Digital Forensics Evidence Analysis: An Answer Set Programming Approach for Generating Investigation Hypotheses

Stefania Costantini[1] stefania.costantini@univaq.it,
Giovanni De Gasperis[1] giovanni.degasperis@univaq.it, and
Raffaele Olivieri[1,2] raffaele.olivieri@gmail.com

[1] Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica,
Universitá degli Studi dell'Aquila,
Via Vetoio 1, 67100 L'Aquila, Italy
[2] Raggruppamento Carabinieri Investigazioni Scientifiche (Ra.C.I.S.),
(The Italian Department of Scientific Investigations of Carabinieri),
viale di Tor di Quinto 119, 00191 Rome, Italy.

Abstract. The results of the Evidence Analysis phase in Digital Forensics (DF) provides objective data which however require further elaboration by the investigators, that have to contextualize Analysis results within an investigative environment so as to provide possible hypotheses that can be proposed as proofs in court, to be evaluated by lawyers and judges. Aim of our research has been that of exploring the applicability of Answer Set Programming (ASP) to the automatization of Evidence Analysis. This offers many advantages, among which that of making different possible investigative hypotheses explicit, while otherwise different human experts often devise and select different solutions in an implicit way. Moreover, ASP provides a potential for verifiability which is crucial in such an application field. Very complex investigations for which human experts can hardly find solutions turn out in fact to be reducible to optimization problems in classes P or NP or not far beyond, that can be thus expressed in ASP. As a proof of concept, in this paper we present the formulation of some real investigative cases via simple ASP programs, and discuss how this leads to the formulation of concrete investigative hypotheses.

1 Introduzione

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 [1, 2]. The aim is to identify digital sources of proofs, and to organize such proofs in order to make them robust in view of their discussion in court, either in civil or penal trials. Digital Forensics is concerned with the analysis of potential elements of proof after a crime has been committed ("post-mortem"). Clearly, the development of Digital Forensics is highly related to the development of Information and Communication Technologies in the last decades, and to the widespread diffusion of electronic devices and infrastructures. It involves various disciplines such as computer science, electronic engineering, various branches of law, investigation techniques and criminological sciences. Rough evidence must be however used to elicit hypotheses concerning events, actions and facts (or sequences of them) with the goal to present them in court. Evidence analysis involves examining fragmented incomplete knowledge, and defining complex scenarios by aggregation, likely involving time, uncertainty, causality, and alternative possibilities. No single methodology exists today for digital evidence analysis. The Scientific Investigation experts usually work by means of their experience and intuition (expertise).

In fact, Evidence Acquisition is supported by a number of hardware and software tools, both closed and open source. These tools are continuously evolving to follow the evolution of the involved technologies and devices. Instead, Evidence Analysis, is much less supported. In Evidence Analysis, the technicians and experts perform the following tasks. (i) Collect, categorize and revise the evidence items retrieved from electronic devices. (ii) Examine them so as to hypothesize the possible existence of a crime and potential crime perpetrators. (iii) Elicit from the evidence possible proofs that support the hypotheses. (iv) Organize and present the proofs in a form which is acceptable by the involved parties, namely lawyers and judges, which may include to exhibit explicit supporting arguments. Figure 1[3] show some real sequences of the technical activities involved. Few software tools exist that only cover some partial aspects, and all of them

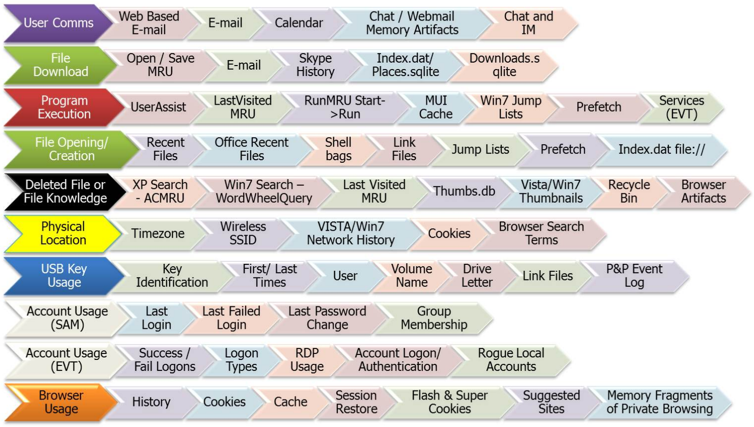

Fig. 1. Example of sequences of technical activities

are “black box” tools, i.e., they provide results without motivation or explanation, and without any possibility of verification. Thus, such results can hardly be presented as reliable proofs to the involved parties. Moreover, the absence of decision support systems leads to undesirable uncertainty about the outcome of Evidence Analysis. Often, different technicians analyzing the same case reach different conclusions, and this may determine different judge’s decisions in court.

[3] Image by courtesy from SANS site url https://blogs.sans.org

Formal and verifiable Artificial Intelligence and Automated Reasoning methods and techniques for Evidence Analysis would be very useful for the elicitation of sources of proof. Several aspects should to be taken into account such as timing of events and actions, possible (causal) correlations, context in which suspicious actions occurred, skills of the involved suspects, validity of alibis, etc. Moreover, given available evidence, different possible underlying scenarios may exist, that should be identified, examined and evaluated. All the above should be performed via "white box" techniques, meaning that such techniques should be verifiable with respect to the results they provide, how such results are generated, and how the results can be explained. The new wished-for software tools should be reliable and provide a high level of assurance, in the sense of confidence in the system's correct behavior. Computational Logic is a suitable candidate to definition and implementation of such tools, and Non-Monotonic Reasoning is clearly extensively required.

The long-term objective of this research is to provide law enforcement, investigators, intelligence agencies, criminologists, public prosecutors, lawyers and judges with decision-support-systems that can effectively aid them in their activities. The adoption of such systems can contribute to making how to proceed clearer and faster, and also under some respects more reliable. In fact, the choice of Computational Logic as a basis guarantees transparency and verifiability of tools and results. The aim of the present paper is to provide a proof-of-concept of the applicability of Computational Logic and Non-Monotonic Reasoning to such tasks. In order to convince the several parties involved, whatever limited their Computer Science expertise might be, we have considered a series of fragments of cases and have transposed them into simple self-explanatory Answer Set Programs which provide results which are easy to understand. However, we have considered fragments of real cases which are presently being investigated by the Italian Department of Scientific Investigations of Carabinieri (the Police branch of the Italian Army http://www.carabinieri.it/multilingua/en).

We have adopted Answer Set Programming (ASP, cf., among others, [3–9]) because ASP programs are declarative and readable and because, as shown in the following sections, as a matter of fact several analysis problems can be nicely reduced to optimization problems for which ASP is particularly well-suited. In fact, the above picture shows how the analysis phase consists in an ordered sequence of detailed technical activity, performed on the seized memories (or on their forensic copy) following hard rules and formal procedures, through tools and/or forensic equipment, to research specific elements, perform the verification of a state or condition, etc. The outcome of the analysis is a detailed and well-motivated technical report. However these reports , although providing a comprehensive response in technical terms, may be insufficient or even not directly usable from the investigation point of view. Further elaboration is in general needed in order to contextualize technical data in the investigation context. Due to the experience in DF gained by one of the authors of this paper as a member officer of a DF Laboratory of an Italian Police Force, with national jurisdiction, it has been possible to identify and experimentally treat (fragments of) complex investigations by reduction to Answer Set Programming. This paper presents the results of these experiments. Results are indeed very promising, as the reduction of investigation cases to Answer Set Programming has allowed the experts to identify new investigative hypotheses, that have been practically exploited.

The paper is organized as follows: in Section 2 we provide a short introduction to ASP; in Sections 3- 5 we present three simple though representative examples; finally, in Section 6 we conclude.

2 Answer Set Programming (ASP) in a Nutshell

"Answer Set Programming" (ASP) is a well-established logic programming paradigm adopting logic programs with default negation under the *answer set semantics*, which [3, 4] is a view of logic programs as sets of inference rules (more precisely, default inference rules). In fact, one can see an answer set program as a set of constraints on the solution of a problem, where each answer set represents a solution compatible with the constraints expressed by the program. The reader may refer to [3–9], among others, for a presentation of ASP as a tool for declarative problem-solving.

Syntactically, a program (or, for short, just "program") Π is a collection of *rules* of the form $H \leftarrow\ L_1, \ldots, L_m,\ not\ L_{m+1}, \ldots, not\ L_{m+n}$

where H is an atom, $m \geq 0$ and $n \geq 0$, and each L_i is an atom. Symbol $\leftarrow$ is usually indicated with :- in practical systems. An atom L_i and its negative counterpart $not\, L_i$ are called *literals*. The left-hand side and the right-hand side of the clause are called *head* and *body*, respectively. A rule with empty body is called a *fact*. A rule with empty head is a *constraint*, where a constraint of the form $\leftarrow\ L_1, ..., L_n.$ states that literals $L_1, \ldots, L_n$ cannot be simultaneously true in any answer set.

A program may have several answer sets, each of which represent a solution to given problem which is consistent w.r.t. the problem description and constraints. If a program has no answer set, this means that no such solution can be found. and the program is said to be *inconsistent* (w.r.t. *consistent*).

In practical terms a problem encoding, in the form of ASP program, is processed by an ASP solver which computes the answer set(s) of the program, from which the solutions can be easily extracted (by abstracting away from irrelevant details). Several well-developed answer set solvers [10] that compute the answer sets of a given program can be freely downloaded by potential users. All solvers provide a number of additional features useful for practical programming, that we will introduce only whenever needed. Solvers are periodically checked and compared over well-established benchmarks, and over challenging sample applications proposed at the yearly ASP competition (cf. [11] for a recent report).

The expressive power of ASP, as well as, its computational complexity have been deeply investigated [12]. Precisely, in the propositional case the problem of deciding whether a given program admits answer sets is NP-complete, and so is the problem of deciding whether there is an answer set containing a specific atom (while deciding whether a specific atom is in all the answer sets is Co-NP.-complete). ASP is clearly able to express NP-complete problems.

3 Case 1: Data Recovery and File Sharing Hypotheses

3.1 The Investigative Case

The Judicial Authority requested the Digital Forensics (DF) Laboratory to analyze the contents of an hard disk, in order to check for the presence of illegal contents files. If so, they requested to check for potential activities of sharing on Internet of illegal materials. The hard disk under analysis was physically damaged (as often done by criminals if they suspect capture). Therefore, after a head replacement, the Evidence Acquistion phase recovered a large amount of files (of various types: images, videos, documents, etc.), however without their original name. This because the damage present on the disk plates disallowed the recovery the information of the MFT[4]. For this reason, an arbitrary name has been assigned to all the files recovered. Information about the original name of files and their original location in the file system is thus missing.

3.2 Elements

By analyzing the recovered files, technicians detected the occurrence of:

- files with illegal contents;
- various “INDX files”, corresponding in the NTFS file system to directory files, which contains the follow METADATA:
  - filename;
  - physical and logical size of the file;
  - created, accessed, modified and changed timestamps;
- index related to the eMule (which is a widely-used file-exchange application), including a file containing sharing statistics, whose original name is “known.met”.

Starting from the elements described above, we have been able to reply to the Judicial Authority’s question with: a reasonably reliable hypothesis of association of the recovered file to the respective original name; a reasonable certainty that illegal files were actually exchanged on the Internet. This has been obtained by modeling the given problem by means of a very simple well-known ASP example, reported below.

3.3 The Marriage Problem

The Marriage Problem (or SMP - Stable Marriage Problem) is a well-known NP-hard optimization problem which finds a stable matching between two sets of elements S_1 and S_2 (say MEN and WOMEN) given a set of preferences for each element. A matching is a mapping from the elements of one set to the elements of the other set which thus creates a set of couples (A, B) where $A \in S_1$ and $B \in S_2$. A matching is stable whenever it is not the case that some element $\hat{A}$ of the first matched set prefers some given element $\hat{B}$ of the second matched set over the element to which $\hat{A}$ is already matched, and the same holds for $\hat{B}$.

[4] Master file Table: structured block table containing the attributes of all files in the volume of an NTFS file system.

3.4 Reduction

The given problem is in fact reducible to SMP as follows. In the real case, the lists have been created as follow:

- MEN list: defined as the list of names extracted from directory files “INDX files”;
- WOMEN: defined as the list of recovered files with have been provisionally assigned arbitrary names.

The PREFERENCES list (or relation order) between the *MEN* and *WOMEN* lists is derived from the comparison of the properties of the individual recovered files (file type, size, etc.) with those identified in file ‘INDX files”.

3.5 Answer Set Programming Solution

Once compiled the lists MEN, WOMEN and PREFERENCES, you can search for answer sets by means of the following ASP program (in the syntax of the *smodels* solver). Facts in the program correspond to a real (though very small) example.

```
preference(f001, flower_jpg).
preference(f001, woman_jpg).
preference(f002, flower_jpg).
preference(f002, child_jpg).
preference(f003, child_jpg).
preference(f003, woman_jpg).

bigamy(X,Y) :- relation(X,Y), preference(X,Y1), couple(X,Y), couple(X,Y1), Y!=Y1.
bigamy(X,Y) :- preference(X,Y), preference(X1,Y), couple(X1,Y), X!=X1.
couple(X,Y) :-  preference(X,Y), not bigamy(X,Y).

#hide.
#show couple(X,Y).
```

3.6 Results

The results obtained with the *smodels* solver on the real example are the follow:

```
smodels version 2.26.
Answer: 1
Stable Model: couple(f002,child_jpg) couple(f001,woman_jpg) couple(f001,flower_jpg)
Answer: 2
Stable Model: couple(f003,child_jpg) couple(f001,woman_jpg) couple(f001,flower_jpg)
Answer: 3
Stable Model: couple(f003,child_jpg) couple(f002,flower_jpg) couple(f001,woman_jpg)
Answer: 4
Stable Model: couple(f002,child_jpg) couple(f002,flower_jpg) couple(f001,woman_jpg)
Answer: 5
Stable Model: couple(f003,woman_jpg) couple(f002,child_jpg) couple(f002,flower_jpg)
Answer: 6
Stable Model: couple(f003,woman_jpg) couple(f002,child_jpg) couple(f001,flower_jpg)
Answer: 7
Stable Model: couple(f003,woman_jpg) couple(f003,child_jpg) couple(f001,flower_jpg)
Answer: 8
Stable Model: couple(f003,woman_jpg) couple(f003,child_jpg) couple(f002,flower_jpg)
```

From the answer sets, it is possible (as the reader can see) to formulate hypotheses about the original names of the recovered files. Furthermore, by comparing the file names indexed in the file *Known.met*[5], it has been possible to make reasonable assumptions about the effective sharing of files with illegal content.

4 Case 2: Path Verification

4.1 The Investigative Case

After a heinous crime, an allegedly suspect has been arrested. The police sequestered all his mobile devices (smartphone, route navigator, tablet, etc...). The Judicial Authority requested the DF Laboratory to analyze the digital contents of the mobile devices in order to determine their position with respect to the crime site during an interval of time which includes the estimated time when the crime was perpetrated.

4.2 Elements

From the analysis of the mobile devices, a set of geographical GPS coordinates have been extracted, some of them related to the the time interval under investigation. There are however some gaps, one of them certainly due to a proven switch off of few minutes around the crime time. To start with, a list called GPS-LIST is generated, collecting all the positions extracted from the various devices, grouped and ordered by *time unit* of interest (seconds, multiple of seconds, minutes, etc..). The objective is that of establishing whether the known GPS coordinates are compatible with some path which locates the given mobile devices at the crime site during the given time interval. If no such path exists, then the suspect must be discharged. If some compatible path is found, then the investigation about the potential perpetrator can proceed. The objective has been reached via reduction to the following simple game.

4.3 Hidato Puzzle (Hidoku)

Hidato is a logical puzzle (also known as "Hidoku") invented by the Israeli mathematician Dr. Gyora Benedek. The aim of Hidato is to fill a matrix of numbers, partially filled a priori, using consecutive numbers connected over a horizontal, vertical or diagonal ideal line. Below we show, as a simple example, a 4x4 initial matrix.

18	0	0	0	26	0
19	0	0	27	0	0
0	14	0	0	23	31
1	0	0	8	33	0
0	0	5	0	0	0
0	0	10	0	36	35

[5] As mentioned, known.met is a file of the widely-used *eMule* file-exchange application that stores the statistics of all files that the software shared, all files present in the download list and downloaded in the past.

4.4 Reduction

It has been possible to perform the reduction of the given investigation problem to the “Hidato Puzzle” problem, by creating a matrix representing the geographical area of interest, where each element of the matrix represents a physical zone crossable in a unit of time. The physical size of the individual cell of the matrix (grid) on the map will be proportionate to the time unit that will be considered, both the hypothetical transfer speed. The matrix has been populated with the elements of the previously-created LIST-GPS, i.e., with known positions of the suspect.

Considering the above matrix, assume that the crime has been committed at location 34, at a time included in the interval with lower bound corresponding to when the suspect was at location 1 and upper bound corresponding to when the suspect was at location 36. All devices have been provably switched off between locations 5 and 10.

4.5 Answer Set Programming Solution

Once built the matrix, we can determine whether a suspect route exists by finding the answer sets of the following ASP program [13] (here we have used the *clingo* solver). Notice that the omitted cells are assumed to have value 0.

```
#const n = 6.
matrix(1, 1, 18).  matrix(1, 5, 26).  matrix(2, 1, 19).  matrix(2, 4, 27).
matrix(3, 2, 14).  matrix(3, 5, 23).  matrix(3, 6, 31).  matrix(4, 1, 1).
matrix(4, 4, 8).   matrix(4, 5, 33).  matrix(5, 3, 5).   matrix(6, 3, 10).
matrix(6, 5, 36).  matrix(6, 6, 35).

size(1..n).
values(1..n*n).
values2(1..n*n-1).
diffs(-1;0;1).

1 { x(Row, Col, Value) : values(Value) } 1 :- size(Row), size(Col).
1 { x(Row, Col, Value) : size(Row) : size(Col) } 1 :- values(Value).
x(Row, Col, Value) :- matrix(Row, Col, Value).

valid(Row, Col, Row2, Col2) :- diffs(A), diffs(B), Row2 = Row+A, Col2 = Col+B,
                                   Row2 >= 1, Col2 >= 1, Row2 <= size, Col2 <= size,
                                   size(Row), size(Col).

:- x(Row, Col, Value+1), x(Row2, Col2, Value),
   not valid(Row, Col, Row2, Col2), values2(Value).
#hide.
#show x(Row, Col, Value).
```

4.6 Results

The results obtained via the *clingo* solver are the following:

```
Answer: 1
x(1,1,18) x(1,5,26) x(2,1,19) x(2,4,27) x(3,2,14) x(3,5,23) x(3,6,31) x(4,1,1)
x(4,4,8) x(4,5,33) x(5,3,5) x(6,3,10) x(6,5,36) x(6,6,35) x(5,1,2) x(6,1,3)
x(6,2,4) x(6,4,6) x(5,5,7) x(5,4,9) x(5,2,11) x(4,2,12) x(3,1,13) x(4,3,15)
x(3,3,16) x(2,3,21) x(3,4,22) x(2,6,24) x(1,6,25) x(1,3,28) x(1,4,29) x(2,5,30)
x(4,6,32) x(5,6,34) x(1,2,20) x(2,2,17)

Answer: 2
x(1,1,18) x(1,5,26) x(2,1,19) x(2,4,27) x(3,2,14) x(3,5,23) x(3,6,31) x(4,1,1)
x(4,4,8) x(4,5,33) x(5,3,5) x(6,3,10) x(6,5,36) x(6,6,35) x(5,1,2) x(6,1,3)
x(6,2,4) x(6,4,6) x(5,5,7) x(5,4,9) x(5,2,11) x(4,3,12) x(3,3,13) x(4,2,15)
x(3,1,16) x(2,3,21) x(3,4,22) x(2,6,24) x(1,6,25) x(1,3,28) x(1,4,29) x(2,5,30)
x(4,6,32) x(5,6,34) x(1,2,20) x(2,2,17)
```

These results are particulary interesting for the investigation, as they both correspond to paths which are compatible with the hypothesis of the suspect committing the crime.

18	20	28	29	26	25
19	17	21	27	30	24
13	14	16	22	23	31
1	12	15	8	33	32
2	11	5	9	7	34
3	4	10	6	**36**	35

18	20	28	29	26	25
19	17	21	27	30	24
16	14	13	22	23	31
1	15	12	8	33	32
2	11	5	9	7	34
3	4	10	6	**36**	35

It should be noted that a variant of the Hidato algorithm exists, that considers maps whose structure is more complex than a rectangular matrix.

5 Case 3: Alibi Verification

5.1 The Investigative Case

During an investigation concerning a bloody murder, it is necessary check the alibi provided by a suspect. In the questioning, the suspect has been rather vague about the timing of his movements. However, he declared what follows.

- to have left home (place X) at a certain time;
- to have reached the office at place Y where he worked on the computer for a certain time;
- to have subsequently reached place Z where, soon after opening the entrance door, he discovered the body and raised the alarm.

In order to verify the suspect's alibi, the Judicial Authority requested the DF Laboratory to analize:

- the contents of the smartphone owned by the suspect;
- the computer confiscated in place Y, where the suspect says to have worked;
- a video-surveillance equipment installed at a post office situated near place Z, as its video-camera surveys the street that provides access to Z.

5.2 Elements

The coroner's analysis on the body has established the temporal interval including the time of death. From the forensic analysis of the smarphone it has been possible to compile a list of GPS positions related to a time interval including the time of death, denoted by GPS-LIST. The analysis of the computer allowed the experts to extract the list of accesses on the day of the crime, denoted by LOGON-LIST. The analysis of the video-surveillance equipment allowed the experts to isolate some sequences, denoted by VIDEO-LIST, that show a male subject whose somatic features are compatible with the suspect. All the above lists have been ordered according to the temporal sequence of their elements. The investigation case at hand can be modeled as a planning problem where time is a fundamental element in order to establish whether a sequence of actions exist that may allow to reach a certain objective within a certain time. Several approaches to causal and temporal reasoning in ASP exist, that could be usefully exploited for this kind of problem[6]. Here, for lack of space and for the sake of simplicity we model the problem by means of the very famous "Monkey & Banana" problem, which is the archetype of such kind of problems in Artificial Intelligence.

5.3 Monkey & Banana

The specification of "Monkey & Banana" is the following: A monkey is in a room. Suspended from the ceiling is a banana, beyond the monkey's reach. In the room there is also a chair (in some versions there is a stick, that we do not consider). The ceiling is just the right height so that a monkey standing on a chair could knock the banana down (in the more general version by using the stick, in our version just by hand). The monkey knows how to move around, carry other things around, reach for the banana. What is the best sequence of actions for the monkey? The initial conditions are that: the chair is not just below the bananas, rather it is in a different location in the room; the monkey is in a different location with respect to the chair and the bananas.

5.4 Reduction

The reduction of the case at hand to the 'Monkey & Banana" problem is the following. Notice the reduction of the "idle" state of the monkey to unknown actions that the suspect may have performed at that time.

Monkey	→	Suspect
Banana	→	Body
Eats Banana	→	Raise Alarm
Initial Position Monkey	→	X
Initial Position Chair	→	Y
Below Banana	→	Z
Walks	→	Walks
Move Chair	→	Motion to Z
Ascend	→	Open the Door
Idle	→	Unknown Action

Problem's constraint are that, at any time, the monkey:

- may perform only one action at each time instant among walk, move chair, stand on chair, or stay idle;
- if the monkey stands on the chair, it cannot walk, and it cannot climb further;
- if the chair is not moved then it stays where it is, and vice versa if it is moved it changes its position;
- the monkey is somewhere in the room, where it remains unless it walks, which implies changing position;

[6] For lack of space we cannot provide the pertinent bibliography: please refer to [14] and to the references therein.

- the monkey may climb or move the chair only if it is in the chair's location;
- the monkey can reach the banana only if it has climbed the chair, and the chair is under the banana.

5.5 Answer Set Programming Solution

The following ASP program, obtained by modifying a version that can be found at http://www.dbai.tuwien.ac.at/proj/dlv/tutorial/, is formulated for the DLV solver, and provides in the answer sets the timed sequences of actions (if any exists) by which the monkey can reach and eat the banana.

```
walk(Time) v move_chair(Time) v ascend(Time) v idle(Time) v eats_banana(Time) :-
                                                                         #int(Time).
monkey_motion(T) :- walk(T).
monkey_motion(T) :- move_chair(T).

stands_on_chair(T2) :- ascend(T), T2 = T + 1.
:- stands_on_chair(T), ascend(T).
:- stands_on_chair(T), monkey_motion(T).
stands_on_chair(T2) :- stands_on_chair(T), T2 = T + 1.

chair_at_place(X, T2) :-
    chair_at_place(X, T1), T2 = T1 + 1,not move_chair(T1).
chair_at_place(Pos, T2) :-
    move_chair(T1),T2 = T1 + 1,monkey_at_place(Pos, T2).
:- move_chair(T1),chair_at_place(Pos,T2),chair_at_place(Pos1,T1),T2 = T1+1, Pos=Pos1.

monkey_at_place(monkey_starting_point, T) v
monkey_at_place(chair_starting_point, T) v
monkey_at_place(below_banana, T) :- #int(T).

:- monkey_at_place(chair_starting_point, 0).
:- monkey_at_place(below_banana, 0).
:- not monkey_at_place(monkey_starting_point, 0).

:- monkey_at_place(Pos1, T2),
   monkey_at_place(Pos2, T1), T2 = T1 + 1,
   Pos1 != Pos2, not monkey_motion(T1).
:- monkey_at_place(Pos, T2), monkey_at_place(Pos, T1),
   T2 = T1 + 1, monkey_motion(T1).
:- ascend(T),monkey_at_place(Pos1, T),
   chair_at_place(Pos2, T),Pos1 != Pos2.
:- move_chair(T),monkey_at_place(Pos1, T),
   chair_at_place(Pos2, T),Pos1 != Pos2.

monkey_at_place(monkey_starting_point, 0) :- true.
chair_at_place(chair_starting_point, 0) :- true.

reach_banana(T) :- can_reach_banana(T).
can_reach_banana(T) :- stands_on_chair(T),
        chair_at_place(below_banana, T).
:-eats_banana(T), not can_reach_banana(T).
:- eats_banana(T1),eats_banana(T2), T1!=T2.
happy :- eats_banana(T).
:- not happy.

step(N, walk, Destination) :- walk(N),
         monkey_at_place(Destination, N2),N2 = N + 1.
step(N, move_chair, Destination) :-
         move_chair(N),monkey_at_place(Destination, N2),
         N2 = N + 1.
step(N, ascend, " ") :- ascend(N).
step(N, idle, " ") :- idle(N).
step(N, eats_banana, " ") :- eats_banana(N).
```

5.6 Results

The proposed reduction in the first place allows investigators to verify the alibi provided by the suspect. In fact, the possible timed lists of actions performed by the suspect are determined as answer sets of the above program. Such lists are constructed so as to be compatible with the detected GPS positions of the suspect, the detected computer activity and the actions that the suspect has declared to have performed. By running the solver on the real case with a maximum number of steps N = 3, corresponding to the case where the suspect is provably at the office at time 0, we get exactly the action sequences needed to reach the goal.

```
 {step(0,walk,chair_starting_point), step(1,move_chair,below_banana),
  step(2,ascend," "), step(3,eats_banana," "
```

Therefore, if the suspect raised the alarm at time 3 he actually had no time for committing the crime and and therefore he should presumably be discharged.

In case instead the alibi is not fully verified, then further investigation is needed. By increasing the time, for example to N = 5, we in fact get many sets of possible alternative actions, where *idle* is an unknown action for which it might interesting to investigate further so as to prove or reject the investigation thesis.

```
{step(0,idle," "),step(1,walk,chair_starting_point),
step(2,move_chair,below_banana), step(3,ascend," "),
step(4,idle," "),step(5,eats_banana," "),}

{step(0,walk,below_banana), step(1,walk,chair_starting_point),
step(2,move_chair,below_banana), step(3,ascend," ")
step(4,idle," "), step(5,eats_banana," ") }
```

Among the answer sets there are many which suggest suspicious behavior. The first one above outlines a scenario where the initial suspect's actions are unknown. Then he moves to the crime site where however he has the time and opportunity to commit the crime at step 4. Even worse is the second answer set, where the suspect moves to the crime site, than moves back to the office, moves a second time to the crime site where again he has the time and opportunity to commit the crime at step 4. As the suspect's presence at the crime site is confirmed by the video-surveillance equipment records, this behavior is suggestive of, e.g., going to meet the victim and having a discussion, going back to the office (maybe to get a weapon) and then actually committing the crime.

6 Conclusions

In this paper we have demonstrated the applicability of Non-Monotonic Reasoning techniques to evidence analysis in Digital Forensics by mapping some fragments of real cases to existing simple answer set programs. The application of Artificial Intelligence and in particular of Non-Monotonic Reasoning techniques to Evidence Analysis is a novelty: in fact, even very influential publications in Digital Forensics such as [1, 2] are basically a guide for human experts about how to better understand and exploit digital data. Therefore the present work, though preliminary, opens significant new perspectives. Future developments include building a toolkit exploiting not only ASP but also other non-monotonic-reasoning techniques such as abduction, temporal reasoning, causal reasoning and others, as elements of decision-support-systems that can effectively aid investigation activities and support of the production of evidence to be examined in trial. The multidisciplinary future challenge is that of making such tools formally accepted in court proceedings: this involves in general terms complex societal and psychological issues. From the technical point of view, for making such tools acceptable and perceived as reliable, it is crucial to develop verification, certification, assurance and explanation techniques.

References

1. Casey, E.: Handbook of Digital Forensics and Investigation. Elsevier (2009)
2. Casey, E.: Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet. books.google.com (2011)
3. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In Kowalski, R., Bowen, K., eds.: Proc. of the 5th Intl. Conf. and Symposium on Logic Programming, MIT Press (1988) 1070–1080
4. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Generation Computing **9** (1991) 365–385
5. Baral, C.: Knowledge representation, reasoning and declarative problem solving. Cambridge University Press (2003)
6. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In: Logic Programming and Nonmonotonic Reasoning, 9th Intl. Conference, LPNMR 2007. (2007)
7. Truszczyński, M.: Logic programming for knowledge representation. In Dahl, V., Niemelä, I., eds.: Logic Programming, 23rd Intl. Conference, ICLP 2007. (2007) 76–88
8. Dovier, A., Formisano, A., Pontelli, E.: An empirical study of constraint logic programming and answer set programming solutions of combinatorial problems. Journal of Experimental and Theoretical Artificial Intelligence **21**(2) (2009) 79–121
9. Dovier, A., Formisano, A.: Programmazione Dichiarativa in Prolog, CLP, ASP, e CCP. (2008) Available (in Italian) at `https://users.dimi.uniud.it/~agostino.dovier/DID/corsi.html`.
10. Web references of ASP solvers: Clasp: `potassco.sourceforge.net`; Cmodels: `www.cs.utexas.edu/users/tag/cmodels`; DLV: `www.dbai.tuwien.ac.at/proj/dlv`; Smodels: `www.tcs.hut.fi/Software/smodels`.
11. Calimeri, F., Ianni, G., Krennwallner, T., Ricca, F.: The answer set programming competition. AI Magazine **33**(4) (2012) 114–118
12. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Computing Surveys **33**(3) (2001) 374–425
13. Kjellerstrand, H. Available at `http://www.hakank.org/answer_set_programming` (2015)
14. Cabalar, P.: Causal logic programming. In: Correct Reasoning - Essays on Logic-Based AI in Honour of Vladimir Lifschitz. Volume 7265 of Lecture Notes in Computer Science., Springer (2012) 102–116