolitissa,irena_jordanova)).
.
.
```

Listing 5: Output generated by Listing 4 from the facts reported in Listing 3.

A similar transformation is needed in order to create localization sequences as shown in Listing 6.

```
avg_len_sequences(53).
number_of_sequences(164).
max_len_sequences((1,2,2041),129).
seq((1,9,2040),1,(acheson_boulevard,acheson_boulevard)).
seq((1,9,2040),2,(acheson_boulevard,ashcott_street)).
seq((1,9,2040),3,(acheson_boulevard,ashby_place)).
.
.
seq((2,9,2040),1,(alder_road,none)).
seq((2,9,2040),2,(alexander_muir_road,adenmore_road)).
seq((2,9,2040),3,(alder_road,abigail_place)).
.
.
```

Listing 6: Output generated by Listing 3.2 from the facts reported in Listing 3.

This can be done by replacing the rule in line 6 of Listing 4 with the following:

```
seq(Date, Seq_position, (Street_a, Street_b)) :- seq_event(Date, Seq_position, (_,_,_,
  Street_a,Street_b,_,_,_)).
```

3.3. Our ASP Encoding for the Analysis of Mobile Phone Records

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, the basic algorithm provided by [10] was modified in such a way as to elaborate patterns whose items have a more complex structure including elements such as caller, callee, type of operation, and time when this occurred (see Section 3.2).

Listing 7 reports the adapted basic algorithm to handle items with an internal structure (Line 1). Consequently, all the rules that managed the embedding were modified to manage a complex item (Lines 11 and 13). For investigative purposes it is necessary to understand in which and how many daily sequences the patterns were found (Lines 21 and 34). Lines 22 and 35, on the other hand, allows you to associate each pattern found with information such as: type of operation (*Type*) carried out between the two communicating entities (*CC*) and the precise time of day (*Time*) with the relative date (*T*). Furthermore, since the dataset contains rows with undefined values (indicated with none), two constraints have been added to eliminate all patterns with a value of none (Lines 25 and 26). A further modification concerns the possibility of being able to search for patterns between a certain minimum and maximum length. To do this, in addition to the *maxlen* parameter, already present, the *minlen* parameter with relative constraint has been added (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 pattern of minimum lenght
29 :- #count{T : pat(T, _)} < minlen.
30
31 % atom to print
32 #show pat/2.
33 #show len_support/1.
34 #show support/1.
35 #show pat_information/4.
```

Listing 7: Modified ASP encoding for sequential pattern mining.

Each answer set returned by the ASP encoding in Listing 7 is a sequential pattern represented by means of the $pat/2$ predicate. The answer includes addition information which is deemed useful for investigation in forensic practice such as: the days in which that pattern was found (see predicate $support/1$), the type of operation carried out between caller and callee and the precise time of the day (see predicate $pat\_information/4$), and the support of that pattern (predicate $len\_support/1$). The support can be useful to understand if the pattern is of interest or a fact given that it occurs every day.

```
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 8: First of the 15 answers generated by Listing 7.

As an example, the first answer out of the 15 returned by the ASP encoding in Listing 7

is reported in Listing 8. It refers to the running over 100 instances, with maximum pattern length equal to 3 and minimum support threshold equal to 25%. Here, Answer 1 highlights the existence of a sequential pattern which consists of a first communication event between Margaret Hasse and Karen Cook McNally followed by the one between Karen Cook McNally and Lucie Julia (see Lines 2 and 3). The pattern occurs in the days 8, 9 and 12 of September 2040, as shown at Line 4. The fact $len\_support(3)$ provides numerical information about the pattern support, in this case 3. Looking at the facts $pat\_information/4$ concerning the date 08/09/2040 (see Lines 5 and 7), we get to know that Karen (the subject of the phone records) received a text message from Margaret at 01:00:55 and then Karen sent a text message to Lucie at 08:55:09.

4. Experiments

The goal of the following experiments is to evaluate the number of patterns discovered by varying the key parameters. For the first group, the minimum support threshold varies from 10% to 50% while keeping the maximum pattern length fixed at 5. For the second group, the maximum pattern length varies over (3, 5 and 8) while keeping the minimum support threshold fixed to 25%.

All the experiments were 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 the ASP program is assumed, Therefore, the timeout has been set to 5 hours.

In all presented experiments, we 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.

Table 2
Number of frequent patterns extracted by varying the minimum support threshold (from 10% to 50%) while keeping the maximum pattern length fixed to 5.

Min. Supp. Threshold	# Patterns	Execution time
10%	5,135	18000s (5h)
20%	1,004	18000s (5h)
30%	730	18000s (5h)
40%	55	18000s (5h)
50%	78	18000s (5h)

Table 2 summarizes the results from the first group of experiments. One can observe the decrease in the number of patterns as the minimum support threshold increases. It is interesting to observe the time taken for the computation: in all cases the computation reached 5 hours

and stopped. This means that the program did not finish the computation but was interrupted by the set time-out. The reason is to be attributed to the size of the analyzed dataset. Given the nature of ASP (*generate&test* paradigm), the high number of combinations contributed to the long time taken to extract the patterns. Finally the patterns extracted are not all the possible ones but they are only those extracted within 5 hours. There may be others or not by continuing to analyze the search space.

Table 3 summarizes the results from the second group of experiments. Here, it is evident the increase in the number of patterns as the maximum pattern length increases. It is interesting to observe that, as the maximum pattern length increases, the time taken for computation increases as well. Only with a maximum pattern length of 3 the computation ended, whereas for length 5 and 8 the computation was interrupted as soon as the time-out was reached.

Table 3
Number of frequent patterns extracted by varying the maximum pattern length (3, 5, 8) while keeping the minimum support threshold fixed to 25%.

Max. patt. length	# Patterns	Execution time
3	769	4816s (1h20s)
5	922	18000s (5h)
8	1,528	18000s (5h)

4.0.1. Scalability tests

With scalability tests, the goal is to assess the performance of Listing 7 over a dataset of increasing size. Also in this case we considered the dataset concerning Karen Cook McNally, from which we extracted test instances ranging over 100, 1000 and 10,000. We ran two groups of tests, by varying the maximum pattern length from 3 to 5, while keeping the minimum support threshold fixed to 25%. Tables 4 and 5 report the results from the two groups.

Table 4
Number of frequent patterns extracted by varying the number of instances (100, 1K, 10K), while leaving the minimum support threshold (25%) and maximum pattern length (3) unchanged.

# Instances	# Sequences	# Patterns	Execution time
100	6	15	0.079s
1,000	9	9,831	34.962s
10,000	93	947	2468.745s (41m15s)

A similar trend appears evident with a peak of patterns found when the number of instances is equal to 1,000. This peak is due to the fact that the 1,000 instances are distributed over 9 sequences and with a threshold equal to 25% in minimum number of sequences necessary for a pattern to be frequent is equal to 3 (rounded up because ASP considers only integers). As for

Table 5
Number of frequent patterns extracted by varying the number of instances (100, 1K, 10K), while leaving the minimum support threshold (25%) and maximum pattern length (5) unchanged.

# Instances	# Sequences	# Patterns	Execution time
100	6	15	0.101s
1,000	9	127,657	625.498s (10m25s)
10,000	93	2,050	18,000s (5h)

the execution time, we observe that the execution ends before the time-out when the patterns have a maximum length of 3. Conversely, when the maximum pattern length goes from 3 to 5, the execution is interrupted for the time-out set at 5 hours with 10,000 instances while the execution time for 1,000 instances goes from 34 seconds to more than 10 minutes.

5. Final remarks

Sequential pattern mining provides a suite of powerful tools for discovering regularities in sequences of events. Therefore it is naturally suitable for analysing evidence in the context of DF investigations. The expressive power of ASP makes the definition of algorithmic variants of the basic encoding pretty easy, mainly thanks to a clever use of constraints. As a case study we have considered the analysis of a real-world dataset of anonymised phone recordings. The preliminary results are encouraging, although they highlight several weaknesses. A major limit of the current encoding is the combinatorial explosion due to several factors. In order to address this limit, we are currently working on extended versions of the basic ASP encoding here presented, which are aimed at mining so-called condensed patterns (maximal and closed).

For the future we intend to significantly go beyond the state of the art in declarative pattern mining, *e.g.*, by devising effective constraints to reduce the size of the output. In parallel to the methodological work, we plan to solicit a feedback from the DF experts involved in DigForASP, as regards the validity and the usefulness of the work from their viewpoint. The interaction with experts could trigger new interesting directions of research.

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

[4] J. Han, H. Cheng, D. Xin, X. Yan, Frequent pattern mining: current status and future directions, Data Min. Knowl. Discov. 15 (2007) 55–86. URL: https://doi.org/10.1007/s10618-006-0059-1. doi:10.1007/s10618-006-0059-1.

[5] S. Jabbour, L. Sais, Y. Salhi, Decomposition based sat encodings for itemset mining problems, in: Pacific-Asia Conference on Knowledge Discovery and Data Mining, Springer, 2015, pp. 662–674.

[6] T. Guns, A. Dries, S. Nijssen, G. Tack, L. De Raedt, Miningzinc: A declarative framework for constraint-based mining, Artificial Intelligence 244 (2017) 6–29.

[7] 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.

[8] 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.

[9] L. De Raedt, T. Guns, S. Nijssen, Constraint programming for data mining and machine learning, in: Twenty-Fourth AAAI Conference on Artificial Intelligence, 2010.

[10] 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.

[11] 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.

[12] 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.

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

[14] 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.