Annals of Mathematics and Artificial Intelligence (2019) 86:193–229
https://doi.org/10.1007/s10472-019-09632-y

Digital forensics and investigations meet artificial intelligence

Stefania Costantini[1,2] · Giovanni De Gasperis[1,2] · Raffaele Olivieri[1,2]

Published online: 24 April 2019

Abstract
In the frame of Digital Forensic (DF) and Digital Investigations (DI), the "Evidence Analysis" phase has the aim to provide objective data, and to perform suitable elaboration of these data so as to help in the formation of possible hypotheses, which could later be presented as elements of proof in court. The aim of our research is to explore the applicability of Artificial Intelligence (AI) along with computational logic tools – and in particular the Answer Set Programming (ASP) approach — to the automation of evidence analysis. We will show how significant complex investigations, hardly solvable for human experts, can be expressed as optimization problems belonging in many cases to the $\mathbb{P}$ or $\mathbb{NP}$ complexity classes. All these problems can be expressed in ASP. As a proof of concept, in this paper we present the formalization of realistic investigative cases via simple ASP programs, and show how such a methodology can lead to the formulation of tangible investigative hypotheses. We also sketch a design for a feasible Decision Support System (DSS) especially meant for investigators, based on artificial intelligence tools.

Keywords Digital forensics · Digital investigation · Artificial intelligence · Answer set programming · Automatic investigation · Forensic models

Mathematics Subject Classification (2010) 68T27 · 97R40

✉ Stefania Costantini
stefania.costantini@univaq.it

Giovanni De Gasperis
giovanni.degasperis@univaq.it

Raffaele Olivieri
raffaele.olivieri@gmail.com

1 Università degli Studi dell'Aquila, L'Aquila, Italy

2 Dipartimento di Ingegneria e Scienze dell'Informazione e Matematica via Vetoio snc Loc. Coppito, I-67100, L'Aquila, Italy

1 Introduction

An investigation consists, in general terms, in the series of actions and initiatives implemented by the investigators (law enforcement and judges) in order to ascertain the "*truth*" and acquire all possible information and data about a perpetrated crime and related facts with their logical implications. A large number of subjects are involved in this process, where they help to pursue a criminal activity, which could still be in progress. In an accurate vision, and according to the Italian Code of Criminal Procedure, investigations can be defined as "the set of activities carried out by the *officers* and *agents* of the *criminal police*". An investigation has, overall, the aim of establishing the existence of a crime and the consequences that it has determined (generic proof or "*de delicto*"), and identifying the criminals (specific proof or "*de reo*").

These activities start from the act of acquisition of the crime notice or from the analysis of a crime scene. Through a series of initiatives and actions, the investigation allows the collection of data and elements which, according to certain deductive logical reasoning, should lead to draw conclusions. Investigative cases are usually complex, and involve a number of factors that need to be taken into account. Most of the collected data are nowadays obtained through digital devices and platforms either seized from the suspects, or available on the Internet or shared by telecommunication companies. Thus, Digital Forensics 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. In particular, the main focus of this paper is to address the phase of "Evidence Analysis", performed on data collected from various electronic devices by the means of specialized software, and according to specific regulations.

Such evidence is examined and aggregated so as to reconstruct possible events, event sequences and scenarios related to a crime. Results of the Evidence Analysis phase are elements of proof that are then made available to law enforcement, investigators, intelligence agencies, public prosecutors, lawyers and judges. In the case of a cybercrime for example, in line with the recommendation of the European Commission: "At a national level, Member States should ensure common standards among police, judges, prosecutors and forensic investigators in investigating and prosecuting cybercrime offenses". Concerning the policies used by Police institutions for sharing technologies and information, the European institutions adopt procedures established within the ENFSI (European Network of Forensic Science Institutes).

Nowadays, commercial-off-the-shelf (COTS) software products exist to manage Digital Forensics cases. Such products are usually workflow-driven, and provide mechanisms for information access, for search and data visualization. However, support for the effective aggregation and organization of useful evidence is simply non-existent. Human experts proceed to the analysis of data, including temporal sequences, and reach their conclusions according to their intuition and their experience. A formal explanation of such conclusions cannot generally be provided. Often, different experts reach different conclusions, and the arguments that they provide in support seem equally valid.

Although each case has distinct characteristics, and a variety of cases can be found, practical experience allows one to notice that many cases, or better significant fragments of cases, be modeled via well-known algorithmic puzzles. As an officer of the Digital Forensics Laboratory of an Italian police force with national jurisdiction, Raffaele Olivieri (one of the authors of the present paper) has performed, for his Ph.D thesis, [49] a deep analysis of a great number of real cases. The thesis has been carried on as a part-time research student, in parallel with everyday investigation work. The aim of attending a Ph.D. was exactly that of devising novel techniques to be applied to cases in the Laboratory, which has actually been done in practice. Such analysis has made it possible in some parts of investigative cases to identify and experimentally treat samples which are representative of significant classes of complex investigations by analogy to known Computational Logic techniques. In particular, we have adopted *Answer Set Programming* (ASP) problem encodings. ASP is in fact a well-established Computational Logic paradigm for solving combinatorial/optimization/planning problems starting from a declarative specification of them and their associated constraints. Solutions to the given problem or, better, plausible scenarios underlying the ASP problem formulation can be found via publicly available inference engine called "solvers".

Thanks to the experience gained over the years in investigations, we are able to claim with good reason that indeed a wide range of real cases can be mapped to computational problems, often to known ones. Modern investigative activities are composed of well-established practical steps, such as the crime scene reconstruction, alibi verification, as well as the analysis of huge amounts of data coming from data files, smart-phone and telephone logs. So, in this paper we present ASP formulations of these sample problems. We do not pretend though that every possible case is reducible to such problem templates. Our wider long-term perspective is in fact to construct a tool-kit that can be extended in time, to provide support to the investigation activities, automating most of the low level data handling methods, supporting the investigator at the abstract level as well. When applicable, the ASP formulations generate all possible scenarios compatible with the case's data and constraints. In the general case, this can be of great help as the human expert might sometimes overlook some of the possibilities. This has been verified by everyday practice, where different experts often generate different interpretations.

In a future perspective, we may notice that logical methods (like ASP) could provide a broad range of proof-based reasoning functionalities (including, e.g., time and time intervals logic, causality, forms of induction, etc.) that can be possibly integrated into a declarative framework for Evidence Analysis where the problem specification and the computational program are closely aligned. The encoding of cases via such tools would have the benefit that (at least in principle) correctness of such declarative systems based on computational logic can be formally verified. Moreover, recent research has led to new methods for visualizing and explaining the results of computed answers (e.g., based on argumentation schemes). So one could not only represent and solve relevant problems, but might also employ suitable tools to explain the conclusions (and their proofs) in a transparent, comprehensible and justified way.

The mapping of a case to a computational-logic-based formulation is clearly a responsibility of the analyst, who may hopefully succeed or possibly fail to devise it. More generally, it may happen that an investigator fails to find the solution to a case exactly because he is not aware of suitable techniques and tools, or because he fails in reconstructing the logical thread that connects the available elements. The European judicial authorities, in case of unsolved or controversial cases, refer in fact to external consultants to ascertain that all that could be possibly done has actually been done by the investigators in charge. So, the investigators are in practice expected to be able to make appropriate analogies in order to decode case elements, and are (nowadays) supposed to be able to resort to automated tools whenever available. Thus, it is by no means unreasonable to assume that the proposed methodology, which improves over the (few) existing tools, can be used by investigators. Digital forensic experts are usually computer scientists (at the Master or Ph.D. level) who are more skilled than average and who have the duty to keep themselves updated and well informed about the latest ICT technological advancements.

In the future however, we envisage the specification and implementation of Decision Support Systems (DSS) to support the analysts in such task. Available investigative information should be treated, in this perspective, via algorithmic solutions whose correctness can be proven. A very important point concerns the complexity of the required algorithms. After a thorough analysis and systematization of past real cases, anonymising data, we have been able to assess that $\mathbb{NP}$ formulations are often sufficient, with few cases where one has to climb a further level of the polynomial hierarchy. The $\mathbb{NP}$ representation is mandatory, as in general the available data will not provide a unique solution, but rather a set of plausible scenarios which are compatible with what is known. In particular cases, the $\mathbb{NP}$ forensic formulation has a solution in $\mathbb{P}$. Though an investigative case may involve several data, it translates in general into a relatively small or medium instance of an $\mathbb{NP}$ problem, solvable by state-of-the-art ASP inference engines. However, case analysis is required to obtain the instance to be calculable in a timely process, that must be performed with due degree of accuracy and precision, so the necessary CPU time can be scheduled and spent.

In this paper, we make a first step in the above-outlined direction. In fact we illustrate, as a proof of concept, the transposition of a number of sample common investigation cases into *Answer Set Programming* (ASP), and we devise in general terms a methodology for doing so.

ASP is, as mentioned, a well-established paradigm for representing problems in $\mathbb{P}$ and $\mathbb{NP}$ or, with some extensions, even higher in the polynomial hierarchy [4, 27, 28, 31, 40, 43, 57]. ASP has been selected for our experiments because of its easy of use and readability, for the availability of efficient inference engines ("ASP solvers", [3]) and for the possibility of performing proof of correctness of the software (the reader may refer to [41, 50, 51] for the definition of the underlying formal properties); recent research work [2, 22, 25] allows such proofs to be resilient w.r.t. changes in the instance or even (with some well-defined restrictions) in the representation. The ASP approach to problem-solving consists basically in the following: (i) encoding of given problem via an ASP program; (ii) computing the "answer sets" of such program via an inference engine, or "ASP solver", considering that a program may have none, one or several answer sets; if no answer set exists, then the given problem instance is not consistently solvable; different answer sets represent instead alternative solutions; (iii) extracting the problem solution by examining such answer sets so as to choose the most suitable one; answer set contents must be in general reformulated in the terms of given problem. Many real-world (industrial) applications have demonstrated the effectiveness of ASP in practice, since complex business-logic features can be developed with ASP-based technologies at a lower implementation price with respect to imperative languages. This in addition to other several advantages observed from the software engineering perspective, while developing real world applications, like flexibility, readability, extensibility, ease of maintenance, etc. Indeed, the possibility of modifying complex reasoning tasks by simply editing a text file with ASP rules, and testing it on web site together with the customer, is a widely-recognized advantage of ASP-Based development. This aspect of ASP-based software development has been a success factor for applications where the high complexity of the requirements was a main obstacle. Moreover, ASP solvers implement advanced optimization techniques leading to a huge speed up of the computation so as to handle real-world applications where a timely response has to be provided to instances involving huge data sets. In addition, ASP has been endowed with effective programming tools supporting the activities of researchers and implementers and simplify the integration of ASP into the existing and well-assessed development processes and platforms.

We choose on purpose to translate some sample investigation problems into well-known combinatorial problems and to use for demonstration existing ASP encodings, in analogy to the reduction methodology that is customary in complexity theory. This is because our intent here was not that of devising new code, rather it was exactly that of demonstrating how sample cases might be reduced to well-known computational problems. These encodings and many others might in perspective constitute elements to exploit, combine and customize in the envisioned Decision Support System. We do not claim that such a system should be based on ASP only: on the contrary, we believe that many other AI techniques can be exploited in a synergistic way in this field; for instance, but not exclusively, abduction, abductive logic programming and argumentation (cf. among many, [1, 34, 35, 38]), modal and temporal logics in various forms [5, 26, 39, 52, 58] or hybrid approaches [12, 47], probabilistic logic programming [54], are all good candidates under many aspects. In fact, a plausible architecture for the envisioned system is depicted in Fig. 1. There, an intelligent agent is supposed to be in charge of supporting the human investigator in her activities. The agent should help identify, retrieve and gather the various kinds of potentially useful evidence, process them via suitable reasoning modules, and integrate the results into coherent evidence. In this task, the agent may need to retrieve and exploit knowledge bases concerning, e.g., legislation, past cases, criminal history of the suspects, and so on. The picture should be seen simply as an illustration of the envisaged environment with no pretence of precision; the agent considers the sample proof elements discussed in the next sections: results from blood-pattern analysis on the crime scene, which lead to model such a scene via

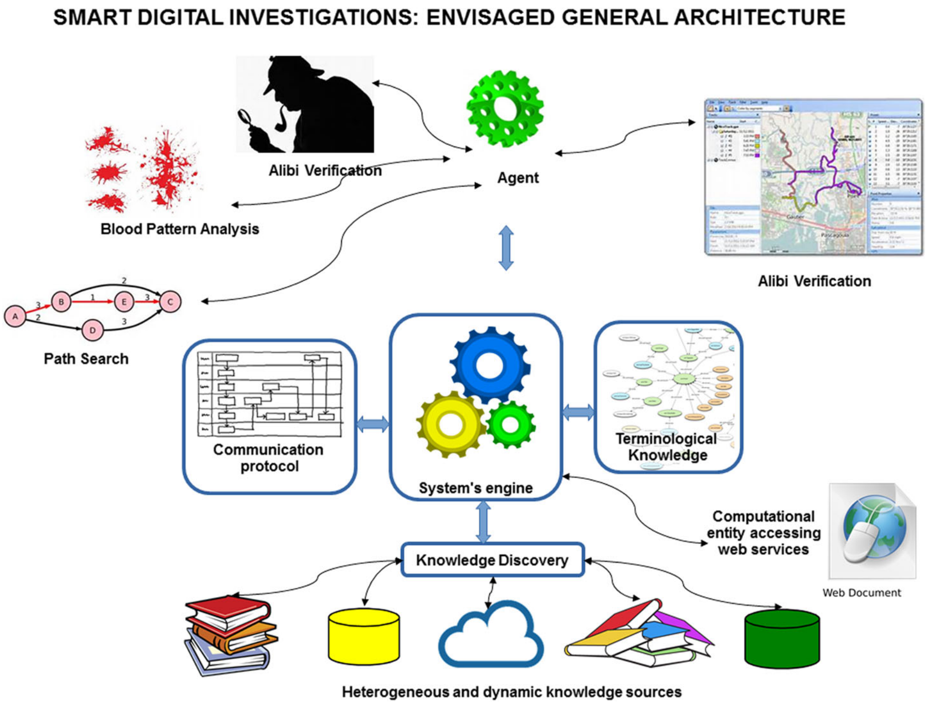

Fig. 1 Envisioned DSS architecture

a graph, where suitable graph reasoning may reconstruct the possible patterns of action of the murderer; alibi verification in the sense of a check of the GPS positions of suspects, so as to ascertain the possibility of her/him being present on the crime scene at the crime time; alibi verification in the sense of double-checking the suspect's declarations with digital data such as computer logs, videos from video-cameras situated on the suspect's path, etc. All the above can be integrated with further evidence such as the results of DNA analysis and others. The system can also include Complex Event Processing modules so as to infer from significant clues the possibility that a crime is being or will be perpetrated. For the system's engine, we plan to explore the usability of Multi-Context Systems (MCS) [7, 8, 10] and of their agent-oriented extensions such as DACMACS (Data-Aware Commitment-based managed Multi-Agent-Context Systems, [18, 20]) and ACEs (Agent Computational Environments, [17]). Thus, in this paper we intend to demonstrate the suitability of automated reasoning, and of ASP in particular, for realizing the reasoning module to be exploited in such a prospective system.

This work does not intend to be exhaustive on the covered topics. Rather, the main objective has been to create a link between the scientific and legal "worlds" to foster further studies, analysis and research aimed at creating *Intelligent Agents* and *Intelligent Systems* [29], supporting the decisions of *judges, lawyers, police forces, criminologists, government offices, etc.*. Notice that, according to current legislation they can merely be **auxiliary tools, and not substitutes, to the human decision-maker**. So, the results of Evidence Analysis provide elements to be considered in trials and do not **"solve the case"** as a naive interpretation might suggest. We must remark that, according to the current Italian legislation and to European directives, such a comprehensive system would be a "*Decision Support System*", that is, it might support a judge in her/his decision making process but **not act as a substitute**.

Concerning related work, a wide corpus of literature about the applications of Artificial Intelligence (AI) in the legal field exists. The reader may refer to the Springer Journal "*Artificial Intelligence and Law*" for a wide perspective. Some papers treat the cognitive and psychological aspects, and the ethical and social implications of the application of Artificial Intelligence in the legal field. Such issues are of great importance, and are indirectly related to this research, concerning the acceptability of the developed techniques by the involved parties. The technical papers however mostly concern the formalization of norms, and the automation of reasoning on formalized norms, and the construction and use of expert systems. A survey (in Italian) about existing Expert Systems for legal reasoning is [44]. An interesting line of work concerns the representation of norms in terms of modal and deontic logics, so as to be able to represent obligations and permissions, and to model legal responsibility. The reader may refer to [33, 55, 56] and to the Ph.D. thesis (in Italian) [32]. Therefore, the line of work that we propose is novel in this area of research. In a hypothetical workflow, existing work would be situated "*a posteriori*" with respect to the content of the present work; in fact, we propose a procedure for generating investigation hypothesis from available data. The aforementioned systems might then be used for evaluating such hypothesis and reaching a solution according to the normative corpus.

The paper is organized as follows. After an overview (Section 2) on *Digital Forensics* (DF) and *Digital Investigation* (DI), in Section 3 we briefly outline the proposed approach. In Section 4 we provide a basic introduction to Answer Set Programming (cf., e.g., [9] for a more complete though synthetic introduction to ASP, accessible also to non-experts readers). In Sections 5, 6, 7, 8, 9 and 10, we will examine the possible solution of some realistic investigative cases via ASP, through a *reduction* to well-known optimization problems. We have tried to make the paper readable by the non-expert, including readers from the legal field; this is why we have explicitly included, for all examples, the commented ASP code (even when it is simple and well-known) and the results returned by ASP solvers, and we have made the code self-contained so that it can be re-run by the interested reader.

2 Digital forensics and digital investigation

The *Digital Forensics* (DF)[1] is the discipline that deals in the legal field of the study, identification, conservation, protection, extraction, documentation and any other form of treatments of *digital data* [14]. The aim is that, after being transformed into an information content by applying suitable forms of reasoning, the data can be assessed as part of a criminal or civil trial.

The DF is thus a branch of *criminalistics* whose activities are aimed at highlighting the existence of *digital evidence* relevant to the fulfilment of a trial, or even before to the investigative phases. It is the process that uses science and technology to analyse digital objects that developing, and tests, theories can also be used legally to answer questions about events that have happened [13]. DF has quickly carved out an ample space also in business environments, aiming to highlight the violations within the industrial secret and/or breach of the security policies and to deal with cyber incidents.

Digital Investigation (DI) involves investigation procedures in which there is the need to develop and test hypotheses to answer questions about *digital events*, in order to understand whether something illegal is happening or is likely to happen. In fact, the operation phase of the DI concerns a time-frame when the crimes or facts are still ongoing and the aim is exactly to identify them.

Although the two disciplines are distinct, their operation is not necessarily performed in sequence, i.e., the latter upon completion of the former. Rather their activities may overlap, and similar techniques can be used.

DF and DI have some peculiarities with respect to other forensic sciences, among which the fact that they cannot be considered "comparative sciences". This is a difference with respect to forensics concerning, e.g., *biology*, *chemistry* or *ballistics*. DF & DI are characterized by constantly evolving and changing techniques, according to the fast process of innovation in the ICT sector, involving analysis methods, instruments, tools, protocols, and structural components relative to the digital systems being analyzed.

The DF and DI approaches are not confined to the obvious cyber-crime spectaculars which capture media attention, but are applied also in many other penal, civil and industrial contexts.

2.1 Sectors of digital forensics and digital investigation

Depending on the area of investigation, DF and DI can be seen as divided into subcategories, specialized in particular fields of technology. The following are the main fields:

[1]The *forensic* term is understood as the science that studies the value in a case of certain events to the formation of possible evidence.

- Computer Forensics: is the “oldest” and best scientifically settled sector, and covers the activities of analysis of the contents of storage devices, such as HDD, SSD, CD, DVD, Flash Memory, etc.;
- Mobile Forensics: it relates to the activities of analysis of the contents of the memory of *mobile* devices, such as mobile phones, smart-phones, tablets, satellite navigator systems, etc.;
- Network & Internet Forensics: it covers the activities of analysis of network systems (including the Internet) and involves the collection, storage and analysis of network events, aimed to the identification of data sources;
- Live Forensics: covers the activities of analysis of the contents of RAM and running systems that cannot be turned off;
- Embedded Forensics: concerns the analysis of “embedded” systems including triggers explosives, actuators, detectors, functional analysis of electronic systems that are unknown in features and functionality;
- Cloud Forensics: concerns the analysis of distributed hardware and software resources, affordable and accessible via the Internet;
- Videos & Multimedia Forensics: concerns the analysis of multimedia files such as audio, video and images aimed at finding traces of alteration and/or to research hidden contents.

2.2 Phases of digital forensics and investigation

The DF and DI, as any other forensic discipline, have to follow rigorous methodologies and procedures whose main stages are reported in Fig. 2.

- Identification: is the phase aimed to research, document and justify all devices (digital or not) relevant to the investigations, with the aim to establish which among them are capable, at least theoretically, to store any reliable information relating to the investigative case;
- Acquisition: is the activity directed to evidence collection, i.e., acquisition of digital and other data that are relevant or might have some connection with a crime; procedures must be carried on without altering the data or the system, or at least so as to minimize the impact;
- Preservation: the technical activity of preserving both logical and physical digital evidence so as to ensure admissibility and resilience w.r.t. possible disputes in trial, offering the reliability required by law for all trial phases;
- Analysis: is the set of operations performed in a scientific context with methodical and demonstrable procedures, directed to produce elements that confirm or disprove an accusatory or defensive hypothesis;

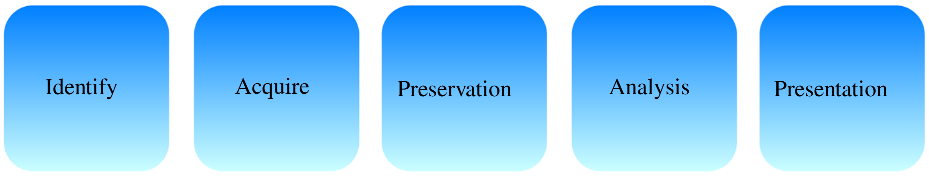

Fig. 2 Main Phases of DF and DI

– Presentation: is the phase aimed to document the activities and results of the single phases, performed through formal reports.

The goal of the analysis phase is precisely to identify pieces of evidence extracted from the digital evidence, organize them, and make them *robust* for their next transformation into *proof* to be presented in *trial*. The constant and ever increasing development of electronic and digital technologies is rapidly bringing the analysis phase of DF to a *breakdown*. This phase is conventionally conducted and performed on the digital evidence sample (or on its forensic copies) with forensic tools currently available on the various platforms, where both proprietary and open-source software co-exist; although it is composed of ordered sequences of activities, with rigorous and formal procedures, it is based on consolidated conventional paradigms that tends to be obsolete while challenging the complexity of current technology.

Thus, the difficulties are discharged on the human operator not avoiding her inevitable limitations, such as the responsibility of managing heterogeneous mass of data, often regarding events, actions, facts and/or sequences of them, so as to produce investigative hypothesis. In the phases of analysis and evidence evaluation the operator should in fact construct and evaluate alternative scenarios. Such scenarios should be constructed from the examination of knowledge that is incomplete, fragmented, or referring to different points of view. This requires in general the aggregation of heterogeneous knowledge sources and may involve time, uncertainty, causality, fortuity or randomness. The operator should also consider collateral though very relevant issues such as the demonstration of the awareness of a digital action or the conscious possession of certain data, for which it is necessary to consider the context.

The experience gained in the field of the DF, as well as the observation of the evolution of the complexity of these analyzes occurred in the last decades, allows us to affirm that more and more frequently:

– Technical assessments carried out on large amounts of data or digital evidence, although providing a comprehensive response from the technical profile, are insufficient in terms of investigative procedure or decision-making for judges, and sometimes are even unusable, for the reasons mentioned above.
– Evaluation and decision, requiring an interpretation to allow a formulation of hypotheses are charged exclusively to the intellectual effort and skills of the investigator and her team. This methodology does not guarantee unequivocal and unambiguous results, because the outcome is necessarily connected to the perception, sensitivity, culture, experience and interpretation of the individual operator in charge. This is in practical and ethical terms unacceptable, as even too often episodes occur where the same evidence, analysed from different human operators, leads to different conclusions.

Currently, no single established procedure exists for Evidence Analysis, which is usually performed by Scientific Investigation experts on the basis of their experience and intuition. We intend instead to promote formal and verifiable AI, Knowledge Representation (KR) and Automated Reasoning methods and techniques for Evidence Analysis. In summary, relevant aspects to consider include at least the following.

– Timing of events and actions.
– Possible causal correlations.
– Contexts in which suspicious actions occurred.
– Skills of the involved suspects.

– Awareness of the involved suspects of committing a violation or a crime and awareness of the degree of severity of the violation/crime.

Moreover, given available evidence, all possibilities of interpretation should be identified, examined and evaluated.

3 Proposed approach

The aim of this research is that all the above-mentioned activities should be performed via techniques that are verifiable with respect to: (i) the results that they provide; (ii) how such results are generated, and (iii) how the results can be explained. In the future, such software tools should be reliable and provide a high level of assurance, in the sense of confidence in the system's correct behaviour. Otherwise, there remains an undesirable uncertainty about the outcome of these stages, and different technicians analysing the same case can reach different conclusions which may lead to different judgements in court.

In AI and Automated Reasoning, several methods and techniques have been developed over the years for uncertain, causal and temporal reasoning, and for devising and examining alternative consistent scenarios that might be compatible with a set of known facts. To the best of our knowledge, these techniques are rarely applied to Digital Forensics and in particular to Evidence Analysis. Therefore, studying their applicability for future development of suitable prototypes is "per se" a significant advance over the state of the art. Moreover, the application to such a challenging field will foster refinements and improvements of the known methods and techniques, and development of novel ones.

In the phase of Crime Identification or detection, the exploration of big data and the application of machine learning techniques can be useful. Instead, the phase of Evidence Analysis has particular requirements that make our proposal based upon KR and Automated Reasoning a much more promising approach, potentially becoming a breakthrough in the state-of-the-art. The ultimate goal of Evidence Analysis is in fact the formulation of verifiable evidence that can be rationally presented in a trial. Under this perspective, results provided by machine learning classifiers or other types of "black box" recommender systems do not have more value than human witness' suspicions and cannot be used as a 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. This has the benefit that the correctness of such declarative systems based on computational logic can be formally verified. Moreover, recent research has led to new methods for visualizing 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.

In summary, the rationale for the choice of Computational Logic relies on the fact that, by its very nature, it is based on precise formalizations, and thus allows for the affordable verification of desired properties of the systems that will be devised in the future as a follow-up of the proposed Action. Verifiability, reliability and justifiability are key features for software tools to be applied in a field such as Digital Forensics, where the evidence produced is aimed at the reconstruction of crimes and establishing innocence or guilt.

For our first experiments that should pave the way to the wider picture that we envision we have chosen Answer Set Programming because it can express combinatorial and planning problems in an easy and natural way, and because ASP formulations are understandable also by the non-expert. The ASP code can be in fact translated or, more properly, transliterated into natural language much more easily than code written in imperative programming languages. Another reason is exactly that ASP formulations exist for many problems in both $\mathbb{P}$ and $\mathbb{NP}$ complexity classes and, due to well-established programming extensions, even higher on the polynomial hierarchy.

In the following sections we illustrate some sample investigative cases in which the use of logic programming, and in particular of Answer Set Programming, allows one to formulate actual investigative hypothesis from the case's description provided in terms of an ASP program solving some known combinatorial problem, where actual data about the case given as facts in the program (problem's instance).

More generally, we propose a methodology to exploit ASP in DF and DI. We assume that a tool-kit of ASP programs which solve puzzles related to relevant and often-occurring investigation problems is available. We also assume that such tool-kit is enriched in time: whenever no already-included ASP piece of code matches the problem at hand, a new one will be either found or developed. In fact, DF and DI activities are carried on in Laboratories by a team involving Computer Science and ICT experts. In view of our approach, the team will involve personnel which is expert in the applications of ASP and in ASP programs development. The proposed methodology is organized into the following steps:

1 Description of the given investigative case, and partition into independent fragments.
2 Identification of a problem (typically in the $\mathbb{NP}$ complexity class) whose features fit as best as possible a fragment of the case at hand. This can be repeated in relation to several independent fragments of a case.
3 For each fragment:
    - 3.1 Definition of a *reduction* from the case's elements to the problem's elements.
    - 3.2 Formalization of the known facts concerning the actual case.
    - 3.3 Identification of an ASP formulation of the problem, in two possible ways:
        - (i) by adapting a solution already present in the tool-kit;
        - (ii) if the former way fails, by enriching the tool-kit via a new solution, tailored to the problem at hand.
    - 3.4 Execution of an ASP solver to find the answer sets.
    - 3.5 Interpretation of the answer sets as possible scenarios for the case fragment at hand.
4 Integration of results obtained for the case fragments. Formulation of investigation hypotheses that can constitute pieces of evidence to be provided to the judge and discussed in Court. This possibly includes transliteration of ASP code and of answer set contents in terms understandable by lawyers, witnesses and jury.

Notice that the skill of the investigators is however required to identify possible correspondences of (parts of) the given case with a known computational problem and thus with one element of the ASP puzzles tool-kit. Moreover, reductions cannot be *proved* correct, at least not in a formal way. The proposed methodology is however a great advance over the state of the art. Nowadays in fact, all steps must be performed by human investigators and (some of the) scenarios that might correspond to solutions of the case are formed purely on the base of experience and intuitions. So, the identified scenarios can be partial or incomplete, and different experts may come to divergent interpretations. Defining a correspondence with well-known problems with provably correct solutions that can be computed via publicly available tools, whose results can be made public and can be explained constitutes a great advancement and a safeguard for all the involved parties. In perspective, the reduction process should be, if not fully automated, at least supported by suitable tools. At present, step 2 is probably the most critical. The difficulty can be alleviated by constructing over time a library of annotated past cases from which those more similar/analogous to the one at hand can be extracted The corresponding formulation can then be suitably adapted.

To improve the methodology, we believe that a *Case Specification Framework* should be defined, to describe each given case in a formal (or at least semi-formal) and organized way. In fact, at present a case description consists of an unstructured set of pieces of natural language text, pictures, graphics, tables, etc. Such definition effort goes beyond the scope of this paper, however it will constitute subject of future work. The envisaged framework should be human-friendly and should provide an environment for generating an integrated picture of each given case. It should thus allow, with methods similar to those of Software Engineering, to assess the quality of a specification and to check the reduction performed in step 3. Coverage of all elements of the case might for instance be verified.

4 Answer set programming (ASP): an overview

Answer Set Programming (ASP) [30, 31], is a highly declarative and expressive programming language oriented towards difficult search problems. It has been used in a wide variety of applications in different areas, like problem solving, configuration, information integration, security analysis, agent systems, semantic web, and planning (for more information about ASP and its application the reader can refer, among many, [4, 6, 28, 40, 57] and the references therein).

Answer set programming has emerged from interaction between two lines of research: first on the semantics of negation in logic programming [30, 31], and second on applications of satisfiability solvers to search problems [36]. It was (independently) identified as a new programming paradigm in [42, 46, 48]. In ASP, search problems are reduced to computing answer sets (or “stable models” as they were originally denominated), and an answer set solver (i.e., a program for generating stable models) is used to find solutions (if any exists) [3].

The expressiveness of ASP, the readability of its code and the performance of the available solvers gained ASP an important role in the field of artificial intelligence; in particular, it is increasingly adopted to encode problems of planning, diagnosis and, more generally, combinatorial problems in knowledge representation and automated reasoning; applications have been developed in several fields, as seen below. In the answer set semantics [30, 31], a (logic) program Π is a collection of *rules* of the form: $H \leftarrow L_1, \ldots, L_n$

The left-hand side and the right-hand side of rules are called *head* and *body*, respectively, where H is an atom, $n \geqslant 0$ and each literal L_i is either an atom A_i or its *default negation not* A_i. A rule can be rephrased as $H \leftarrow A_1, \ldots, A_m, not\ A_{m+1}, \ldots, not\ A_n$ where $A_1, \ldots, A_m$ can be called *positive body* and $not\ A_{m+1}, \ldots, not\ A_n$ can be called *negative body*. In practical programming, '$\leftarrow$' is usually indicated as ':-'. A rule with empty body ($n = 0$) is called a *unit rule*, or *fact*. A rule with empty head, of the form $\leftarrow L_1, \ldots, L_n$, is a *constraint*, and states that literals $L_1, \ldots, L_n$ cannot be simultaneously true in any answer set.

In the ASP literature, it is customary to implicitly refer to the “ground” version of answer set program (ASP program) Π, which is obtained by replacing in all possible ways the variables occurring in Π with the constants occurring in Π itself, and is thus composed of ground atoms, i.e., atoms which contain no variables. The answer sets semantics approach [30, 31] considers a logic program as a set of inference rules (more precisely, default inference rules), or, equivalently, a set of constraints on the solution of a problem: each answer set represents a solution compatible with the constraints expressed by the program. Consider simple program $\{q \leftarrow not\ p \quad p \leftarrow not\ q\}$. For instance, the first rule is read as "assuming that p is false, we can *conclude* that q is true." This program has two answer sets. In the first one, q is true while p is false; in the second one, p is true while q is false. So, negation in this paradigm is understood as *default negation*, i.e., *not a* indicates the (default) assumption that a is false and thus its negation holds. Unlike other semantics, a program may have several answer sets or may have no answer set, where however every answer set is a minimal model of given program,[2] and thus the answer sets form an anti-chain. Whenever a program has no answer sets (as some minimal models may not be answer sets), the program is said to be *inconsistent*. For instance, the following program has no answer set: $\{a \leftarrow not\ b \ \ b \leftarrow not\ c \ \ c \leftarrow not\ a.\}$. The reason is that in every minimal model of this program there is a true atom that depends (in the program) on the negation of another true atom, which is strictly forbidden in this semantics, where every answer set can be considered as a self-consistent and self-supporting set of consequences of a given program. The program $\{p \leftarrow not\ p\}$ has no answer sets either as it is contradictory not in the sense of classical logic, but in the sense that p cannot be derived from *not p*, i.e., from the (default) assumption that p itself is false. Constraints of the form defined above can be in fact simulated by plain rules of the form $p \leftarrow not\ p, L_1, \ldots, L_n$. where p is a fresh atom. Consistency of an ASP program is related (as discussed at length in [15, 16, 19]) to the occurrence of "odd cycles" (of which $p \leftarrow not\ p$ is the basic case, though odd cycles may involve any odd number of atoms) and to their connections to other parts of the program. The reason is that, in principle, the negation *not A* of atom A is an assumption, that must be dropped whenever A can be proved, as answer sets are by definition non-contradictory. Many extensions of the original "Answer Set Prolog" have been proposed, mainly motivated by applications. Some of them are "syntactic sugar" and some other strictly adds expressiveness to the language. For details about these extensions, the reader can refer to [28, 29] and to references therein.

In the ASP paradigm, as mentioned each answer set is seen as a solution of given problem, encoded as an ASP program (or, better, the solution is extracted from an answer set by ignoring irrelevant details and possibly re-organizing the presentation). So, differently from traditional logic programming, the solutions of a problem are not obtained through substitutions of variables values in answer to a query. Rather, a program Π describes a problem, of which its answer sets represent the possible solutions. To find these solutions, an ASP-solver is used. Several solvers have become available, see [3], each of them being characterized by its own prominent valuable features. Recently "meta-solvers" such as ME-ASP (multi-engine ASP system) [45] have been developed, which select the most appropriate solver according to specific syntactic features of given program. The expressive power of ASP and its computational complexity have been deeply investigated (cf. e.g., [21]).

In the following section we provide an example of ASP program, with specific attention to its possible use for the formalization of investigative cases.

[2]By "minimal model of program Π" we mean a minimal model of Π intended as a classical first-order logic theory, where $\leftarrow$ is intended as implication, the comma as conjunction, and *not* as negation in classical logic terms.

5 The zebra escaped

The case of *escaped zebra* is a well-known academic problem, presented in many textbooks on declarative programming, which aims to demonstrate the potential of this form of programming. At the same time, in its simplicity this problem can also help to demonstrate the feasibility of the reduction of investigative cases to combinatorial puzzles. Thus, by reporting the description of this problem, reported from [24], we intend to: (i) illustrate ASP to the non-expert via a pertinent example; (ii) highlight the possibility of application of logic programming, and particularly of Answer Set Programming in investigative cases.

In fact, this problem can be seen as a prototype of the many investigation cases where it is possible to identify two sets of relevant data collected at different times:

– the first dataset is general, and has usually been collected in the immediacy of the fact, most probably during the acquisition of the criminal complaints and/or during the initial inspection; this dataset allows the investigators to outline the elements of the case in terms of logic;
– the other data, subsequently acquired with the progress of the investigations, can be used to reduce the circle of suspects in relation to the offence.

In this section the investigative case is purely hypothetical; in the next sections we will instead treat (skeletons of) real cases.

5.1 The investigative case

As the problem is formulated in the textbook, there are five houses, each of them occupied by different tenants. On the main street, a zebra in total freedom was found. Question:

Who let the zebra escape?

In terms of real investigations for analogy the question might instead concern, for instance, who let a wanted person escape, or who was the accomplice of house robbers, or who sold the drugs to children, etc.

Given the investigative question, sufficient evidence must be collected in order to solve the case. In the present toy formulation, suppose that an initial investigation carried out in the immediacy of the fact made it possible collect further general data, listed below:

– each house has a different color: red, green, white, blue, yellow;
– each tenant comes from a different country: Japan, England, Norway, Italy, and Spain;
– each tenant owns an animal: horse, cat, zebra, fox, and dog;
– each tenant usually drinks one of the following beverages: water, coffee, tea, milk, wine;
– each tenant has one car: Fiat, Lancia, BMW, Toyota, Audi.

5.2 ASP solution

Based on the first acquired data, an ASP encoding involves to define the domain of houses, each of them represented by an integer from 1 to 5, proceeding from left to right:

```
house(1..5).
```

For other entities we use constants corresponding to the name of the entity (in lower-case). We can starts from the following ASP *facts*. The first ones define which colors and which nations we consider. We then proceed to list the animals that can be involved. Then we list the drinks. Finally we provide a list of cars. Notice that we do not list *all* possible colors, nations, cars, etc. Rather, we concentrate on those which are mentioned in the specification of the problem that we try to solve.

```
color(red). color(green). color(white). color(blue). color(yellow).
nation(japan). nation(england). nation(norway). nation(italy).
nation(spain).
animal(horse). animal(cat). animal(zebra). animal(fox). animal(dog).
drink(water). drink(coffee). drink(tea). drink(milk). drink(wine).
car(fiat). car(lancia). car(bmw). car(toyota). car(audi).
```

We define below the concept of proximity between the houses, by means of ASP *rules*:

```
on_right(X,X+1) :- house(X), house(X+1).
on_left(X+1,X) :- house(X), house(X+1).

near(X,Y) :- on_right(X,Y).
near(X,Y) :- on_left(X,Y).
```

At this point, via the rules below we characterize prospective solutions so that each entity (color, home, car, etc.) has a single type, and that this is associated with only one of them.

```
other_color(House,Color) :-
        house(House), color(Color), color(C), have_color(House,C),
C!=Color.
other_color(House,Color) :-
       house(House), color(Color), house(C1), have_color(H1,Color),
       H1!=House.
have_color(House,Color) :-
       house(House), color(Color), not other_color(House,Color).

other_car(House,Car) :-
       house(House), car(Car), car(A), have_car(House,A), A!=Car.
other_car(House,Car) :-
       house(House), car(Car), house(C1), have_car(C1,Car),
       C1!=House.
have_car(House,Car) :-
       house(House), car(Car), not other_car(House,Car).
other_drink(House,Drink) :-
       house(House), drink(Drink), drink(B), have_drink(House,B),
B!=Drink.
other_drink(House,Drink) :-
       house(House), drink(Drink), house(C1), have_drink(C1,Drink),
       C1!=House.
have_drink(House,Drink) :-
       house(House), drink(Drink), not other_drink(House,Drink).

other_nation(House,Nation) :-
       house(House), nation(Nation), nation(N), have_nation(House,N),
       N!=Nation.
other_nation(House,Nation) :-
```

```
        house(House), nation(Nation), house(C1),
        have_nation(C1,Nation),

        C1!=House.
have_nation(House,Nation) :-
        house(House), nation(Nation), not other_nation(House,Nation).

other_animal(House,Animal) :-
        house(House), animal(Animal), animal(A), have_animal(House,A),
        A!=Animal.
other_animal(House,Animal) :-
        house(House), animal(Animal), house(C1),
        have_animal(C1,Animal),

        C1!=House.
have_animal(House,Animal) :-
        house(House), animal(Animal), not other_animal(House,Animal).
```

5.3 More information available

The above formulation is able to generate a high number of solution, equal to $(n!)^x$, where n is the number of houses, and x is the number of entities considered (color, car, etc.). In our case, $n = 5$ and the solutions amounts to $(5!)^5 = 24.883.200.000$.

Assume however that at this point, as a result of further investigations, the investigators have been identified additional elements such as:

(i) the British tenant lives in a red house;
(ii) the Spanish tenant owns the dog;
(ii) the Norwegian tenant lives in the first house on the left;
(iv) in the garage of the yellow house, there's a Toyota;
(v) the BMW driver lives in the house next to those have the fox;
(vi) the Norwegian lives in a house next to the blue house;
(vii) who owns the Lancia also owns the cat;
(viii) the Fiat driver drinks wine;
(ix) the Italian tenant drinks tea;
(x) the Japanese tenant drives the Audi;
(xi) Toyota is parked in the garage of a house next to the house where there is the horse;
(xii) in the green house people drink coffee;
(xiii) the green house is immediately to the right of the white house;
(xiv) the milk is drunk in the house in the middle (the third one);

By adding suitable ASP *constraints*, the number of *answer sets* of the program will be drastically reduced. In fact, the ASP programming style is widely based on a *generate-and-test* methodology. Below we define an ASP rule for each constraint. Then, the additional rule with head *constraints* states that the conjunction of all constraints must hold. Finally, the rule with empty head states that it is not possible that constraints **do not** hold. In fact, connective :- at the beginning of a rule means *it cannot be that* ...

```
i :- house(C), have_nation(C,england), have_color(C,red).
ii :- house(C), have_animal(C,dog), have_nation(C,spain).
iii :- have_nation(1,norway).
```

```
iv :- house(C), have_car(C,toyota), have_color(C,yellow).
v :- house(C), house(C1), near(C,C1), have_car(C,bmw),
have_animal(C1,fox).
vi :- house(C), house(C1), near(C,C1), have_color(C,blue),
      have_nation(C1,norway).
vii :- house(C), have_car(C,lancia), have_animal(C,cat).
vii :- house(C), have_car(C,fiat), have_drink(C,wine).
ix :- house(C), have_nation(C,italy), have_drink(C,tea).
x :- house(C), have_nation(C,japan), have_car(C,audi).
xi :- house(C), house(C1), near(C,C1), have_car(C,toyota),
       have_animal(C1,horse).
xii :- house(C), have_color(C,green), have_drink(C,coffee).
xiii :- house(C), house(C1), have_color(C,white),
       have_color(C1,green),
       on_right(C,C1).
xiv :- have_drink(3,milk).
constraints :- i, ii, iii, iv , v, vi, vii, viii, ix, x, xi,
xii, xiii, xiv.
:- not constraints.
:
% The following are directives to the solver
% to state what to show or not when displaying answer sets
% hide everything unless the predicates which are required by #show
#hide.
#show have_nation(C,D).
#show have_animal(C,D).
#show have_car(C,D).
#show have_drink(C,D).
#show have_color(C,D).
```

5.4 Results

By running an ASP Solver (any of the available ones is equally usable), the output will be:

```
Answer: 1
Stable Model: have_color(1,yellow) have_color(2,blue)
have_color(4,white)
have_color(5,green) have_color(3,red) have_drink(4,wine)
have_drink(3,milk) have_drink(2,tea) have_drink(5,coffee)
have_drink(1,water) have_car(5,audi) have_car(1,toyota)
have_car(2,bmw)
have_car(3,lancia) have_car(4,fiat) have_animal(4,dog)
have_animal(1,fox)
have_animal(5,zebra) have_animal(3,cat) have_animal(2,horse)
have_nation(4,spain) have_nation(2,italy) have_nation(1,norway)
have_nation(3,england) have_nation(5,japan)
```

The following graphical representation highlights the solution in boldface (Fig. 3):

house	*1*	*2*	*3*	*4*	**5**
color	yellow	blue	red	white	**green**
drink	water	tea	milk	wine	**coffee**
car	toyota	bmw	lancia	fiat	**audi**
animal	fox	horse	cat	dog	**zebra**
nation	norway	italy	england	spain	**japan**

Fig. 3 Solution of 'The Zebra Escaped' puzzle

In this example we have made the simplifying assumption that all collected items of information are clearly related and non-contradictory. In reality, investigations may provide different kinds of information, sometimes incoherent, sometimes apparently contradictory. Therefore, data should be suitably cleansed and pre-processed either by humans or by automated techniques [53] or both, as it is however always the case in problem scenarios that involve data collection. Intrinsic contradiction can partly managed by ASP through the generation of several answer sets outlining possible scenarios, or also by resorting to more advanced Computational Logic techniques.

6 Data recovery and file sharing hypothesis

6.1 The investigative case

The judicial authority requested the Digital Forensics Laboratory to analyse the contents of a hard disk, in order to check for the presence of illegal contents files. If so, they requested to check for potential activities of sharing illegal materials on the Internet.

The hard disk under analysis was physically damaged (as often done by criminals if they expect to be captured soon). Therefore, after a heads replacement, the evidence acquisition phase recovered a large amount of files (of various types: images, videos, documents, etc.), however without their original name because the damage present on the disk plates disallowed the recovery the information of the MFT.[3] For this reason, an arbitrary name is assigned to all the recovered files. Information about the original names of files and about their original location in the file system is thus missing.

Starting from the elements described below, 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 and 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 formulation of the "Stable Marriage Problem", that we describe below.

6.2 Elements

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

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

- files with illegal contents;
- various "INDX files", corresponding in the NTFS file system to directory files, which contain some METADATA such as filename, physical size of file, logical size of file, modified timestamp, accessed timestamp, changed timestamp, created timestamp;
- index related to the file-sharing application (which is a widely-used file-exchange application), including a file containing sharing statistics.

6.3 The marriage problem

The Marriage Problem (or SMP - Stable Marriage Problem) is a well-known $\mathbb{NP}$-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 $\hat{B}$.

6.4 Reduction

The idea of the reduction from an investigative case is to exploit the SMP to find a correspondence between file names and file contents, in analogy to a correspondence between men and women. In order to find a plausible solution it is important to identify a suitable preference criterion. In particular, each file name should "prefer" to be coupled with those files that might indeed correspond to its original content. 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 (with associated properties, such as size, type) 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". For each file name, a preference is created with those file contents which match its properties.

6.5 Answer set programming solution

Once compiled the lists *men*, *women* and *preferences*, one can search for answer sets by means of the following ASP program. 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) :- preference(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).
```

The preference/2 predicate merely expresses possible pairings, but not (numerical) preferences as in the original Stable Marriage problem.

6.6 Results

The results obtained via a solver on the real example are the following:

```
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)
... ...
```

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*,[4] it has been possible to make reasonable assumptions about the effective sharing of files with illegal content.

It is worth noticing that the results returned by the program, showing that in most answer sets (i.e., scenarios) illicit files appear to have been exchanged, constitutes a piece of evidence that a judge will evaluate together with other elements. By considering the suspect's precedents and, for instance, illegal material seized at the suspect's premises, communication records, etc., the judge will decide whether there are elements enough for conviction. Thus, Evidence Analysis does not "solve a case", rather it provides elements to allow a judge to do so (beyond any reasonable doubt).

[4]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.

7 Identification of “Puppeteers”

7.1 The investigative case

After a long and complicated investigation, the police locates a criminal organization. All technological devices in possession of members of this criminal organization were sequestered.

The DF analysis aims to hypothesize who can be the subjects in charge of criminal organization, by examining the records of digital communication made with any means (phone calls, text messages, chat applications, etc.). The objective is to identify *active* subjects, i.e., those subjects who serve as a hub for members of the organization, and that therefore transmit directives and orders to other members or are even the bosses of the organization. It may be supposed that such subjects are those who communicate with various sub-groups of the organization.[5] It is clear that a communication network can be modeled as a graph (cf., e.g., [23]). In fact, there are many approaches to do this, and a large body of research for identifying various forms of influence (neutral or fraudulent) among network members by means of algorithms based upon graph theory.

7.2 Elements

In the case at hand, data were collected as follows:

- by requests to the operators of phone records for seized telephone devices;
- by forensic analysis of seized devices for the exploration and extraction of all conversations made through the various messaging clients (WhatsApp, Telegram, Messenger, Facebook Messenger, etc.).

7.3 Clique problem

The problem of *clique* is a problem considered difficult, as in fact it belongs to the class of $\mathbb{NP}$-*complete* problems. The *clique problem* is defined as follows: given an undirected graph $G = (V, E)$, a clique in G is a subset of vertices $C \subseteq V$, such that every pair of vertices is connected by an edge:

$$\forall u, v \in C \implies (u, v) \in E$$

Essentially, a *clique* is a complete subgraph of the graph G (Fig. 4).

The problem of *clique* is an optimization problem, which can be seen as a decision problem which consists in asking whether there is in the graph G a clique of size k, where k is the size of the involved set of vertexes. For this search, a traditional algorithm performs an enumeration of all subsets of vertices of size k of V to check if each of them forms a clique, with running time of such algorithm in $\Omega(k^2\binom{|G|}{k})$; that is, polynomial with constant k for value of k low. If the value of k if next to $|V|/2$, the algorithm exhibits a super-polynomial running time.

[5]There is the notable exception of Mafia bosses, who even nowadays never use electronic means but rather write their directions on small pieces of papers, called ’pizzini’, to be memorized and then destroyed by their minions

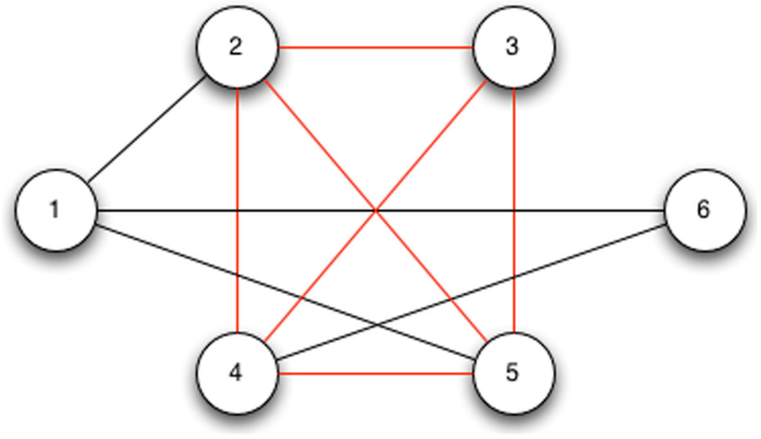

Fig. 4 Example of clique

7.4 Reduction

The reduction here is simple and direct, and as a result the set of conversations obtained from the analysis is transformed into a graph, of the type shown in Fig. 5, where:

- every chat account, phone number, etc., is a node of the graph;
- every conversation is an edge of the graph.

If you need to create a directed graph, you might consider the direction of the conversation to represent the direction of the arcs.

7.5 Answer set programming solution

From an investigative point of view the clique problem is particularly well-suited to this investigation as it is useful to analyse *big data* including phone records, log files, activities of *social network analysis*. The following ASP code[6] searches a graph for the existence of a clique:

```
e(1, 2).
......

v(X) :- e(X,Y).
v(Y) :- e(X,Y).

n {in(X) : v(X)}.

:- in(X), in(Y), v(X), v(Y), X!=Y, not e(X,Y), not e(Y,X).
:
#maximize [in(X) : v(X)].
```

[6]source: Håkan Kjellerstrand

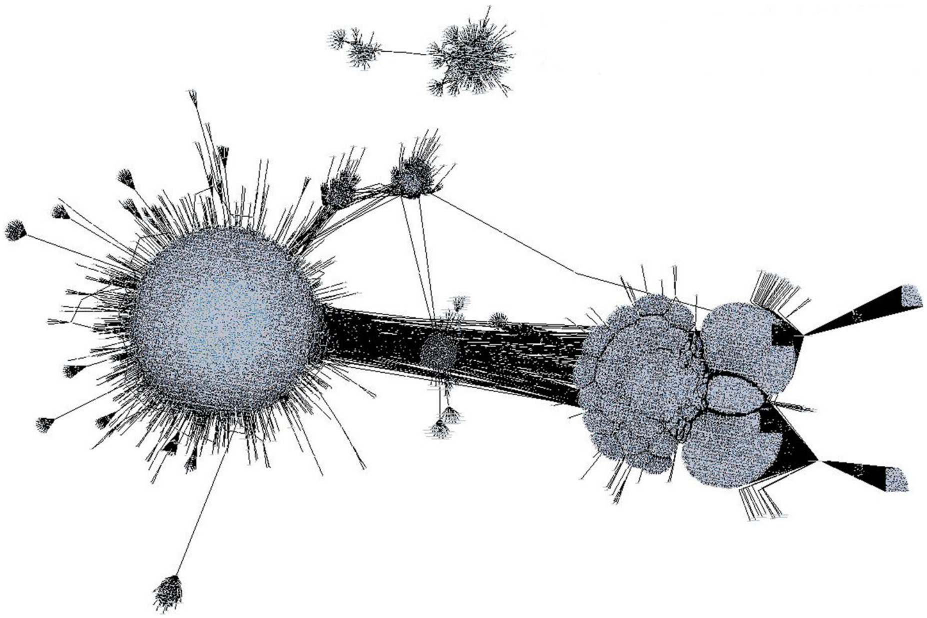

Fig. 5 Typical communication graph

7.6 Results and observations

The result of the execution of the ASP program allows the investigators to highlight the nodes connected to each other that represent sub-groups of the organization (in fact, 'clique' is translated into Italian as 'cricca', that outside graph theory indicates a sinister group of people, presumably adept to obscure common objectives). More importantly, nodes belonging to several sub-groups may presumably indicate the persons acting as representatives for communication aspects of the organization, and for transmitting orders among subgroups.

We do not claim that the ASP solution provides complete knowledge on who is a puppeteer. However, it provides an automation and a systematization of a routine check that is always performed on telephone and social network logs. "Busy" nodes resulting from the solution of the clique problem may provide suggestions on who can be the Puppeteers. This is a pieces of evidence that needs however to be combined with the results of other investigation activities in order to form robust evidence (against some suspect) to be presented and defended in court. Basically, solutions of the puzzle provide the investigators with useful well-founded suggestions for further analysis.

8 Crime scene solver

8.1 The investigative case

After a crime has been committed, the first activity that often gives rise to the investigations is the *crime scene analysis* (inspection), a means for gathering evidence aimed to observation and description of places, to finding and collecting items suitable for the reconstruction

Fig. 6 Crime scene

the facts. Particular importance, during the inspection, are the descriptive, planimetric and photographic survey (Fig. 6).[7] Starting from the evidence and elements location on crime scene, the investigators have to try reconstruct the sequence of events.

8.2 Elements

Investigators can rely upon the data collected on the crime scene, including fingerprints, bloodstains, shoe-prints left on the floor etc. On this basis, they have to establish a hypothetical sequence of actions taken in the crime scene during the crime. So, a first step is to produce a scaled planimetry of the location of the crime scene, where points of interest can be identified and marked, or where the evidence were detected during the inspections (Fig. 7).

Once again, the set of points of interest and their connections can be represented as a graph, which suggests that graph theory can profitably be used to analyse the situation. A first aspect that must be considered is to establish the possible path followed by the culprit during the action.

8.3 Hamiltonian path

The Hamiltonian path problem is defined as follows: given a directed graph G, a path in G is called "Hamiltonian" if it passes through all nodes of G exactly once (Fig. 8). For the graph (G, s, t) in the Fig. 8, the path $s, 1, 2, 3, 4, 5, t$ is a Hamiltonian path.

8.4 Reduction

We can perform the reduction of a relevant part of the investigative case, i.e., hypothesizing a possible 'modus operandi', to the problem of finding an Hamiltonian path on a suitable directed graph. This graph has the aim to identify the possible route covered by the culprit. It is obtained through the analysis of the evidence collected at the crime scene, and marked on the planimetry (Fig. 7) (Fig. 9).

To draw the directed graph we can use in particular the following schema:

[7] Image courtesy from Wired.it

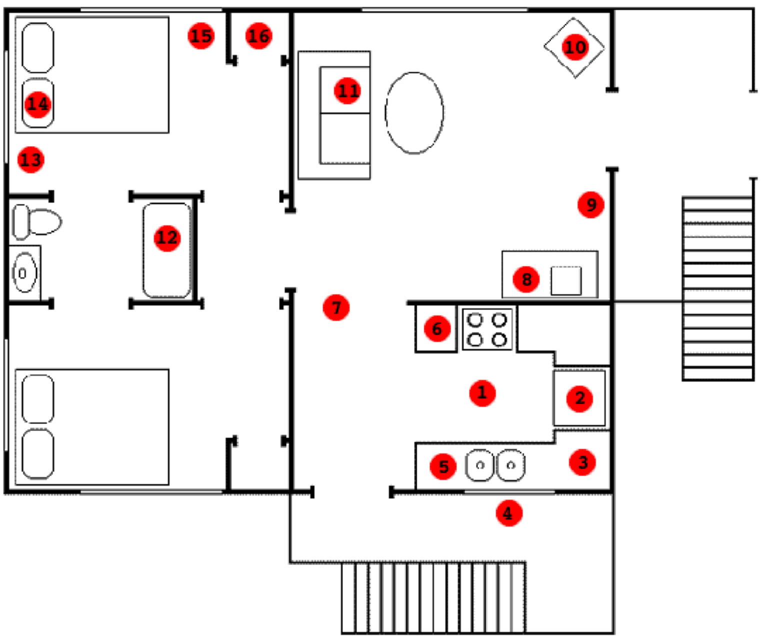

Fig. 7 Crime Scene planimetry with marked points of interest

- vertices correspond to evidences indicated in the planimetry;
- vertices closed to potential "escape routes" are indicated the as initial and final vertices of the searched-for path;
- arcs are inserted between each couple of nodes which are "in sight" of each other;
- the direction of each arc is obtained from:
  - directional trajectory of the evidence (for, example direction of shoe-prints);

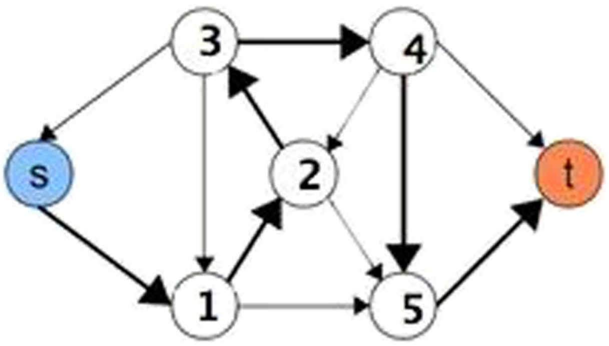

Fig. 8 Hamiltonian path

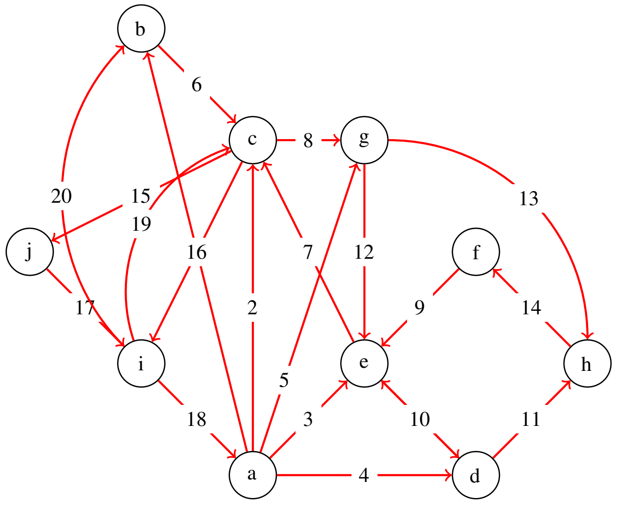

Fig. 9 Crime scene graph

– the study of the bloodstains (through bloodstain pattern analysis,[8] see Fig. 10 below;
– for ambiguous directions we insert arcs in both directions.

8.5 Answer set programming solution

To solve this problem we can use the following ASP code taken from the course notes by Prof. Vladimir Lifschitz, University of Texas at Austin.

```
{in(X,Y)} :- edge(X,Y).

:- 2 {in(X,Y) : edge(X,Y)}, vertex(X).
:- 2 {in(X,Y) : edge(X,Y)}, vertex(Y).
:
r(X) :- in(0,X), vertex(X).
r(Y) :- r(X), in(X,Y), edge(X,Y).

:- not r(X), vertex(X).
```

8.6 Results and possible generalization

The result of the execution of the ASP program allows in general several answer sets to be obtained, corresponding to hypothetical sequences of actions undertaken at the crime scene by criminal and victim, compatible with the release of the tracks identified. To generalize the work, it also possible to create separate graphs, one with the victim tracks, and the others with the criminal(s) tracks, find the answer sets separately and verify whether there are

[8]Bloodstain Pattern Analysis (BPA), is a scientific method of forensic analysis of the morphology of the sketches, blotches or spots of blood at a crime scene.

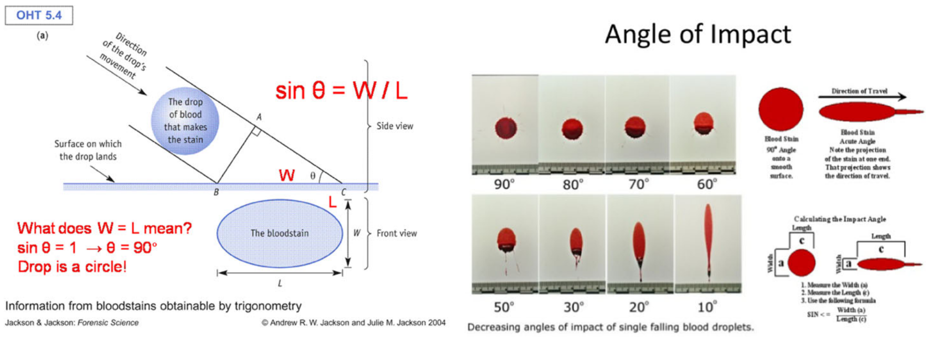
OHT 5.4
(a)
Direction of the drop's movement
The drop of blood that makes the stain
Surface on which the drop lands
sin θ = W / L
Side view
A
B
C
W
θ
L
What does W = L mean?
sin θ = 1 → θ = 90°
Drop is a circle!
The bloodstain
W
Front view
L
Information from bloodstains obtainable by trigonometry
Jackson & Jackson: Forensic Science
© Andrew R. W. Jackson and Julie M. Jackson 2004
Angle of Impact
90°
80°
70°
60°
50°
30°
20°
10°
Decreasing angles of impact of single falling blood droplets.
Blood Stain 90° Angle onto a smooth surface.
Direction of Travel
Blood Stain Acute Angle Note the projection of the stain at one end. That projection shows the direction of travel.
Calculating the Impact Angle
Length
c
Width
a
1. Measure the Width (a)
2. Measure the Length (c)
3. Use the following formula
SIN < = Width (a) / Length (c)

Fig. 10 Blood stains analysis mathematical principles

answer set that matches or are compatible with each other. Also, in case a full Hamiltonian path cannot be found, we can employ *weak constraints*, which is a well-known ASP extension which allows to express "optional" constraints that will be satisfied whenever possible. By this construct one can obtain quasi-Hamiltonian paths, and can even specify the degree of coverage.

9 Path verification

9.1 The investigative case

After a crime, a suspect has been arrested. The police sequestered all his mobile devices (smart-phone, 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.

9.2 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. This is a very common investigative problem, which is proposed virtually everyday to the DF Laboratories for a variety of issues. This problem in fact may concern minor violations ("The speed camera failed, it was not my car at that position at that date, so the fine is ill-placed") but also major crimes such as murder or meetings of criminal organizations. The task of solving the problem in an automated though reliable can be accomplished via reduction to the following simple game.

9.3 Hidato puzzle (Hidoku)

Hidato, from the Hebrew word "Hida" meaning riddle, 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.

9.4 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 as follow:

- in each box corresponding to a GPS position in GPS-LIST the corresponding numeric value has been inserted;
- each box corresponding to a GPS position not in GPS-LIST has been set to a conventional value 0.

Considering the above matrix, assume that the crime has been committed at the cell marked with **0** located between 14, 8 and 5, 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 when long sequences of zero's occur (Fig. 11).

9.5 Answer set programming solution

Once the matrix has been built, we can determine whether a possible route for the suspect to have committed the crime exists; this by finding the answer sets of the following ASP program [37] (here we have used the *clingo* solver). Notice that the omitted cells are assumed to have value 0.

```
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). mtrix(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).
```

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

Fig. 11 Hidato Matrix from GPS-LIST

```
#const n = 6.
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).
```

9.6 Results

The results obtained by running an ASP 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)
```

graphically represented by the matrices shown in Fig. 12.

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

Fig. 12 Matrix Representation of answer sets 1 and 2

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

Finally it can be noted that the above program, through a small variant, allows one to calculate solutions not only in rectangular matrices but also in irregularly shaped to better adapt the program to the complex structure of the maps occurring in real cases.

Notice that the ASP solution does not provide a complete proof about who was present on the crime scene. It however allows to perform reliably a systematic check that must however be done in such circumstances. The solution of the Hidato puzzle says that a telephone (and presumably its holder, but maybe not) might possibly have been at the crime scene at the given time or not, considering its previous and past positions. Other investigation activities will charge or discharge a suspect, and the resulting investigative hypothesis must however be presented and defended in court.

10 Alibi verification

In many investigations of the Criminal Police, the investigators find themselves confronted with the problem of verifying an alibi for one or more suspects. An alibi is made up of precise moments, indications of some places and times well specified and of some places and times which are known within some approximation. Faced with not fully detailed scenarios, it is often not easy for the investigators to verify the perfect sequence of actions. The following is a generic case resolvable via Computational Logic.

10.1 The investigative case

During an investigation concerning a murder, it is necessary to check the alibi provided by a (male) suspect. In the interrogation, the suspect was vague on the timing of his movements, but said:

– 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;
– to have forgotten his smart-phone at the office (place Y).

In order to verify the suspect's alibi, the judicial authority requested the DF laboratory to analyse:

– the contents of the smart-phone owned by the suspect (that has actually been retrieved where indicated);
– 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.

10.2 Elements

The coroner's analysis on the body has established the temporal interval including the time of death. From the forensic analysis of the smart-phone 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 many frames with a male subject in the distance, 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 modelled 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 (cf, e.g., [11] and the references therein). 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 "drosophila" of such kind of problems in Artificial Intelligence.

We may notice that if the suspect has achieved the objective of being at the places he declared to have been and to commit the murder he has, consciously or unconsciously, previously or step-by-step, devised a plan to do so. Therefore, it must be checked if a feasible plan indeed exists encompassing all the certain available elements *and* the crime. ASP is indeed a state-of-the-art tool for planning, and several examples exist of planners written in ASP. However, for the sake of simplicity we will resort to a simple formulation, extrapolated again from a well-known puzzle.

10.3 Monkey & Banana

The problem "*Monkey & Banana*" is a typical planning problem of Artificial Intelligence. 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 also 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.

10.4 Reduction

The case at hand can be reduced to the "*Monkey & Banana*" problem by finding a correspondence between actors and actions of the case and actors and actions of the puzzle. A possible correspondence is listed below. 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

The constraints of the problem are that, at any time, the monkey (and, in the case, the suspect):

– can perform only one action at each time instant;
– if the monkey ascends on the chair, it cannot walk, and it cannot climb further;
– if the chair is not moved then it stays where it is;
– if the chair is moved it changes its position;
– the monkey is somewhere in the room, where it remains unless it walks, which implies changing position;
– 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 (the suspect may have possibly committed the crime only if he had reached the crime place and he opened the door).

10.5 Answer set programming solution

The following ASP program, a modified version from the one that can be found online,[9] 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),
```

[9]at http://www.dbai.tuwien.ac.at/proj/dlv/tutorial

```
   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).
```

10.6 Results

The proposed reduction allows investigators to perform a first though reliable verification of 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 checked for compatibility 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 situation where the suspect is provably at the office at time 0, we get the action sequences by which the suspect might have reached the place where the crime has been perpetrated. For the sake of clarity, below we rewrite such sequences in terms of elements of the case rather than of elements of the puzzle.

```
 {step(0,walk,at_office),
step(1,motion_to_crime_location,at_crime_location),
  step(2,open_door," "), step(3,raise_alarm," ")}
```

Therefore, if the suspect raised the alarm at time 3 he actually had no time for committing the crime 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 "unknown action" provides open possibilities for which it might be interesting to investigate further so as to prove or reject the investigation thesis.

```
{step(0,unknown_action," "),step(1,walk,at_office),
        step(2,motion_to_crime_location,at_crime_location),
step(3,open_door," "),
        step(4,unknown_action," "),step(5,raise_alarm," ")}

{step(0,walk,at_crime_location), step(1,walk,at_office),
        step(2,motion_to_crime_location,at_crime_location),
step(3,open_door," ")
        step(4,unknown_action," "), step(5,raise_alarm," ") }
```

Among the answer sets there are many which suggest suspicious behaviour. The first one above outlines a scenario where the suspect would not have had the time to commit the crime, as he moved to the office and then to the location of the crime where he immediately raised the alarm. In the second one 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 third 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.

11 Conclusions & future work

The challenge of which this paper constitutes just a first step aims to create an infrastructure for the application of Artificial Intelligence (AI) and Automated Reasoning in the Digital Forensics field. In particular, the challenge is to address the Evidence Analysis phase. In this phase, evidence about possible crimes and crime perpetrators collected from various electronic devices (by means of specialized software, and according to specific regulations) must be examined and aggregated so as to reconstruct possible events, event sequences and scenarios related to a crime. Evidence Analysis results are then made available to law enforcement, investigators, intelligence agencies, public prosecutors, lawyers and judges.

The challenge is actually meant for both areas: digital forensics on the one hand and automated reasoning in AI on the other hand. From the AI perspective, this research aims in the long term at the construction of software tools that will require a complex combination of results and techniques from different areas of Knowledge Representation and Reasoning such as diagnosis, causal explanation, temporal reasoning about actions, epistemic reasoning, treatment of incomplete knowledge, deontic and legal reasoning, inductive learning and formal concept analysis, to cite some of the most relevant ones surely involved in the digital forensics activity. On the other hand, the application of (intelligent) automated tools in digital forensics, capable of an exhaustive search for exploring evidence and going much further than the scope of human observation will surely become a breakthrough with an immediate impact in the practical investigation of crime scenarios.

In the medium-long term, we envisage a situation where law enforcement, investigators, intelligence agencies, criminologists, public prosecutors, lawyers and judges will be provided with decision-support-systems that can effectively aid them in their activities. The adoption of such systems can contribute to making legal proceedings clearer and faster, and also under some respects more reliable. Our choice of Computational Logic as a basis has a strong reason: the formality of logic indeed guarantees transparency and verifiability of tools and results. Results can thus be explained and motivated by the system to human users; explainability and accountability are in fact of particular importance in this field.

The main risk concerning the proposed approach and its future developments is in our opinion that it may be to some extent difficult to convince the involved parties and the general public of the real applicability of such systems. While for some forensic techniques, such as DNA analysis, there is nowadays a high and widespread level of trust, an Artificial-Intelligence-Based decision support system may appear unconvincing or even threatening. So, a parallel interdisciplinary challenge of no less importance will be that of transforming scientific concepts such as verifiability and correctness into concepts such as psychological reliability and trust.

References

1. Alberti, M., Gavanelli, M., Lamma, E., Mello, P., Torroni, P.: The Sciff abductive proof-procedure. In: Bandini, S., Manzoni, S. (eds.) AI*IA 2005: Advances in Artificial Intelligence, 9th Congress of the Italian Association for Artificial Intelligence, Proceedings, volume 3673 of Lecture Notes in Computer Science, pp. 135–147. Springer (2005)
2. Alferes, J.J.: Preserving strong equivalence while forgetting. In: Logics in Artificial Intelligence - 14Th European Conference, JELIA 2014, Proceedings, Volume 8761 of Lecture Notes in Computer Science, pp. 412–425. Springer (2014)
3. ASP. Answer set programming solvers online (incomplete list), 2016. http://assat.cs.ust.hk, http://www.cs.utexas.edu/users/tag/ccalc/, http://www.cs.utexas.edu/users/tag/cmodels/, http://www.cs.uky.edu/ai/, http://www.dbai.tuwien.ac.at/proj/dlv, http://www.potassco.org, http://www.tcs.hut.fi/Software/smodels
4. Baral, C.: Knowledge Representation, Reasoning and Declarative Problem Solving. Cambridge University Press, Cambridge (2003)
5. Ben-Ari, M., Manna, Z., Pnueli, A.: The temporal logic of branching time. Acta Informatica **20**, 207–226 (1983)
6. Borchert, P., Anger, C., Schaub, T., Truszczynski, M.: Towards systematic benchmarking in answer set programming the dagstuhl initiative. In: LPNMR, volume 2923 of Lecture Notes in Computer Science, pp. 3–7. Springer (2004)
7. Brewka, G., Eiter, T., Fink, M.: Nonmonotonic multi-context systems: A flexible approach for integrating heterogeneous knowledge sources. In: Balduccini, M., Son, T.C. (eds.) Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning - Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday, volume 6565 of Lecture Notes in Computer Science, pp. 233–258. Springer (2011)
8. Brewka, G., Eiter, T., Fink, M., Weinzierl, A.: Managed multi-context systems. In: Walsh, T. (ed.) IJCAI 2011, Proc. of the 22nd Intl. Joint Conf. on Artificial Intelligence, pp. 786–791. IJCAI/AAAI (2011)
9. Brewka, G., Eiter, T., Truszczynski, M.: Answer set programming at a glance . Commun. ACM **54**(12), 92–103 (2011)
10. Brewka, G., Ellmauthaler, S., Pührer, J.: Multi-context systems for reactive reasoning in dynamic environments. In: Schaub, T. (ed.) ECAI 2014, Proc. of the 21st European Conf. on Artificial Intelligence. IJCAI/AAAI (2014)
11. 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, pp. 102–116. Springer (2012)
12. Cabalar, P., Diéguez, M.: Strong equivalence of non-monotonic temporal theories. In: KR. Citeseer (2014)
13. Casey, E.: Handbook of Digital Forensics and Investigation. Elsevier, Amsterdam (2009)

14. Casey, E.: Digital Evidence and Computer Crime: Forensic Science, Computers, and the Internet. Academic press, Cambridge (2011)
15. Costantini, S.: Contributions to the stable model semantics of logic programs with negation. Theor. Comput. Sci. **149**(2), 231–255 (1995)
16. Costantini, S.: On the existence of stable models of non-stratified logic programs. TPLP **6**(1-2), 169–212 (2006)
17. Costantini, S.: Ace: a flexible environment for complex event processing in logical agents. In: Baresi, L., Baldoni, M., Dastani, M. (eds.) Engineering Multi-Agent Systems, Third International Workshop, EMAS 2015, Revised Selected Papers, volume 9318 of Lecture Notes in Computer Science, Springer (2015)
18. Costantini, S.: Knowledge acquisition via non-monotonic reasoning in distributed heterogeneous environments. In: Truszczyński, M., Ianni, G., Calimeri, F. (eds.) 13th Int. Conf. on Logic Programming and Nonmonotonic Reasoning LPNMR 2013. Proc., volume 9345 of Lecture Notes in Computer Science Springer (2015)
19. Costantini, S., D'Antona, O.M., Provetti, A.: On the equivalence and range of applicability of graph-based representations of logic programs. Inf. Process. Lett. **84**(5), 241–249 (2002)
20. Costantini, S., DeGasperis, G.: Exchanging data and ontological definitions in multi-agent-contexts systems. In: Paschke, A., Fodor, P., Giurca, A., Kliegr, T. (eds.) RuleMLChallenge track, Proceedings, CEUR Workshop Proceedings. CEUR-WS.org (2015)
21. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001)
22. Delgrande, J.P., Wang, K.: Proceedings of the twenty-ninth AAAI conference on artificial intelligence, january 25-30, 2015, austin, texas, USA. In: Bonet, B., Koenig, S. (eds.) Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence, pp. 1482–1488. AAAI Press (2015)
23. Deswal, S., Singhrova, A.: Application of graph theory in communication networks. International Journal of Application or Innovation in Engineering and Management 1 (2012)
24. Dovier, A., Formisano, A.: Programmazione Dichiarativa in Prolog, CLP, ASP, e CCP. free, 2008. Available (in Italian) at http://users.dimi.uniud.it/agostino.dovier/DID/corsi.html
25. Eiter, T., Wang, K.: Semantic forgetting in answer set programming. Artif Intell. **172**(14), 1644–1672 (2008)
26. Emerson, E.A.: Temporal and Modal Logic. InL: Handbook of theoretical computer science, vol. B. MIT Press (1990)
27. Erdem, E., Gelfond, M., Leone, N.: Applications of answer set programming. AI Mag. **37**(3), 53–68 (2016)
28. Gelfond, M.: Answer sets. In: Handbook of Knowledge Representation. Chapter 7, pp. 285–316. Elsevier, Amsterdam (2007)
29. Gelfond, M., Kahl, Y.: Knowledge Representation, Reasoning, and the Design of Intelligent Agents: The Answer-Set Programming Approach. Cambridge University Press, New York (2014)
30. 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, pp. 1070–1080. MIT Press (1988)
31. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. N. Gener. Comput. **9**, 365–385 (1991)
32. Governatori, G.: Un modello formale per il ragionamento giuridico. PhD thesis, Dottorato Di Ricerca in Informatica Giuridica E Diritto Dell'informatica - CIRFID Università Di Bologna, 1996. Supervisors: Professors Alberto Artosi and Maurizio Matteuzzi
33. Governatori, G., Olivieri, F., Rotolo, A., Scannapieco, S.: Computing strong and weak permissions in defeasible logic. J. Philos. Log. **42**(6), 799–829 (2013)
34. Kakas, A.C., Kowalski, R.A., Toni, F.: Abductive logic programming. J. Log. Comput. **2**(6), 719–770 (1992)
35. Kakas, A.C., Toni, F.: Computing argumentation in logic programming. J. Log Comput. **9**(4), 515–562 (1999)
36. Kautz, H.A., Selman, B., et al.: Planning as satisfiability. In: ECAI, vol. 92, pp. 359–363. Citeseer (1992)
37. Kjellerstrand, H.: Hidato. Available at http://www.hakank.org/answer_set_programming (2015)
38. Kowalski, R.A., Toni, F.: Abstract argumentation. Artif. Intell. Law **4**(3-4), 275–296 (1996)
39. Koymans, R.: Specifying real-time properties with metric temporal logic. Real-Time Systems **2**(4), 255–299 (1990)
40. Leone, N.: Logic programming and nonmonotonic reasoning: From theory to systems and applications. In: Baral, C., Brewka, G., Schlipf, J. (eds.) Logic Programming and Nonmonotonic Reasoning, 9th Intl. Conference, LPNMR 2007. Springer (2007)

41. Lifschitz, V., Pearce, D., Valverde, A.: Strongly equivalent logic programs. ACM Trans. Comput. Log. **2**, 526–541 (2001)
42. Lifschitz, V.: Action languages, answer sets, and planning. In: The Logic Programming Paradigm, pp. 357–373. Springer (1999)
43. Lifschitz, V.: Twelve definitions of a stable model. In: de la Banda, M.G., Pontelli, E. (eds.) Proc. of the 24th Intl. Conference on Logic Programming, volume 5366 of LNCS, pp. 37–51. Springer (2008)
44. Lucatuorto, P.L.M.: Intelligenza artificiale e diritto: le applicazioni giuridiche dei sistemi esperti. Ciberspazio e Diritto **7**(2), 103–125 (2006)
45. Maratea, M., Pulina, L., Ricca, F.: Multi-engine ASP solving with policy adaptation. J. Log. Comput. **25**(6), 1285–1306 (2015)
46. Marek, V.W., Truszczyński, M.: Stable models and an alternative logic programming paradigm. In: The Logic Programming Paradigm, pp. 375–398. Springer (1999)
47. Montali, M., Chesani, F., Gavanelli, M., Lamma, E., Mello, P.: Evaluating compliance: from LTL to abductive logic programming. In: Ancona, D., Maratea, M., Mascardi, V. (eds.) Proceedings of the 30th Italian Conference on Computational Logic, volume 1459 of CEUR Workshop Proceedings, pp. 101–116. CEUR-WS.org (2015)
48. Niemelä, I.: Logic programs with stable model semantics as a constraint programming paradigm. Ann. Math. Artif. Intell., Springer **25**(3-4), 241–273 (1999)
49. Olivieri, R.: Digital Forensics meets Complexity Theory and Artificial Intelligence: Towards Automated Generation of Investigation Hypothesis. PhD thesis, Dottorato di ricerca in Ingegneria e Scienze dell'Informazione, Università degli Studi dell'Aquila. Supervisor: Prof. Stefania Costantini. (2016)
50. Pearce, D.: A new logical characterization of stable models and answer sets. In: Non-Monotonic Extensions of Logic Programming, Number 1216 in LNAI, pp. 55–70. Springer (1997)
51. Pearce, D., Valverde, A.: Synonymous theories in answer set programming and equilibrium logic. In: Proc. of ECAI04, 16th Europ. Conf. on Art. Intell. pp 388–390 (2004)
52. Pnueli, A.: The temporal logic of programs. In: Proc. Of FOCS, 18th Annual Symposium on Foundations of Computer Science, pp. 46–57. IEEE (1977)
53. Rahm, E., Do, H.H.: Data cleaning: Problems and current approaches. IEEE Data Eng. Bull. **23**(4), 3–13 (2000)
54. Riguzzi, F., Bellodi, E., Zese, R., Cota, G., Lamma, E.: A survey of lifted inference approaches for probabilistic logic programming under the distribution semantics. Int. J. Approx. Reason. **80**, 313–333 (2017)
55. Rotolo, A., Governatori, G., Sartor, G.: Deontic defeasible reasoning in legal interpretation: two options for modelling interpretive arguments. In: Proceedings of the 15th International Conference on Artificial Intelligence and Law, ICAIL 2015, June 8-12, 2015, pp. 99–108. San Diego (2015)
56. Smith, C., Calardo, E., Rotolo, A., Sartor, G.: Legal responsibility for the acts of others a logical analysis. In: Rules on the Web. From Theory to Applications - 8Th International Symposium, RuleML 2014, Co-Located with the 21St European Conference on Artificial Intelligence, ECAI 2014, August 18-20, 2014. Proceedings, pp. 329–338, Prague (2014)
57. Truszczyński, M.: Logic programming for knowledge representation. In: Dahl, V., Niemelä, I. (eds.) Logic Programming, 23rd Intl. Conference, ICLP 2007, pp. 76–88. Springer, Berlin (2007)
58. Vardi, M.Y.: Branching vs. linear time: Final showdown. In: Proceedings of the 2001 Conf. on Tools and Algorithms for the Construction and Analysis of Systems, TACAS 2001, number 2031 in LNCS, pp. 1–22. Springer (2001)