UNIVERSIDADE DE ÉVORA

Digital Forensics Using Constraint Programming

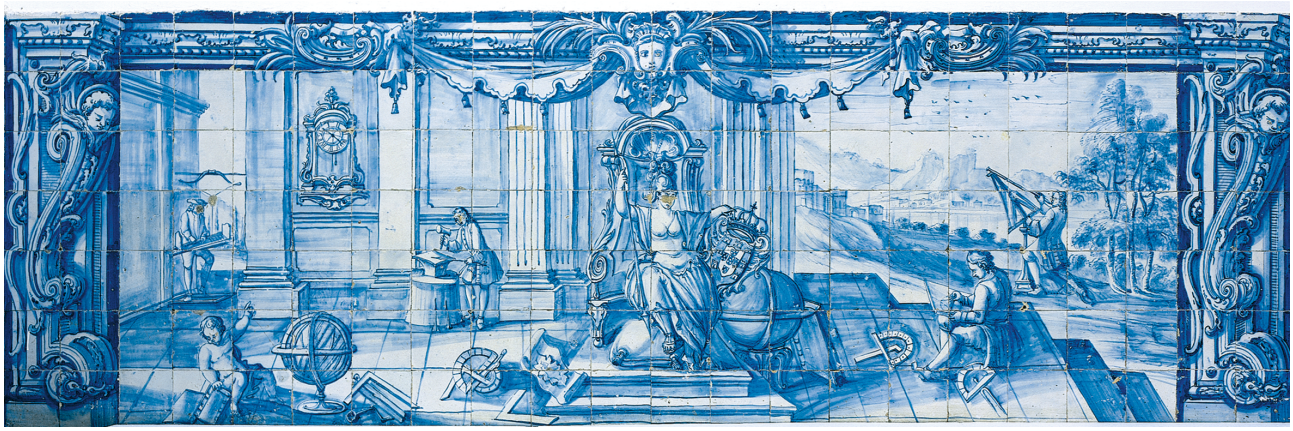
João Calhau

Tese apresentada à Universidade de Évora
para obtenção do Grau de Mestre em Engenharia Informática

Orientador *Pedro Salgueiro*
Orientador *Salvador Abreu*
Orientador *Nuno Goes*

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri

19 de Julho de 2018



INSTITUTO DE INVESTIGAÇÃO E FORMAÇÃO AVANÇADA

UNIVERSIDADE DE ÉVORA

Escola de Ciências e Tecnologia

Departamento de Informática

Digital Forensics Using Constraint Programming

João Calhau

Orientação *Pedro Salgueiro*
Salvador Abreu
Nuno Goes

Engenharia Informática

Dissertação

19 de Julho de 2018

Esta dissertação não inclui as críticas e sugestões feitas pelo Júri

Nothing yet.

Prefácio

Nothing yet.

Agradecimentos

Nothing yet.

Contents

List of Figures

List of Tables

Acronyms

CSP Constraint Satisfaction Problem

Abstact

Nothing yet.

Keywords: Digital Forensics, Constraint Programming, Security, Declarative Programming

Sumário

Forense Digital Utilizando Programação por Restrições

Nothing Yet.

Palavras chave: Forense Digital, Programação por Restrições, Segurança, Programação Declarativa

1

Introduction and Motivation

1.1 Introduction

Digital forensics is a complex task that consists in analyzing large amounts of data, that can come from the most various digital sources, with a multitude of different tools.

In this work we present a system that makes use of the constraint programming paradigm and methods to solve digital forensics problems. Through the use of constraint programming, we are able to describe a digital forensics problem in a declarative and expressive way and efficiently find a solution to such problem, if it exists: a set of files or other resources that match the description of the digital forensic problem.

The main goal of the system presented in paper is to allow for an easy and efficient method to search for relevant information in the contents of digital equipment.

1.2 Constraint Programming

Constraints can be found in our day to day experiences, almost ubiquitous, representing the conditions that restrict our freedom of decision. Constraint programming is a powerful paradigm mostly used to solve combi-

natorial problems. We can consider it as a simple way to model real world problems but it can actually turn into a complex challenge when we want to find solutions for the problem that is being solved [**?**].

1.3 Digital Forensics

Digital forensics is a very important discipline in criminal investigations, where relevant evidences are stored in digital devices. The digital forensics tools have become a vital tool to ensure we can rebuild information after a cyberattack or even if we just want to analyze any type of digital equipment [**?**].

1.4 Digital Forensics and Constraint Programming

1.5 Experimental Evaluation

2

Constraint Programming

2.1 Introduction

Constraint Satisfaction, in it's most basic form, consists in finding a value for each one of a set of problems where constraints specify the restrictions related to those problems.

Constraint Satisfaction Problems have been tackled by the most various methods, from automata theory to ant algorithms and are a topic of interest in many fields of computer science [**?**].

2.2 History of Constraint Programming

Before understanding what constraint programming is we need to understand that Constraint Satisfaction is the process of finding a solution to a set of constraints that impose conditions that the variables must satisfy [**?**]. A solution is found when a set of variables satisfies all constrains. These constraints and variables will be explained in deeper detail later in section 2.3.

Constraint Satisfaction originally appeared in the field of artificial intelligence in the early 1970s. During the 1980s and 1990s constraints started being embedded into programming languages, as they were being devel-

oped, and the term constraint programing started taking form. Examples of these programming languages are *Prolog* and *C++* [**?**].

In artificial intelligence interest in constraint satisfaction developed into two streams, the language stream and the algorithm stream. The first stream refers to the side of Constraint Satisfaction that deals with algebraic equations, constraint statements and declarative languages, while the second stream refers to actual algorithms used to solve said constrains [**?**].

2.3 Working with Constraint Programming

There are several definitions we have to delve into to better understand how constraint programming works. We need to fully understand what a domain and a variable is and we also need to know what a constraint is.

2.3.1 Variable

A variable has several known definitions, the one we want is in the field of mathematics. According to mathematics a variable is a symbol, usually an alphabetic character, and it represents a number, known as the value of the variable, which can be arbitrary, not fully specified or unknown [**?**].

2.3.2 Domain

A domain, like the variable, also has various known definitions, however the one that interests us here is in the field of mathematical analysis.

According to Hans Hahn, an open set is connected if it cannot be expressed as the sum of two open sets. An open connected set is called a domain [**?**]

In our case, a domain is an open connected set of variables.

2.4 Approaches to Constraint Programming

In constraint programming, we usually have a set of variables, which takes values from an initial domain, to which constraints are applied in order to reduce its domain, and thus reach a solution. Once a constraint is placed on the system it cannot violate another constraint previously applied. This way, we can express the requirements of the possible values of the variables [**?**].

Constraint satisfaction problems are typically solved with the help of solvers. These solvers are essentially search algorithms, usually based on backtracking techniques[**?**], constraint propagation [**?**] or local search [**?**].

2.4.1 Backtracking

Backtracking is a search method that incrementally finds possible candidates to solve the problem. At the same time it removes the candidates that can not be used as a valid solution to the problem [**?**]. One of the most used examples for this type of search method is the n-queens puzzle, where a set of n queens should be organized, in a $n \times n$ chess board, in such a way that none of the queens can attack each other. Any partial solution that

contains two queens that can attack each other is abandoned immediately.

2.4.2 Constraint Propagation

Constraint propagation starts by reducing the variable's domain, strengthening or creating new constraints, reducing the search space, leading to a problem that is easier to solve. Since this algorithm only reduces the search space of the problem variables. After completion, there is still the need to use another algorithm to solve the problem, which is now converted into a simpler problem by the propagators [**?**].

2.4.3 Local Search

Local search is an incomplete search method to find solutions for a problem. It consists in, iteratively, and with the help of previously defined heuristics, assigning values to the problem variables until all the constraints are satisfied. At each step of the iteration, the values of the variables are updated to values *near* the previous value. The algorithm also knows the cost associated with the assignment of specific values to variables, allowing to check if a candidate solution has with a pre-defined cost [**?**].

2.5 Known Tools and Other Approaches

2.5.1 Choco

Choco is an open source and free access library dedicated to constraint programming. It is written in Java and supports several types of variables, including Integers, Booleans, Sets and Reals. It also supports several types of constraints such as AllDifferent and Count, configurable search algorithms and conflict explaining. The first version of Choco was developed in the early 2000s. A few years later, Choco 2 was developed and declared a success in the academic and industrial world. Since then, Choco has been completely re-written and in 2012 the third version of Choco was launched. The current version comes with a simpler API and is denominated Choco 4 [**?**].

2.5.2 Gecode

Gecode is a free access, open source, portable, accessible and efficient programming environment used to develop systems and applications based on restrictions. Gecode, much like Choco, supports various types of variables and restrictions, among them are Integers, Float and Sets. These variables are used to model problems that are then solved with the help of constraint propagators and search algorithms [**?**] [**?**].

2.5.3 Google OR-Tools

Although Choco and Gecode are two of the most widely used libraries, there are also other new tools, such as the Google OR-Tools [**?**]. Google Optimization Tools or OR-Tools is an interface that puts together several linear programming solver and that counts on the use of several types of algorithms such as search algorithms and graph algorithms. What this library has that is so noteworthy is the fact that it doesn't let itself be bound by one language. Although implemented in C++, it can be used in other languages like Python, C# or Java.

3

Digital Forensics

3.1 Introduction

It is a complex task to collect evidence/elements in digital equipment, either connected to a criminal activity or not. If that piece of equipment is connected to any type of computer network, with the consequent increase in digital traffic, more difficult it becomes to detect any anomaly or undesirable communication in the network. Thus, the intrusion detection systems have become a very important tool in computer network security [**?**].

To collect evidence or data in digital equipment, there are many tools capable of analyzing a digital forensics image. Among them, one of the tools that is most used is the *EnCase Forensic* [**?**], is a proprietary and commercial tool, used in many judicial systems. There are also widely used open source and free access tools capable of accomplishing the same work. The *Forensic ToolKit (FTK)* [**?**] and the *Autopsy* [**?**] are two of the most used.

3.2 History of Digital Forensics

3.3 Known Tools and other Approaches

In this section we introduce the practical aspects of constraint programming including some libraries and toolkits that are used to model and solve constraint problems. We also describe similar work already done in this area.

3.3.1 EnCase

3.3.2 Forensic ToolKit (FTK)

3.3.3 The Sleuth Kit and Autopsy

The Sleuth Kit is a C library and a collection of tools that allows to analyze disc images and restore files from it. The Sleuth Kit is what Autopsy [**?**], the forensics tool mentioned earlier, uses in it's background jobs. The Sleuth Kit framework allows the user to incorporate additional modules so he can analyze file contents and build automated systems. In addition, the library can be embedded in larger digital forensics tools and command line tools can be used directly to find any kind of proof [**?**].

Of all the tools The Sleuth Kit has to offer, the most interesting to help us in the type of problem we're trying to solve is the Sorter, which analyzes a file system and organizes what it finds by extension of file. In addition, it provides us details about the organized files, such as the file Inode number. The Sorter can also use a separate hash database to ignore files that are known to be good, such as Dynamic-Link Libraries, or dlls, of the windows file system or even know applications.

4

Our Approach

4.1 Introduction

This section introduces our approach for modeling a Digital Forensics Problem as a Constraint Satisfaction Problem and reaching a valid solution using the Choco Solver. We describe how we modelled the problem as a CSP, the methodologies used to analyze and extract the information from the digital evidences, the data structures used to model the problem and how they are used to reach a solution.

4.2 Methodology

After acquiring the disk image to be analyzed, it is first processed with the help of tools from The Sleuth Kit [**?**], first with Sorter and then with Mactime. Sorter creates multiple files with different names, each name being a pre-determined type of file, such as archive, executable or data. As for Mactime, it creates a single file containing time information about every file present in the file system. Examples of these files can be seen in Figure 4.1 and Figure 4.2.

The files created by Sorter include all relevant information about the files present in the file system that is being analyzed, including: file path in the file system, the file type, the image name (from where the data was extracted)

```
idle_master/CSteamworks.dll
  PE32 executable (DLL) (GUI) Intel 80386, for MS Windows
  Image: pen_4_dd.dd  Inode: 40-128-1

idle_master/HtmlAgilityPack.dll
  PE32 executable (DLL) (console) Intel 80386 Mono/.Net assembly, for MS Windows
  Image: pen_4_dd.dd  Inode: 42-128-1

idle_master/IdleMaster.exe
  PE32 executable (GUI) Intel 80386 Mono/.Net assembly, for MS Windows
  Image: pen_4_dd.dd  Inode: 43-128-1
```

Figure 4.1: Example of Sorter output for executable files

```
Date,Size,Type,Mode,UID,GID,Meta,File Name
Wed Jan 14 2015 19:52:12,134656,m...,r/rrwxrwxrwx,0,0,42-128-1,"/idle_master/HtmlAgilityPack.dll"
Wed Jan 14 2015 19:52:12,26,m...,r/rrwxrwxrwx,0,0,42-128-3,"/idle_master/HtmlAgilityPack.dll:Zone.Identifier"
Mon Feb 09 2015 18:28:22,513536,m...,r/rrwxrwxrwx,0,0,46-128-1,"/idle_master/Newtonsoft.Json.dll"
Mon Feb 09 2015 18:28:22,26,m...,r/rrwxrwxrwx,0,0,46-128-3,"/idle_master/Newtonsoft.Json.dll:Zone.Identifier"
Tue Feb 10 2015 21:49:22,116736,m...,r/rrwxrwxrwx,0,0,40-128-1,"/idle_master/CSteamworks.dll"
Tue Feb 10 2015 21:49:22,26,m...,r/rrwxrwxrwx,0,0,40-128-3,"/idle_master/CSteamworks.dll:Zone.Identifier"
Tue Feb 10 2015 21:49:22,142528,m...,r/rrwxrwxrwx,0,0,51-128-1,"/idle_master/steam_api.dll"
Tue Feb 10 2015 21:49:22,26,m...,r/rrwxrwxrwx,0,0,51-128-3,"/idle_master/steam_api.dll:Zone.Identifier"
Fri Jul 24 2015 17:41:46,207360,m...,r/rrwxrwxrwx,0,0,50-128-1,"/idle_master/Steamworks.NET.dll"
```

Figure 4.2: Example of Mactime output for a 4GB pen drive

and the Inode number, which is an internal representation of that particular file in the file system. As for the Mactime file, it only contains date information about each file. This information is parsed into a database that allows the data to be persistent. This database is described in detail in Section 4.5. The data also passes through a caching system that tries to determine if the exact type of constraints have been applied to the file system being analyzed to find if it can skip the lengthily process of trying to find a solution. The caching system is described in detail in Section 4.6.

The whole program can be shortened into the small flow diagram seen in figure 4.3.

After pre-processing the disk image and persisting all necessary data, the digital forensics problem, which describes the items that being looked for, is modeled as a Constraint Satisfaction Problem (CSP). This CSP is then solved by a CSP solver, to reach a solution that satisfies the initial digital forensics problem, if it exists. A solution to such a CSP will be a set of file identifiers that match the digital forensics problem. It is necessary to use specific constraints to reach the solution mentioned before, which are used to restrict the domain of the variables. These constraints are described in Section **??**.

4.3 Modeling

The problem is modeled as a set of variables that represent the files that need to be found, according to the digital forensics problem. These files are represented as numerical values, which are the Inode numbers extracted from the Sorter mentioned previously. Each of these variables are associated with a domain and a set of constraints which are applied to the variables. As previously stated these constraints were created specifically in the context of this work.

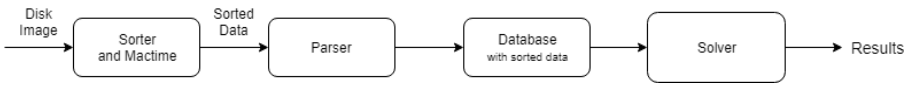


Figure 4.3: Flow diagram

In this work we use the Choco Solver, which allows the use of four different types of variables to model the problems: Integers (IntVar), Booleans (BoolVar), Sets (SetVar) and Reals (RealVar). To model a digital forensics problem as a CSP, we decided to use Set variables (SetVars).

In Choco Solver, SetVars are defined by a domain that is made of two separate domains, the Lower Bound (LB) and the Upper Bound (UB). The Lower Bound is a set of integers that must belong to every solution, while the Upper Bound is composed of the set of integers that may be part of the final solution [**?**]. In our case, when creating the variable with which we are going to work with, the Lower Bound will be left empty, because we do not know what files we want yet. As for the Upper Bound, it will be composed of all the Inodes found during the parsing phase.

Listing 1 Modelling of a Digital Forensics CSP

$$CSP = (V, D, C)$$

$$V = \{V_1, V_2, \ldots, V_n\}$$

$$D = \{D_1, D_2, \ldots, D_n\}, \quad \forall D_i \in D : D_i = \{Y_1, Y_2, \ldots, Y_z\}$$

$$C = \{C_1(V_i, \ldots, V_j), \ldots, C_k(V_i, \ldots, V_j)\},$$
$$\forall C_k \in C : C_k = \{CP_1(V_i, \ldots, V_j), \ldots, CP_z(V_i, \ldots, V_j)\}$$

Listing 1 presents a formal representation of a digital forensics CSP, represented by the triple $CSP = (V, D, C)$. V is the set of variables, which represents the files to be found, D is the set of domains for each variable which, where each $D_i = \{Y_1, Y_2, \ldots, Y_z\}$ is a set of integer values that represent the Inodes present in the file system; and C is the set of constraints which restricts the domain of each variable, where each C_k is mapped into a specific constraint propagator.

4.4 Constraints

To reach a solution, Choco solver makes use of constraints which are implemented as propagators. A propagator declares a filtering algorithm that can be applied to the variables that models the problem, in order to reduce their domain [**?**]. In the context of this work, we implemented the following propagators:

1. **File type:** restricts our domain according to the given type of file passed as argument.
2. **File path:** restricts our domain according to the given path passed as argument.
3. **Word search:** restricts our domain according to the given word passed as argument.
4. **NIST:** restricts our domain according to the NIST database.
5. **Date:** restricts our domain according to the given date passed as argument.

All these propagators were implemented to work with the variables used to model the problem: SetVars.

To create the propagators, we had to implement the following methods: `propagate` and `isEntailed`. The method `propagate` should restrict the domain according to what we need, while the method `isEntailed` informs the propagator if a problem has a solution or not, or if it is not possible to determine if a solution exists. These methods are described in detail in Listings 2 and 3.

Listing 2 propagate method

```
for each value in UB do
    if value does not belong to desired structure then
        Remove value from UB
    end if
end for
```

Listing 3 isEntailed method

```
if UB is empty then
    Problem is impossible to solve
else
    Problem has possible solution
end if
```

4.4.1 Type, Path and Date Propagators

The Type propagator was the first propagator created and it takes the Upper Bound of the SetVar, iterates over it and removes any Inode that does not belong to the type we are looking for. Path and Date propagators work in the same way. The main difference is what they are trying to restrict: the Type propagator receives the type of file we want to restrict and iterates over the Upper Bound to remove every Inode that does not belong to that type of file, while the Path propagator receives a path and, similarly to the Type propagator, iterates over the Upper Bound and removes every Inode that does not belong to that path.

4.4.2 Word Search Propagator

This propagator relies on the Unix4j [**?**] Java library. This library implements most of the Linux bash commands in native Java. We use this library since it provides efficient methods to find contents inside a file in any file system.

This propagator works in a similar way to the ones previously described. It iterates over the Upper Bound and removes any file that does not contain the word we passed as argument. Also, since this is the last constraint ever being run, we add the Inodes that passed through the propagator to the Lower Bound, because at this moment we're certain these will belong to the final solution.

4.4.3 NIST Propagator

The NIST National Software Reference Library, or NSRL for short, is a hash database containing all the signatures of traceable software applications. We use this database to figure out if we have files in our file system that are known to be safe, so we don't have to analyze them. We use of a tool that belongs to The Sleuth Kit [**?**] called `hfind`, that looks for a given hash in the database. If hash is found, we remove the Inode belonging to that hash from the domain.

4.5 Data Persistence

As mentioned before, we decided to use a database to persist parsed data. This database is created at the time of parsing the data. If the data has not been parsed before, a new database is created for the data being parsed.

The database contains five things: 1) the file id (the Inode number); 2) the file name; 3) the file path; 4) the file type and 5) the file date.

4.6 Caching

This caching system was created to lessen the time it takes for the system to solve all the constraint problems it was commissioned to solve by searching in a data structure that contains all the results of previous constraint problems, including the ones that have no results. Every time the system is initiated, it checks if what we're trying to solve has been solved before, and if so, it skips the whole constraint process and just gives us the results we want, if not it runs like normal and saves the results at the end. This applies to all combination of constraints.

5

Experimental Evaluation

5.1 Introduction

5.2 Experimental Results

To evaluate the system, we used device images from different sources that claim their images are for the testing of forensics computer tools, Digital Corpora [**?**] and Linux LEO [**?**]. The images taken from Digital Corpora are two, one from a Canon camera, code named ”canon2” and the other a bootable USB disk with a ext3 Ubuntu file system installed, code named ”casper”. From Linux LEO one image was taken, an Able2 Ext2 disk image, code named ”able2”.

The evaluation results are presented in Table 5.1, which describes the elapsed time to reach a solution in seconds, the total number of files present in the device, number of Inodes found that match the modeled Digital Forensics problem, the constraints used to model the problem and if the caching system was used, for each experiment.

Device Name	Times (sec.)	Total Inodes	Found Inodes	Constraints	Caching
Canon2	1.6	38	33	type, path	no
Canon2	1.0	38	33	type, path	yes
Canon2	1.8	38	33	type, path, date	no
Canon2	0.6	38	33	type, path, date	yes
Canon2	1.4	38	33	type, path, date, word	no
Canon2	0.6	38	33	type, path, date, word	yes
Casper	2.6	1079	4	type, path	no
Casper	0.5	1079	4	type, path	yes
Casper	2.7	1079	4	type, path, date	no
Casper	0.7	1079	4	type, path, date	yes
Casper	2.7	1079	4	type, path, date, word	no
Casper	0.7	1079	4	type, path, date, word	yes
Able2 Partition 3	88.2	11653	12	type, path	no
Able2 Partition 3	0.7	11653	12	type, path	yes
Able2 Partition 3	66.4	11653	12	type, path, date	no
Able2 Partition 3	1.1	11653	12	type, path, date	yes
Able2 Partition 3	61.8	11653	12	type, path, date, word	no
Able2 Partition 3	1.1	11653	12	type, path, date, word	yes

Table 5.1: Experimental Results

6

Conclusion and Future Work

UNIVERSIDADE DE ÉVORA
INSTITUTO DE INVESTIGAÇÃO
E FORMAÇÃO AVANÇADA

Contactos:
Universidade de Évora
Instituto de Investigação e Formação Avançada — IIFA
Palácio do Vimioso | Largo Marquês de Marialva, Apart. 94
7002 - 554 Évora | Portugal
Tel: (+351) 266 706 581
Fax: (+351) 266 744 677
email: iifa@uevora.pt