UNIVERSITÀ DEGLI STUDI DI TRIESTE
Sede Amministrativa del Dottorato di Ricerca

XXV Ciclo - Scuola di Dottorato in
Ingegneria dell'Informazione

Machine Learning Techniques for Document Processing and Web Security

(Settore scientifico-disciplinare ING-INF/05)

DOTTORANDO
Enrico Sorio

RESPONSABILE DOTTORATO DI RICERCA
Chiar.mo Prof. **Walter Ukovich**
Università degli Studi di Trieste

RELATORE
Chiar.mo Prof. **Alberto Bartoli**
Università degli Studi di Trieste

CORRELATORE
Chiar.mo Prof. **Eric Medvet**
Università degli Studi di Trieste

Anno Accademico 2011/2012

Contents

Abstract

The task of extracting structured information from documents that are unstructured or whose structure is unknown is of uttermost importance in many application domains, e.g., office automation, knowledge management, machine-to-machine interactions. In practice, this information extraction task can be automated only to a very limited extent or subject to strong assumptions and constraints on the execution environment.

In this thesis work I will present several novel application of machine learning techniques aimed at extending the scope and opportunities for automation of information extraction from documents of different types, ranging from printed invoices to structured XML documents, to potentially malicious documents exposed on the web.

The main results of this thesis consist in the design, development and experimental evaluation of a system for information extraction from printed documents. My approach is designed for scenarios in which the set of possible documents layouts is unknown and may evolve over time. The system uses the layout information to define layout-specific extraction rules that can be used to extract information from a document. As far as I know, this is the first information extraction system that is able to detect if the document under analysis has an unseen layout and hence needs new extraction rules. In such case, it uses a probability based machine learning algorithm in order to build those extraction rules using just the document under analysis. Another novel contribution of our system is that it continuously exploits the feedback from human operators in order to improve its extraction ability.

I investigate a method for the automatic detection and correction of OCR errors. The algorithm uses domain-knowledge about possible misrecognition of characters and about the type of the extracted information to propose and validate corrections.

I propose a system for the automatic generation of regular expression for text-extraction tasks. The system is based on genetic programming

and uses a set of user-provided labelled examples to drive the evolutionary search for a regular expression suitable for the specified task.

As regards information extraction from structured document, I present an approach, based on genetic programming, for schema synthesis starting from a set of XML sample documents. The tool takes as input one or more XML documents and automatically produces a schema, in DTD language, which describes the structure of the input documents.

Finally I will move to the web security. I attempt to assess the ability of Italian public administrations to be in full control of the respective web sites. Moreover, I developed a technique for the detection of certain types of fraudulent intrusions that are becoming of practical interest on a large scale.

Riassunto

L'estrazione di informazioni strutturate da documenti non strutturati, o di cui non si conosce la struttura, è di estrema importanza in molti campi, ad esempio l'office automation, la gestione della conoscenza, le interazioni machine-to-machine. Nella pratica, l'estrazione di informazioni può essere automatizzata solo in misura molto limitata e in presenza di notevoli assunzioni e vincoli riguardo l'ambito applicativo.

In questo lavoro di tesi vengono presentate diverse applicazioni innovative di tecniche di machine learning con l'obbiettivo di automatizzare l'estrazione di informazioni da documenti di vario tipo. Questi documenti possono variare dalle fatture stampate, ai documenti XML strutturati, fino a documenti potenzialmente dannosi presenti sul web.

Il principale risultato di questa tesi consiste nella progettazione, sviluppo e sperimentazione di un sistema per l'estrazione di informazioni da documenti stampati. Il sistema è stato progettato per operare quando l'insieme dei layout possibili dei documenti da analizzare è sconosciuto e può evolversi nel tempo.

Il sistema utilizza informazioni relative all'impaginazione dei documenti per definire le specifiche regole che possono essere utilizzate per estrarre le informazioni cercate dal documento. Questo sistema è il primo, al meglio della mia conoscenza, ad essere in grado di rilevare se il documento sottoposto ad analisi ha un layout sconosciuto, e quindi richiede nuove regole di estrazione. In tal caso, viene utilizzato un algoritmo di machine learning al fine di costruire le regole di estrazione utilizzando solo il documento in esame. Un altro contributo innovativo del nostro sistema è la capacità di sfruttare continuamente il feedback ottenuto dagli utenti, al fine di migliorare la sua capacità di estrazione di informazioni.

Ho sperimentato un metodo per il rilevamento automatico e la correzione di errori in sistemi OCR. Questo algoritmo utilizza le informazioni riguardanti gli errori più comunementi fatti dai sistemi OCR nel ricono-

scere i caratteri e la tipologia di informazioni estratte dai documenti per proporre e convalidare correzioni.

Viene inoltre proposto un sistema per la generazione automatica di espressioni regolari finalizzate alla estrazione di testo. Il sistema sviluppato si basa sul genetic programming e usa un insieme di esempi forniti dall'utente per guidare la ricerca evolutiva di una espressione regolare adatta al compito.

Riguardo l'estrazione di informazioni da documenti strutturati viene presentato un approccio, basato su genetic programming, per la sintesi di uno schema, a partire da un insieme di documenti XML di esempio. Lo strumento sviluppato accetta come ingresso uno o più documenti XML e automaticamente produce uno schema, in linguaggio DTD, che descrive la struttura dei documenti analizzati.

L'ultimo capitolo di questa tesi tratterà il tema della sicurezza web. Si è cercato di valutare l'abilità delle pubbliche amministrazioni italiane di avere il pieno controllo dei loro siti web. Inoltre si è cercato di sviluppare un sistema capace di individuare un particolare tipo di intrusioni che sta diventando di notevole interesse nel web.

Chapter 1

Introduction

Information extraction from documents plays a key role in many areas: office automation, knowledge management, intelligence, machine-to-machine interactions and so on. This task is particularly challenging when handling unstructured documents, or documents that do not carry any explicit structural description.

For example, consider an invoice processing workflow: each firm generates invoices with its own template and the receiver has to find the desired items on each invoice, e.g., invoice number, date, total, etc. In practice, this information extraction task is often performed by human operators, despite the huge advances and widespread diffusion of Information and Communication Technology. Similar scenarios requiring manual or semi-manual processing occur in many other contexts, for example, the analysis of an XML document or a data log produced by an unknown source. Therefore the development of systems that help to automatize information extraction tasks may have a significant impact in many practical environments.

The practical relevance of this problem may only grow in the near future. The increasing power and diffusion of computing and information-storage resources have reached a level in which virtually every organization may easily collect and preserve huge amount of heterogeneous documents related to its daily operations. In this scenario, the elaboration in real-time of new documents coming from unknown sources will be a daily activity. When handling such kind of data, the ability of an algorithm to perform information extraction accurately on new, unseen data after having trained on a small learning data set will be useful. Machine learning is a computing paradigm that has a great potential in this context.

In this thesis we will present novel applications of machine learning

techniques applied to information extraction from documents of different types, ranging from printed invoices to structured XML documents to potentially malicious documents exposed on the web. The applications presented in this thesis have been published in international journals and conferences

The work described in this thesis has been developed in the Machine Learning Lab at University of Trieste.

1.1 Thesis outline

The main results of this thesis consist in the design, development and experimental evaluation of a system for information extraction from printed (PDF-like) documents. These results have been recently published on a top-level scientific journal [15]. Information extraction from printed documents is still a crucial problem in many interorganizational workflows, since printed documents do not carry any explicit structural or syntactical description. However each printed document is characterized by a specific layout (the geometrical position of text on the page) that heavily depends from the document producer (source). Our system (which we call PATO) uses the layout information to define layout-specific extraction rules (wrappers) that can be used to extract information from a document.

Our proposal makes some novel and crucial contributions: the system (i) is able to figure out whether no suitable wrapper exists and generates one when necessary, in that case, (ii) is able to define a new wrapper starting from few example documents and, finally, (iii) considers feedback from human operators an integral part of the design. PATO assumes that the need for new source-specific wrappers is part of normal system operation: new wrappers are generated on-line based on a few point-and-click operations performed by a human operator on a GUI. The wrapper selection is performed using an SVM-based classifier based only on image-level features and using a nearest-neighbor approach for detecting the need of new wrapper. The approach behind the wrapper itself is based on probability: we derived a general form for the probability that a sequence of blocks contains the searched information. The role of operators is an integral part of the design and PATO may be configured to accommodate a broad range of automation levels. We show that PATO exhibits very good performance on a challenging dataset composed of more than 600 printed documents drawn from three different application domains: invoices, datasheets of electronic components, patents. We also perform an extensive analysis of the crucial trade-off between accuracy and automation level. Its results have been published in [15, 93, 14].

The experimental evaluation showed that PATO is robust to OCR errors, that is, it is able to locate the correct information despite the presence of noise and characters misrecognition. However a system able to perform an automatic correction of these errors will allow PATO to gain an improved accuracy. In Chapter 3 we propose a method for the automatic detection and correction of OCR errors. Our algorithm uses domain-knowledge about possible misrecognition of characters to propose corrections; then it exploits knowledge about the type of the extracted information to perform syntactic and semantic checks in order to validate the proposed corrections. We assess our proposal on a real-world, highly challenging dataset composed of nearly 800 values extracted from approximately 100 commercial invoices, obtaining very good results, which have been published in [94].

Systems presented in Chapter 2 and 3 deals with weakly-structured documents: the weak-structure is given by the geometrical position of text on the page and does play a role in the information extraction process. When the documents under analysis are mere text sequences, a more common approach is the use of regular expressions to perform information extraction. Regular expressions are a long-established technique for a large variety of text processing applications, but the constructing of a regular expression suitable for a specific task is a tedious and error-prone process. In Chapter 4 we propose a system for the automatic generation of regular expression for text-extraction tasks. We propose a system based on genetic programming (GP); the user describes the desired text extraction task by providing a set of labelled examples, in the form of text lines. The system uses these examples to drive the evolutionary search for a regular expression suitable for the specified task. The obtained regular expressions may be used with common engines such as those that are part of Java, PHP, Perl and so on. Usage of the system requires neither familiarity with GP nor with regular expressions syntax. In our GP implementation each individual represents a syntactically correct regular expression and the fitness consists of two objectives to be minimized: the edit distance between each detected string and the corresponding examples, the size of the individual. We performed an extensive experimental evaluation on 12 different extraction tasks applied to real-world datasets. We obtained very good results in terms of precision and recall. Its results can be found here [11] and another paper is pending review

The previous chapters focussed on scenarios where documents are not associated with any explicit structural or syntactical description. In several cases, though, such description is often available and the eXtensible Markup Language (XML) is one of the languages used to this end. The XML documents are an essential ingredient of modern web technol-

ogy and are widely used in machine-to-machine information exchange. A schema document describes the type of information contained in an XML document and the constraints which must be met. Although availability of a schema for a specific application is very important, in practice many applications either do not have any schema or the corresponding schema is incomplete. Chapter 5 proposes a solution to this problem, presenting an approach for schema synthesis starting from a set of XML sample documents. In this chapter we describe the design, implementation and experimental evaluation of a tool for DTD synthesis based on Genetic Programming. Our GP-DEI tool (Genetic Programming DTD Evolutionary Inferer), takes as input one or more XML documents and automatically produces a schema, in DTD language, which describes the input documents. Usage of the GP-DEI requires neither familiarity with GP nor with DTD or XML syntaxes. We performed an extensive experimental evaluation of our tool on a large collection of several sets of real world XML documents, including documents used in an earlier state-of-the-art proposal.

Finally, in Chapter 6 we move our attention to a different application domain: the web security. We attempted to apply several of the machine learning techniques applied to the previous application domains, in order to assess the ability of Italian public administrations to be in full control of the respective web sites. In particular, we attempted to detect certain types of fraudulent intrusions that are becoming of practical interest on a large scale. The analysis of Italian public administrations web sites was performed examining several thousands sites, including all local governments and universities; we found that approximately 1.5% of the analyzed sites serves contents that admittedly are not supposed to be there. A novel form of modification consists of the addition of new pages at URLs where no page should exist. Detecting their existence is very difficult because they do not appear during normal navigation and are not indexed by search engines. Most importantly, drive by attacks leading users to hidden URLs, for example to spread malware or phishing credentials, may fool even tech-savvy users, because the hidden URL could be hosted within a trusted site, possibly with HTTPS. We propose an approach based on an SVM classifier for detecting such URLs using only on their lexical features, which allows to alert the user before actually fetching the page. We assess our proposal on a dataset composed of thousands of URLs. Results have been published in [95] and another paper is pending review.

1.2 Publication list

- [14] A. Bartoli, G. Davanzo, E. Medvet, and E. Sorio. Improving features extraction for supervised invoice classification. In *Proceedings of the 10th IASTED International Conference*, volume 674, page 401, 2010
- [93] E. Sorio, A. Bartoli, G. Davanzo, and E. Medvet. Open world classification of printed invoices. In *Proceedings of the 10th ACM symposium on Document engineering*, DocEng '10, pages 187–190, New York, NY, USA, 2010. ACM
- [15] A. Bartoli, G. Davanzo, E. Medvet, and E. Sorio. Semisupervised wrapper choice and generation for print-oriented documents. *IEEE Transactions on Knowledge and Data Engineering*, page 1, 2012
- [94] E. Sorio, A. Bartoli, G. Davanzo, and E. Medvet. A domain knowledge-based approach for automatic correction of printed invoices. In *Information Society (i-Society), 2012 International Conference on*, pages 151–155. IEEE, 2012
- [11] A. Bartoli, G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1477–1478. ACM, 2012
- [95] E. Sorio, A. Bartoli, and E. Medvet. A look at hidden web pages in italian public administrations. In *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, pages 291–296. IEEE, 2012

Chapter 2

Document processing

2.1 Overview

Despite the huge advances and widespread diffusion of Information and Communication Technology, manual data entry is still an essential ingredient of many interorganizational workflows. In many practical cases, the glue between different organizations is typically provided by human operators who extract the desired information from printed documents and insert that information in another document or application. As a motivating example, consider an invoice processing workflow: each firm generates invoices with its own firm-specific template and it is up to the receiver to find the desired items on each invoice, e.g., invoice number, date, total, VAT amount. Automating workflows of this kind would involve template-specific extraction rules—i.e., *wrappers*—along with the ability to: (i) select the specific wrapper to be used for each document being processed (*wrapper choice*), (ii) figure out whether no suitable wrapper exists, and (iii) generate new wrappers when necessary (*wrapper generation*). The latter operation should be done promptly and possibly with only one document with a given template as it may not be known if and when further documents with that template will indeed arrive. Existing approaches to information extraction do not satisfy these requirements completely, as clarified below in more detail.

In this chapter we propose the design, implementation and experimental evaluation of a system with all these features. Our system, which we call PATO, extracts predefined items from *printed documents*, i.e., either files obtained by scanning physical paper sheets, or files generated by a computer program and ready to be sent to a printer. PATO assumes that the appearance of new templates is not a sort of exceptional event but is part of normal operation.

Wrapper generation has received considerable attention by the research community in the recent years, in particular in the context of information extraction from web sources [110, 51, 25, 35, 69, 30, 40]. Wrapper-based approaches fit this scenario very well as they may exploit the syntactic structure of HTML documents. In this work we focus instead on printed documents, which are intrinsically different from web pages for two main reasons. First, printed documents do not embed any syntactical structure: they consist of a flat set of blocks that have only textual and geometrical features—e.g., position on the page, block width and height, text content, and so on. Second, the representation of a document obtained from a paper sheet usually includes some noise, both in geometrical and textual features, due to sheet misalignment, OCR conversion errors, staples, stamps and so on. PATO addresses wrapper generation based on a maximum likelihood method applied to textual and geometrical properties of the information items to be extracted [73]. The method is semisupervised in that, when no suitable wrapper for a document exists, PATO shows the document to an operator which then selects the items to be extracted with point-and-click GUI selections.

There are significant differences between web information extraction and our scenario even in the wrapper choice component. Web information extraction often focuses on a single source at once [69], in which case wrapper choice is not an issue. Recently, the research focus shifted on a multi-source scenario motivated by the interest in integrating deep web sites, i.e., databases which are only accessible by filling search forms on specialized web sites [26, 35, 25]. In those cases, however, it is the system that actively accesses each web site: since the system knows which source is being accessed, the system also knows exactly which wrapper to choose. Our wrapper choice problem is different since our system passively receives documents without any explicit indication of the corresponding source. PATO, thus, is required to infer the source from the document itself. PATO addresses wrapper choice based on a combination of two classifiers applied to image-level properties of the document being processed [93]. One of the classifiers determines whether the document has been emitted by an unknown source, in which case the wrapper generation component comes into play. The other classifier determines the existing wrapper to use.

Our work complements earlier proposals for multi-source wrapper choice and generation by systematically introducing the *human-in-the-loop* factor, an element that we believe is essential for coping with a dynamic set of sources in a practical setting. PATO accommodates a broad range of automation levels in terms of operator-provided feedback, which may occur independently in the two phases of wrapper choice and wrapper generation. Operators need not have any specific IT-related skills,

because their feedback merely consists of basic point-and-click selections on a dedicated GUI to either confirm or reject the system suggestions. These design choices allow tailoring the system to widely differing scenarios easily, including the scenario where 100% extraction accuracy is required. This case corresponds to the maximal amount of operator involvement, because PATO expects a feedback on each processed document. Even in this case, PATO may still provide a practical advantage with respect to traditional (i.e., spreadsheet-based) data entry, since it can greatly reduce the operators' engagement time without affecting accuracy (see Section 2.6).

We evaluated the performance of PATO on a challenging dataset, composed of 641 digitized copies of real-world printed documents concerning three different extraction scenarios: (i) date, total, VAT etc. from invoices issued by 43 different firms; (ii) title, inventor, applicant etc. from patents issued by 10 different patent sources; (iii) model, type, weight etc. from datasheets of electronic components produced by 10 different manufacturers. We examined the accuracy of PATO from several points of views and in all the 9 different configurations that it supports, corresponding to all possible combinations of full, partial or no automation at the wrapper choice and generation stages. We also evaluated the time required by human operators for each of the processing steps in which they may be involved, as well as the frequency of their intervention as a function of the configuration options. To place these results in perspective, we estimated the time required by human operators to extract the same information from a printed copy and then fill a spreadsheet. All these data enabled us to gain important insights into the practical impact of the various design options available and the accuracy levels that can be obtained.

2.2 Related work

Our work addresses a specific multi-source information extraction problem. In this section we place our contribution in perspective about the existing literature. We discuss approaches designed for web documents separately from those for printed documents.

2.2.1 Multi-source web information extraction

Multi-source web information extraction is primarily concerned with web sources that are accessed through input elements, typically for performing queries [57]. A crucial step for wrapper generation consists of determining the interface of each source, e.g., how to identify the relevant form inputs in the web page. The position of these elements may widely

vary across different sources and there are no textual labels for form inputs that are universally meaningful. Most of the solutions proposed in [57] are based on heuristics that leverage syntactic features of the web documents for identifying candidate input elements. Broadly speaking, our scenario faces similar problems, as a searched information item (e.g., invoice number) can be placed at very different positions among different sources, usually with different textual labels (`Num.`, `N.`, `#.`, `Ref.`, ...) or with no textual label at all. On the other hand, as pointed out in the introduction, our approach to wrapper generation must work at a different abstraction level because printed documents have only textual and geometrical features (which may also be noisy as a result of a scanning process) and do not have any explicit syntactic structure.

Similar remarks apply to all the proposals reviewed in this section, the only exception being the method in [69] that uses only visual features of the rendered web documents. In the cited work a result page produced by a web source is segmented in blocks based on the visual appearance of the page. The blocks are organized in a tree structure and matched to trees for the same source already available and previously annotated with the items to be extracted. The matching uses several heuristics based on hierarchical, textual and geometrical properties of blocks. Our approach works on an unstructured sequence of OCR-generated blocks, each associated with position, size and textual content. The block containing a given information item is the one which maximizes the probability distributions of block variables, whose parameters are fitted with values obtained from previously annotated documents. It shall also be pointed out that in web information extraction there are often multiple *records* in each page, i.e., sets of information items following a predefined schema—e.g., a page returned by an e-commerce site contains many results, each composed of name, price, description and so on. It follows that (i) wrappers must be able to identify a varying number of records, and (ii) even a single page may provide opportunities useful for wrapper generation, for example by identifying portions of the page with similar structure or visual appearance—as done in [69]. In our scenario, in contrast, there is only one record for each document: a wrapper must identify exactly one record and the presence of recurring patterns in a page is generally irrelevant for wrapper generation.

Wrapper generation for search engine results pages is considered in [110]. The cited work aims at connecting thousands of different search engines and argues that it is not practical to manually generate a wrapper for each source. The approach proposes an unsupervised method for generating a wrapper automatically, which is based on stimulating the source with a sample query and then analyzing the obtained result page. In our scenario we cannot stimulate sources to emit sample documents. More-

over, we need to generate a wrapper as soon as the first document from a new source arrives—which also introduces the problem of realizing that the document has been indeed emitted by a new source.

Search engines are also the topic of [51]. This work proposes a tool which integrates several e-commerce search engines (ESE) making them accessible from a single interface. A dedicated component crawls the web for identifying ESE and clusters them according to the domain—ESE selling similar goods will be placed in the same cluster. The wrapper for each ESE is then generated automatically. A similar problem, which consists of crawling and then clustering hidden-web sources, is tackled in [9], with more emphasis on the ability to cope with a dynamic set of sources.

When multiple sources may be stimulated to provide exactly the same data with different templates, the information redundancy of those sources may be leveraged to improve the corresponding wrappers, as proposed in [25]. While this approach may be often exploited in the web, it cannot be applied in our domain—we cannot stimulate two firms to emit invoices with the very same items. Another approach to wrapper generation is proposed in [35], which combines extraction results obtained with existing wrappers (generated for other sources) with operator-provided domain knowledge. In principle a similar approach could be applied to our case, although it would be necessary to carefully examine the resulting amount of operator involvement and, most importantly, the corresponding IT skills required.

In the recent years several studies on web information extraction have started to focus on more automatic techniques for wrapper generation, which implies considering the role of human operators [9, 35, 110]. A new performance measure called *revision* aimed at quantifying the amount of human effort has been proposed in [69]. Revision measures the percentage of sources for which the wrapper cannot perform a perfect extraction, the rationale being that a manual correction should suffice to achieve perfect extraction. We also provide special emphasis on human operators but we prefer to quantify the effort involved in using our system as it is: we perform an extensive experimental evaluation of the trade-off between extraction accuracy and time amount of operators' involvement, including a comparison against a traditional data entry solution, i.e., spreadsheet-based.

The ability to accommodate a large and dynamic set of sources efficiently is a crucial requirement in Octopus [26]. The cited work removes a common assumption in web information extraction, i.e., that the relevant sources for a given domain have been identified a priori, and assumes instead that the operator cannot spend much time on each new data source. Our standpoint is very similar in this respect because PATO assumes that

the appearance of new sources is part of normal operation. Leaving the different document representations and internal algorithms aside, PATO is similar to Octopus in that both systems work automatically but allow the operator to provide feedback and correct errors.

A radically different multi-source web extraction problem is considered in [38]. In this case the target consists of automatically extracting tables from web lists, which is difficult because list delimiters are inconsistent across different sources and cannot be relied upon to split lines into fields. The authors use a language model and a source- and domain-independent table model built from a precompiled corpus of HTML tables aimed at identifying likely fields and good alignments: different field value candidates depending on different line splits are evaluated basing on their likelihood according to the language and table models. The approach here presented consists of training a graphical model—conditional random fields (CRF)—with a corpus in which each text line has been previously labeled. Documents are represented in terms of textual features that turn out to be both meaningful and useful for text-only table documents—e.g., number of space indents, all space lines and so on.

For a wider and deeper survey of web information extraction systems please see [30] and [40].

2.2.2 Information extraction from printed documents

Systems for information extraction from printed documents can be subdivided in two categories, depending on whether all documents are processed with the same wrapper, or each document is processed by a wrapper tailored to the document source. Systems that use the same wrapper for all documents are, broadly speaking, suitable for a single application domain and depend on a fair amount of a priori knowledge about that specific domain—i.e., invoices, medical receipts, and so on [18, 28, 6]. Moreover, these wrappers are usually limited to documents written in a single language. For example, a precompiled table of "main primary tags" is used in [18] to identify labels of interesting information. Text-based and geometrical information is used in [28] for identifying the desired "tags" to be extracted (the cited work proposes a system that may also be configured to use a wrapper tailored to each source, as discussed below). A system focused on table forms is proposed in [6], where the document is subdivided in boxes and then the boxes containing the values to be extracted are identified based on semantic and geometric knowledge.

Systems that instead have a wrapper for each source have wider applicability but require a preliminary wrapper choice operation [28, 4, 89, 81]. This operation may be performed in two radically different scenarios:

static multi-source scenario, where all sources are known in advance and each document submitted to the system has been certainly emitted by one of these sources; *dynamic multi-source* scenario, where the set of sources is not known in advance: a document may be associated with one of the sources already known to the system, but it may also be associated with a source never seen before. In the latter case the system must be able to detect the novelty and generate a new wrapper accordingly. Needless to say, the dynamic multi-source scenario is much more challenging but encompasses a much broader range of practical problems. To the best of our knowledge, our work is the first description of a system for information extraction from printed documents in a dynamic multi-source scenario.

Insights about wrapper choice in this context can be found in [33]. This problem is usually cast as a classification problem, each class being composed by documents emitted by the same source. Different document features (e.g., image-level or text features) can be used for the purpose of classifying printed documents [3, 5, 84, 48], which is a premise for the actual information extraction (that is not considered in the cited works).

A static multi-source approach based only on visual similarity is proposed in [3]. Each document is represented as a Gaussian mixture distribution of background, text and saliency. A nearest-neighbor classifier identifies the document class based on an approximation of the Hellinger distance.

The dynamic multi-source approach proposed in [5] is perhaps the closest to ours. The cited work uses a nearest-neighbor classifier based on the main graphic element of documents emitted by the same source, usually the logo. We use instead a two-stage approach based on the full document image: a nearest-neighbor classifier for detecting whether the document has been emitted by a new source, followed by a multi-class Support Vector Machine which determines the existing source [93] (the reason why we have chosen to use two classifiers is discussed later).

An approach based on image-level features similar to ours, but restricted to a static multi-source scenario, is presented in [84]. A radically different classification approach based on text features (graphs of words) is proposed in [48]. This approach accommodates dynamic multi-source scenarios only in part: the system may detect that a document has not emitted by one of the sources already known, but it is not able to define a new class. Thus, following documents from the same source will be again deemed to belong to a unknown source.

Concerning wrapper generation, most of the commercial solutions currently available require a specialized operator to "draw" a wrapper for the given class of documents. This operation may be performed through a specialized GUI, but it requires specific skills hardly found in data-

entry operators, e.g., writing macros, using data description languages or notations, testing the correctness of the description. An essential aspect of our work is that we strive to keep the wrapper generation as simple and lightweight as possible, on the grounds that this operation may be frequent in the scenarios of our interest. With our approach it suffices to point and click on those OCR-generated blocks that contain the desired information. It seems reasonable to claim that any administrative operator without any specific IT-related skill is in position to perform this operation easily.

The approach proposed in [28] is based on a supervised labeling procedure which produces a table (file) of logical objects of interest and related tags, which have been manually located on a number of sample invoices of the given class. Our wrappers determine relevant OCR-generated blocks basing on a probabilistic approach applied to their geometric and textual properties—size, position, page, text length, text alignment, content. The result is that even when the OCR fails to detect correctly such pieces of text as `"Total"` or `"Price"`, our system generally is still able to identify the relevant information. On the other side, OCR errors simply prevented the system in [28] from identifying the searched tags or labels.

Semisupervised wrapper generation for digitally-born PDF documents has been proposed in [82, 41, 50]. Wrapper choice is not addressed, while wrapper generation is based on structural information. The approach proposed in [82] exploits the knowledge represented in an ontology. The document is first subdivided in portions based on heuristics that analyze several visual features, including lines and space distribution. Based on an ontology expressed into an ontology description language, document portions are then translated into sets of rules that constitute a logic program, which can be used for the actual information extraction from the document. In [41] the operator specifies textual and syntactical properties of the items to be extracted. Spatial constraints which link each textual item with its surrounding items (e.g., labels) are then defined using fuzzy logic. The blocks that best satisfy those constraints are finally identified with a hierarchical bottom-up exploration. In [50], the system generates an attributed relational graph describing adjacency relations between blocks. An operator annotates this graph with geometric, logical, structural and content-related block attributes, including the specification of which graph nodes contain data to be extracted.

We explored similar ideas in our early experiments and associated each block with some information about its surrounding blocks. First, by requiring that each block selection by the operator be accompanied by the selection of the corresponding label block (e.g., when selecting a block containing a price item, also the block containing “Amount” or

"Total" should be selected). Then, by introducing further block variables measuring the distance from the closest blocks. In either case we could not find any significant improvement, hence we chose to pursue the approach that appeared to be simpler to analyze and implement.

A system for information extraction from forms, and only from them, is proposed in [81]. The system assumes a static multi-source scenario and performs wrapper choice based on a classifier fed with a mixture of textual, graphical and geometrical document features. Wrappers identify the desired information items by means of heuristics based on block position, searched keywords, syntax of block contents. Wrapper generation is not automatic.

Finally, this work integrates our previously separated solutions for wrapper choice [93] and wrapper generation [73]. With respect to the cited works, we show how to include novel document sources in the system seamlessly and without interrupting the normal processing flow; we provide a more compact version of the wrapper construction algorithm; and we assess the role of human operators experimentally, by exploring the trade-off between accuracy of extraction and automation level extensively.

2.3 Our Framework

Documents of our interest are *multi-page images*. A document is associated with a *schema* and a *template*, as follows. The schema describes the information to be extracted from the document and consists of a set of typed *elements*: for each element, the document contains zero or one *value*. For example, a schema could be `date`, `totalAmount`, `documentNumber`; a document with this schema could contain the values `"7/2/2011"`, `"23,79"` and no value for the respective elements. Without loss of generality, we consider that the system receives in input a set of documents with the same schema, e.g., invoices, or patents, or electronic datasheets.

The template describes how the information is physically arranged on the pages of a document; a document source generates documents with the same template. The *wrapper* is the set of rules for locating and extracting the information described by the schema from a document generated by a given source. We assume a multi-source scenario and do not require that the set of sources be known in advance, i.e., we assume a *dynamic* multi-source scenario.

PATO processes a document d according to the following workflow.

1. In the *wrapper choice* stage (Section 2.4), the system selects the appropriate wrapper W for d or, when such a wrapper is not avail-

able, prepares itself to generate and use a new wrapper in the next stage.

2. In the *blocks location* stage (Section 2.5), the system executes an OCR procedure on d and then uses W to locate the required information. That is, for each element e of the schema, the system locates the rectangular region of d where the value v_e^* is graphically represented—we call that region the *block* b_e^*.

3. Finally, for each element e of the schema, the system extracts the value v_e^* from the textual content of the corresponding block b_e^*. This stage consists of applying an element-specific regular expression that aims to remove from the textual content of the block the extra-text that is not part of v_e^*. A method for performing this task is proposed in Chapter 3.

The workflow processing stages share access to a *knowledge repository* containing the data needed for system operation—i.e., information about different document sources (Section 2.4), wrappers (Section 2.5.3) and mapping between them (Section 2.4).

We carefully designed our system to allow selecting different trade-offs between accuracy and amount of human involvement. As described in the next sections, the two main workflow stages can be configured independently of each other so as to be fully automated, or partly automated, or not automated at all.

2.4 Wrapper choice

The wrapper choice workflow consists of 4 stages: image processing, feature extraction, novelty detection and classification. The *image processing* stage includes binarization, deskew and *rolling*, i.e., an operation aimed at aligning all documents in the same way, by removing the empty part at the top of the document. We found that this operation is often necessary in practice, because images obtained by scanning very similar real-world documents could be significantly different due to human errors made during their digitization—positioning errors on the scanner area, non-standard document sizes, cut documents, and so on. To implement rolling, we identify the upper relevant pixel of the image using an edge recognition algorithm applied to a low-resolution version of the image obtained by resizing the original with a $\frac{1}{6}$ scaling factor; we reduce the image in order to remove the noise caused by the scanner and small texts. We consider the first edge as the upper relevant pixel. To maintain the image size, we remove all the content between the upper relevant pixel and the top border to append it at the end of page.

The *feature extraction* stage transforms the image into a numerical vector f including: (i) density of black pixels; and (ii) density of the image edges. In detail, we divide the image in a 16×16 grid and for each cell of the grid we compute the black pixel density, i.e., a number in the range $[0, 100]$ representing the percentage of black pixels in the cell. Then, we reduce the resolution of the original image and apply an edge detector. We repeat the previous procedure on the resulting image consisting only of the edges, i.e., we compute the black pixel density for each cell in a 16×16 grid. We concatenate the two resulting vectors in order to obtain a features vector f of length $16 \times 16 \times 2 = 512$. Figure 2.1 summarizes the process here described for a generic invoice, for further details please refer to [93].

The next workflow stages are based on the notion of *class*. Documents with similar graphical appearance belong to the same class. Classes are stored in the knowledge repository as feature vectors sets, as follows. For each class C, we store a set $\mathcal{F}_C$ composed of up to k documents of that class. If there are less than k documents for a class, $\mathcal{F}_C$ contains all the documents available, otherwise it contains the last k documents of the class submitted to the system. After early experiments performed on a small portion of our dataset, we chose $k = 10$. For each class C, the system also maintains: (i) the centroid of the elements in $\mathcal{F}_C$, denoted x_C; (ii) the distances between each element in $\mathcal{F}_C$ and x_C; (iii) mean μ_C and standard deviation σ_C of these distances. These values are updated whenever the composition of $\mathcal{F}_C$ changes. A predefined knowledge of a few classes is needed in order to initialize the system for the wrapper choice stage: after early experiments performed on a small portion of our dataset, we set this starting knowledge to 7 classes with 2 documents each.

It is important to emphasize that a class description does not provide any clue as to how to locate and extract the desired information from documents of that class. The rules for this task are encoded into wrappers. The association between classes and wrappers is stored in the knowledge repository.

The *novelty detection* stage is a crucial component for coping with the dynamic multi-source scenario: it determines whether the document can be processed with a wrapper already known to the system, or whether a new wrapper has to be generated (i.e., $W = \emptyset$). Note that in the former case the wrapper is *not* identified and the choice of the (existing) wrapper to be used will be made in the following classification stage, discussed below. The novelty detection stage works as follows:

1. find the class C with the closest centroid x_C to the features vector f of the input document d; let δ_C denote the distance between f

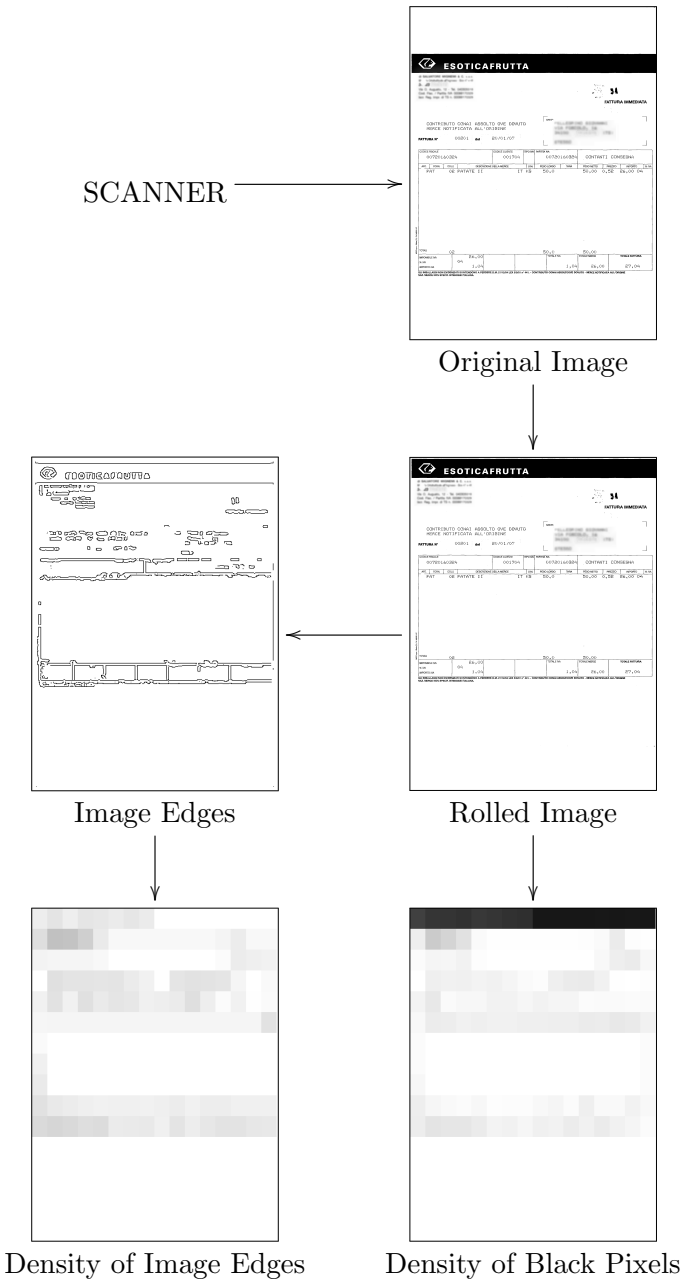


Figure 2.1: Features extraction work-flow.

and x_C;

2. compute two values $r_C = \mu_C + \alpha \cdot \sigma_C$ and $r_C^t = \epsilon \cdot r_C$ ($\alpha > 0$ and $\epsilon > 1$ are two system parameters);

3. if $r_C^t < \delta_C$, set $W = \emptyset$ and skip the classification stage, otherwise make the document proceed to the classification stage.

In other words, a new wrapper is needed when the distance of the document d from the closest centroid x_C exceeds a certain threshold. This threshold r_C^t depends on the distances from x_C of the documents that are already known to be instances of class C—the closer these documents to x_C the smaller r_C^t and vice versa. The threshold also depends in a similar way on the standard deviation of these distances—the more uniform the distances, the smaller the threshold.

The *classification* stage is entered only when a new wrapper is not needed. This stage takes the document d as input and outputs the wrapper W to be used for d. More in detail, this stage associates d with a class C and the wrapper W associated with the class is then extracted from the knowledge repository (the reason why we do not associate d with the class selected by the novelty detector and use an additional classifier is described below). The classification is performed by a multi-class Support Vector Machine (SVM) with linear kernel. The SVM operates on a feature vector f' obtained from f with a dimensionality reduction based on Principal Component Analysis (PCA). The set of features in f' has been selected so as to ensure a proportion of variance greater than 95% on the documents of the knowledge repository. This set of features is re-evaluated during system operation as discussed in the next section.

Since the novelty detector is essentially a k-nearest neighbor (k-NN) classifier, we could have avoided the use of the additional SVM-based classifier. That is, when the novelty detector finds that $\delta_C \leq r_C^t$ for a document d, the wrapper W for d could have been set to the class C of the nearest centroid. Our early experiments, however, showed that an SVM-based classifier exhibits much better accuracy than a k-NN classifier.

We attempted to remove the need of a separate novelty detector by using only the SVM classifier, in particular, by taking the probability estimates produced by the classifier as indications useful for novelty detection (as suggested in [109]), but this approach did not yield good results. An option that we did not explore consists of using an additional one-class SVM for novelty detection (as opposed to the multi-class one which we use for classification).

Another design option which we did not explore consists of performing wrapper choice based on the OCR-generated document representation, rather than working exclusively on image-level document features.

2.4.1 Human Intervention and Updates to the Knowledge Repository

The wrapper choice stage may be configured to work in three different ways based on the value of a parameter H_{WC}, which can be one among Unsupervised (wrapper choice never delegated to a human operator), Semisupervised, Supervised (always delegated to a human operator). We remark that the human operator only deals with the notion of graphical similarity—he sees neither classes nor wrappers, which exist only within the system. The mapping between the two notions occurs internally to the knowledge repository, as each class is associated with exactly one wrapper.

In detail, let d denote the document being processed and C the class selected by the SVM classifier. The systems works as follows:

- $H_{\mathrm{WC}} = \mathsf{Unsupervised} \Longrightarrow$ Human intervention is never required. Updates to the knowledge repository occur as follows:

  1. $\delta_C \leq r_C \Longrightarrow d$ is associated with C.
  2. $r_C < \delta_C \leq r_C^t \Longrightarrow$ A new class C' which contains only d is defined. This class is associated with the (existing) wrapper W associated with C.
  3. $r_C^t < \delta_C \Longrightarrow$ A new class C' which contains only d is defined. This class is associated with a new empty wrapper W.

- $H_{\mathrm{WC}} = \mathsf{Semisupervised} \Longrightarrow$ The system operates as in Unsupervised, except that in cases 2 and 3 it requires human intervention, as follows.

  The system determines the sequence of four classes, say C_1, C_2, C_3, C_4, whose centroids are closest to the feature vector f of d (these classes are thus selected by the novelty detector). Then, the system presents on a GUI an image of d and the image of the most recent document processed by the system for each of C_1, C_2, C_3, C_4 (Figure 2.2). We chose to show 4 classes because we found, during our early experiments, that this number suffices to always identify the correct class; the operator may browse through the full list of existing classes, however. The operator is required to choose from two options:

  1. Select the document most graphically similar to d. In this case the system performs the following (let C_1 be the class of the document selected by the operator): (a) $r_C < \delta_C \leq r_C^t \Longrightarrow$ associates d with C_1; (b) otherwise $\Longrightarrow$ defines a new class C' containing only d and using the (existing) wrapper associated with C_1.

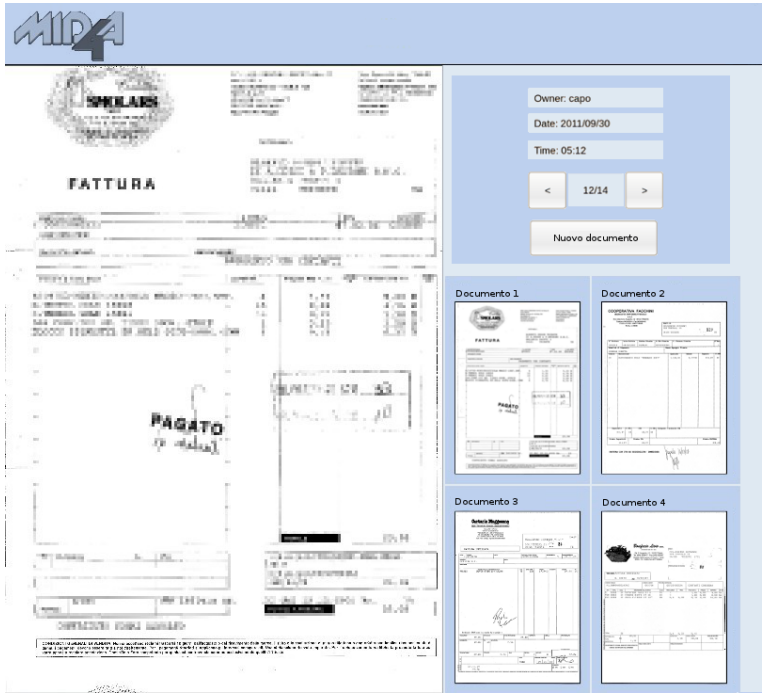


Figure 2.2: Wrapper selection GUI.

2. Specify that in the list of presented documents there is no one similar enough to d. In this case the system defines a new class C' containing only d and associated with a new *empty* wrapper.

- $H_{\text{WC}} = \mathsf{Supervised} \implies$ The system operates as in Semisupervised, except that in case 3 (i.e., $\delta_C \leq r_C$) the human intervention is as follows.

  The system presents on a GUI a set of images selected as above and augmented with the image of a document associated with the class C selected by the SVM classifier. Class C is indicated as the class suggested by the system for d. The operator is required to choose from the same two options as above plus a third option:

  1. Select the document most graphically similar to d.
  2. Specify that none of the presented documents is similar enough to d.
  3. Confirm the suggestion by the system.

  Options (1) and (2) are handled as above, whereas option (3) results in the association of d with the class C chosen by the classifier. Whenever the human operator modifies the choice suggested by the system,

moreover, the system recomputes again its choice for all queued documents and does so before presenting them to the human operator (end of Section 2.3).

Whenever the wrapper choice stage is completed, the system performs the following tasks:

1. Records the association between the input document d and the corresponding class C. This step also involves following operations: (i) the features vector f of d is added to $\mathcal{F}_C$; (ii) if the number of elements of $\mathcal{F}_C$ is greater than a predefined value k ($k = 10$ in our prototype), then the oldest element is removed; (iii) the parameters of C required by the novelty detector are recomputed (centroid, distances: Section 2.4).

2. Recomputes the PCA parameters, i.e., evaluate again the set of features which grants a proportion of variance greater than 95%.

3. Retrains the SVM classifier.

2.5 Blocks location

2.5.1 Overview

The blocks location stage takes as input the document d and the corresponding wrapper W selected by the wrapper choice stage. In case W is empty, W has to be generated by the system with the feedback provided by a human operator. A key strength of our proposal is that the wrapper can be generated automatically from intuitive operations performed on a GUI by an administrative operator without any specific IT-related skills. The operator has merely to point and click on those OCR-generated blocks that contain the desired information (Section 2.5.4).

The blocks location stage begins with an OCR process, that transforms the document d into a set of *blocks*, each specified by position, size and content. The position is specified by the page number p and the coordinates x and y from the page origin (upper left corner). The size is specified by width w and height h. The content is specified by the single-line textual content l, expressed as a character sequence. It follows that a block is a tuple of *block attributes* $b = \langle p, x, y, w, h, l \rangle$.

The blocks containing the required information are located based on a probabilistic approach that we developed earlier in a more general form [73] and summarized in the next sections in a slightly more compact way, specialized for the specific application domain of interest in this work. The basic idea is as follows. We derived a general form for

the probability that a block b contains a value for a given schema element e. This function, which we call the *matching probability*, is a parametric function of the attributes of b. The wrapper generation consists of estimating these parameters based on the maximum likelihood approach applied to a set of sample documents. Applying a wrapper to a document d consists of selecting, for each element, the block of d that maximizes the corresponding matching probability.

2.5.2 Matching probability

The matching probability P_e of a given block is the probability that the block contains a value for the schema element e. To simplify the notation, in the following we shall focus on a single schema element and omit the specification of e. We want to express this probability, which concerns rather complex events, as a function of simple univariate probability distributions of independent random variables obtained from the block attributes—p, x, y, w, h, l.

The matching probability includes 6 attributes whose corresponding 6 random variables are, in general, dependent on each other:

$$P(b) = P(\langle p, x, y, w, h, l \rangle)$$

We identified a domain knowledge-based set of dependencies, which allowed us to elaborate and simplify the form of P, as follows.

First, we can use marginalization in order to write P basing on the possible values for the page p:

$$P(b) = \sum_k P(b \cap p = k)$$

where $P(b \cap p = k)$ is the joint probability of the following two events: b is the matching block and $p = k$. These two events are in general dependent. For example, consider a template of invoices where the total amount value may be located at the bottom of the first page or at the top of the second page: it follows that small values for y are more probable if the page attribute is equal to 2 and large values for y are more probable if the page attribute is equal to 1—in other words, the y attribute of the block is dependent on the p attribute. Accordingly, we can rewrite the joint probability in terms of conditional probability on the page p:

$$\begin{aligned} P(b \cap p = k) &= P(b|p = k)P(p = k) \\ &= P_k(\langle x, y, w, h, l \rangle)P(p = k) \end{aligned}$$

where $P_k(\langle x, y, w, h, l \rangle)$ is the probability that a block identified by the tuple $\langle x, y, w, h, l \rangle$ is the matching block, given that its page p is equal

to k. Concerning $P(p = k)$, we assume that there is a finite set $K = \{k_1, k_2, \dots\}$ of possible values for p, whose corresponding probabilities are $s_{k_1}, s_{k_2}, \dots$. In other words, $P(p = k) = s_k I(p; k)$, where $I(p; k) = 1$ for $p = k$ and 0 otherwise and $s_k = 0$ if $k \notin K$. In summary, we may rewrite P as follows:

$$P(b) = \sum_k s_k I(p; k) P_k(\langle x, y, w, h, l \rangle)$$

Concerning $P_k(\langle x, y, w, h, l \rangle)$, we assume that y and h are independent from the other three variables. In particular, note that, since blocks contain exactly one line of text, the height attribute h is largely independent from its text content l. Hence, we can write:

$$P_k(\langle x, y, w, h, l \rangle) = P_k^y(y) P_k^h(h) P_k^{x,w,l}(\langle x, w, l \rangle) \tag{2.1}$$

Concerning now $P_k^{x,w,l}(\langle x, w, l \rangle)$, we split the x, w, l dependency in one between x and w and another between w and the text content l.

- The dependency between x and w represents the fact that a given text could be aligned in three different ways: left, center or right (justified text may be handled in any of these three cases, for the purpose of this analysis). It follows that:
  - in case of left-alignment, x and w are independent;
  - in case of center-alignment, $x^c = x + \frac{w}{2}$ and w are independent;
  - in case of right-alignment, $x^r = x + w$ and w are independent.
- The dependency between w and l represents the fact that, in general, the longer the text content, the larger the block width. We define $w' = \frac{w}{\mathcal{L}(l)}$ as the average width of the characters composing the block text content, being $\mathcal{L}(l)$ the number of characters in l: we assume that w' and l are largely independent, since w' depends on the font size and type, rather than on the text content.

We can hence write $P_k^{x,w,l}(\langle x, w, l \rangle)$ in three possible forms depending on text alignment:

$$P_k^{x,w,l}(\langle x, w, l \rangle) = \begin{cases} P_k^x(x) P_k^{w'}(w') P_k^l(l), & \text{left} \\ P_k^{x^c}(x^c) P_k^{w'}(w') P_k^l(l), & \text{center} \\ P_k^{x^r}(x^r) P_k^{w'}(w') P_k^l(l), & \text{right} \end{cases}$$

which can be summarized as:

$$P_k^{x,w,l}(\langle x, w, l \rangle) = P_k^{x'}(x') P_k^{w'}(w') P_k^l(l)$$

where x' is a shortcut symbol which represents one among x, x^c and x^r.

Finally, we obtain the following general form for the matching probability:

$$P(b) = \sum_k s_k I(p;k) P_k^y(y) P_k^h(h) P_k^{x'}(x') P_k^{w'}(w') P_k^l(l)$$

Note that each P_k is a univariate distribution.

At this point we have to choose a form for each of the above distributions. We assume that the size attributes (w' and h) and the position attributes (x' and y) can be described as random variables with Normal Distribution (denoted by $N(\mu, \sigma)$). In a preliminary phase of our study, we considered using other distributions for the aforementioned random variables—in particular the Uniform Distribution and the Kernel Density Estimation (Parzen windows)—but we found the Normal Distribution models them better.

Concerning the textual content, $P_k^l(l)$ is the probability that the text l is the text of the matching block; P_k^l hence operates on text, differently from all other probabilities which operate on numbers. We assume that $P_k^l(l)$ can be expressed as a Markov chain of order 2. Its state space corresponds to the set of possible characters and 2 pseudo-characters representing the begin and end of the text. The probability $\mathcal{M}$ of the text l is defined as the probability of the state sequence corresponding to l. For example, the probability of the word `"two"` is given by $P(\texttt{"}\triangleright\,\texttt{t"}|\texttt{"}\triangleright\,\texttt{"})P(\texttt{"tw"}|\texttt{"}\triangleright\,\texttt{t"})P(\texttt{"wo"}|\texttt{"tw"})P(\texttt{"o}\triangleleft\texttt{"}|\texttt{"wo"})$, where $\triangleright$ and $\triangleleft$ represent the begin and the end of the text, respectively, and each transition probability corresponds to an element of the *transition matrix* T_k. The details about how we set T_k are given in Section 2.5.3. The final form for the matching probability is the following:

$$\begin{aligned} P \equiv \sum_k & s_k I(k) N(\mu_k^y, \sigma_k^y) N(\mu_k^h, \sigma_k^h) \\ & N(\mu_k^{x'}, \sigma_k^{x'}) N(\mu_k^{w'}, \sigma_k^{w'}) \mathcal{M}(T_k) \end{aligned} \tag{2.2}$$

where we omitted the function arguments for readability.

The wrapper, thus, merely requires the set of values for the parameters of Equation 2.2 (one set of parameters for each schema element). These parameters are summarized in Table 2.1. In the next section we describe how we generate the wrapper, i.e., how we choose these values.

We analyzed the impact of the various parameters on performance. The key result of the analysis, omitted for brevity, is that removing any one feature rendered the wrapper too inaccurate to be useful. In particular, text and position are a powerful combination of features but they are not enough.

Table 2.1: Parameters of the matching probability. All parameters are numerical except for the last one, which is a matrix.

Param.	Meaning
s_k	probability of $p = k$
$\mu_k^{x'}$	mean for the x position (x')
$\sigma_k^{x'}$	scale for the x position (x')
μ_k^y	mean for the y position
σ_k^y	scale for the y position
$\mu_k^{w'}$	mean for the character width w'
$\sigma_k^{w'}$	scale for the character width w'
μ_k^h	mean for the height h
σ_k^h	scale for the height h
T_k	transition matrix for text probability

2.5.3 Wrapper generation

A wrapper requires a set of values for the parameters described in the previous section. These values are selected based on the maximum likelihood method with respect to the distributions assumed in Equation 2.2. The maximum likelihood is computed based on the ground truth information stored in the knowledge repository for a few documents. The ground truth for a wrapper W and an element e takes the form of a set of *true blocks*, denoted $\mathcal{B}_e$. These values are initialized and updated as described in this section and the following one.

The procedure for generating a wrapper is as follows. We will describe the procedure for one single element, since the matching probability for an element can be generated independently from the matching probabilities for the other elements.

- We set s_k to the frequency of the true blocks in $\mathcal{B}$ whose page p is k.
- For each k-th Normal Distribution of Equation 2.2, we estimate its μ_k and σ_k parameters, using the corresponding attributes of the true blocks in $\mathcal{B}$ whose page p is k. In order to prevent overfitting, we impose a lower bound for the σ parameter, which we denote with σ_{xy} for the position attributes and σ_{wh} for the size attributes; σ is also set to this lower bound in case $\mathcal{B}$ contains only one element.
- For each k, we choose the x, w, l dependency which maximizes the probability of blocks in $\mathcal{B}$ whose page p is k.

After early experiments performed on a small portion of our dataset, we set our parameters as follows: $\sigma_{xy} = \sigma_{wh} = 0.05$ inches $= 1.27$ mm, $\epsilon = \frac{1}{3}$.

Finally, concerning the text probability $\mathcal{M}_k$, we perform some preprocessing on each piece of text l: (i) transform to lowercase, (ii) replace all digit characters with `"#"`, (iii) replace all space characters with standard space character, (iv) replace all punctuation characters with `"."` and finally (v) replace any other character with `"⋆"`. We denote the result with l'. Recall that we can describe a chain of order 2 using a transition matrix T of size $a \times a^2$, being a the number of states. In our case, given the aforementioned text elaborations, $a = 32$ and T indexes represent respectively a single character and a sequence of two characters: e.g., $t_{3,34} = t_{\texttt{"c"},\texttt{"ab"}} = P(\texttt{"bc"}|\texttt{"ab"})$. In order to set the T_k matrix for $\mathcal{M}_k$ we start with a frequency matrix F with the same size of T_k and each element set to 0. Then, we process the textual content l'^* of each true block $\mathcal{B}$ whose page p is k and increment the corresponding F elements. For example, after processing the sequence `"banana"` we will have $f_{\texttt{"a"},\texttt{"▷b"}} = f_{\texttt{"n"},\texttt{"ba"}} = f_{\texttt{"◁"},\texttt{"na"}} = 1$ and $f_{\texttt{"a"},\texttt{"an"}} = 2$. At the end, we set for each pair of indexes u, v:

$$t_{u,v} = \begin{cases} (1-\epsilon)\frac{f_{u,v}}{\sum_{z=1}^{a^2} f_{u,z}}, & \text{if } f_{u,v} > 0 \\ \frac{\epsilon}{a^2 - N_u}, & \text{otherwise} \end{cases} \tag{2.3}$$

where N_u is the number of $f_{u,v}$ which are greater than 0. We use the term ϵ to make the text probability smoother, i.e., so that it assigns non-zero (yet low) probabilities also to textual contents which are not present in the training set.

Wrapper generation example

We provide here an example of building a wrapper when $\mathcal{B}$ contains only one element. Figure 2.3 shows the top-right portions of two documents of our dataset. The documents share the same wrapper and represent invoices (see Section 2.6 for a description of our dataset). The true blocks for the `date` elements, i.e., set $\mathcal{B} = \{b_1, b_2\}$, are highlighted.

Values for blocks attributes follow (where applicable, values are expressed in mm):

$$\begin{array}{rcccccc} & p & x & y & w & h & l \\ b_1 = \langle & 1, & 54.9, & 56.5, & 23.5, & 6.2, & \texttt{"16/01/07"} \rangle \\ b_2 = \langle & 1, & 54.8, & 62.8, & 23.5, & 6.2, & \texttt{"11/01/07"} \rangle \end{array}$$

The values for the variables derived as explained in Section 2.5.2 are

hence computed by the system as:

$$\begin{array}{rcccccccc} & p & x & x^c & x^r & y & w' & h & l' \\ b_1 \rightarrow \langle & 1, & 54.9, & 66.6, & 78.4, & 56.5, & 2.93, & 6.2, & \texttt{"\#\#*\#\#*\#\#"} \rangle \\ b_2 \rightarrow \langle & 1, & 54.8, & 66.6, & 78.4, & 62.8, & 2.93, & 6.2, & \texttt{"\#\#*\#\#*\#\#"} \rangle \end{array}$$

At this point all parameters for the matching probability—see Table 2.1—can be computed. Concerning the page attribute, we have:

$$s_k = \begin{cases} 1, & \text{if } k = 1 \\ 0, & \text{otherwise} \end{cases}$$

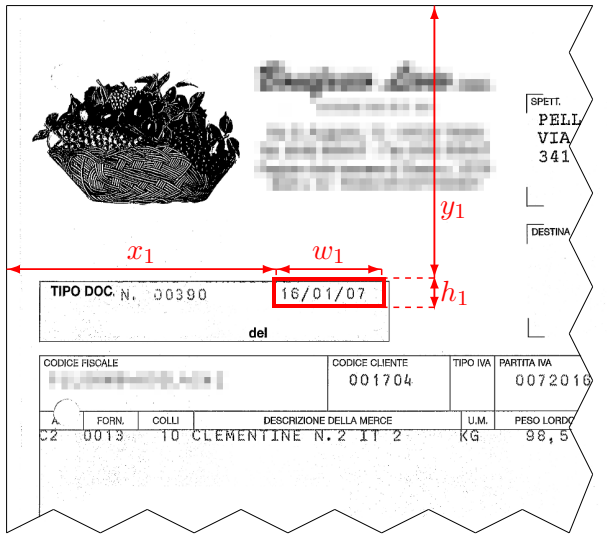
For the other parameters, only the values for $k = 1$ are needed, since all the true blocks lay on the first page. Concerning x', the system chooses to set $x' := x$, because the Normal Distribution fits the x values of $\mathcal{B}$ better, compared to the x^c and the x^r values—i.e., $\mathcal{B}$ true blocks of this example are justified to the left. The model parameter values are as follow (we omit the transition matrix for the sake of brevity):

$$\begin{aligned} \mu_1^{x'} &= 54.85 \\ \sigma_1^{x'} &= 1.27^{\dagger} \\ \mu_1^{y} &= 59.65 \\ \sigma_1^{y} &= 4.45 \\ \mu_1^{w'} &= 2.93 \\ \sigma_1^{w'} &= 1.27^{\dagger} \\ \mu_1^{h} &= 6.2 \\ \sigma_1^{h} &= 1.27^{\dagger} \end{aligned}$$

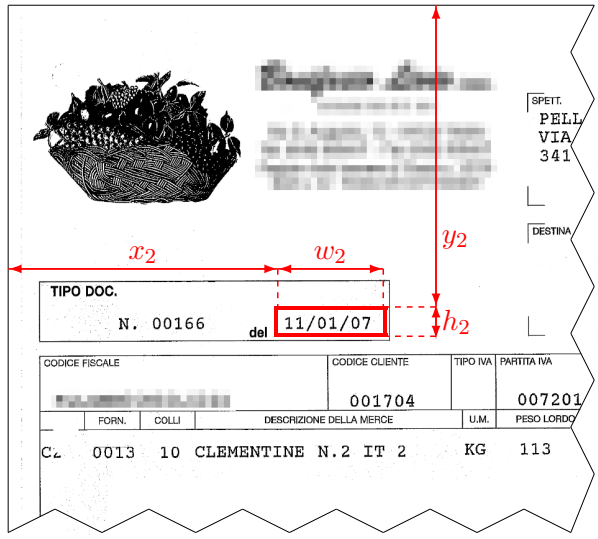
The values denoted by the symbol † are obtained by lower-bounding the corresponding computed σ estimates with the proper lower bound (i.e., σ_{xy} or σ_{wh}).

2.5.4 Human Intervention and Updates to the Knowledge Repository

The block selection stage may be configured to work in three different ways based on the value of a parameter H_{BL}, which can be one among Unsupervised, Semisupervised, and Supervised. Each value provokes a different behavior in terms of human intervention and determines whether wrappers are updated during the normal system operations, as described in full detail below. The value for H_{BL} can be selected independently



(a)



(b)

Figure 2.3: The portions of two documents of our dataset, belonging to the same class. The corresponding true blocks for the `date` element are highlighted and the attributes for the corresponding blocks are depicted (see Section 2.5.3). Note that sensible informations are blurred for privacy reasons.

of the value selected for the analogous parameter H_{WC} in the wrapper choice stage.

Wrappers are stored in the knowledge repository in the form of: for each element e, (i) a set of true blocks $\mathcal{B}_e$, and (ii) values for the parameters of the matching probability. The parameters of the matching probability are computed by using $\mathcal{B}_e$, as explained in the previous section. These parameters are recomputed whenever the composition of a $\mathcal{B}_e$ changes.

The set $\mathcal{B}_e$ contains up to w elements, where w is a system-wide parameter (we chose $w = 20$ according to the experimental findings of [73]). If there are less than w elements, parameters are computed based on all the true blocks available, otherwise one could choose the w true blocks of the schema element in many different ways. Our prototype uses the *last* w true blocks found by the system.

Operations in the block selection stage depend on whether the input wrapper W to use is empty or not. If it is not, processing proceeds as follows:

- $H_{\mathrm{BL}} = \mathsf{Unsupervised} \Longrightarrow$ Human intervention is never required. For each element e of the schema, the matching block b_e^* is added to the set of true blocks $\mathcal{B}_e$. If the set of true blocks $\mathcal{B}_e$ contains more than w elements, the oldest one is removed.

- $H_{\mathrm{BL}} = \mathsf{Semisupervised} \Longrightarrow$ Human intervention is required only for those elements whose set of true blocks B_e contains 4 elements or less. In this case, the system presents to the operator an image of the document d with the matching block b_e^* highlighted and offers two options:

  1. Confirm the choice made by the system.
  2. Select a different block.

  The GUI has been carefully designed in the attempt of speeding up the processing of each document as much as possible (Figure 2.5). In particular, the system highlights all the matching blocks, i.e., one block for each schema elements. If all the choices are correct, the processing of the entire document requires one single click on the "confirm" option. Otherwise, the operator corrects only the wrong blocks and then clicks on "confirm".

  The knowledge repository is then updated as in the Unsupervised case, with the true blocks either selected or confirmed on the GUI.

  If the set of true blocks $\mathcal{B}_e$ contains more than 20 elements, the oldest one is removed.

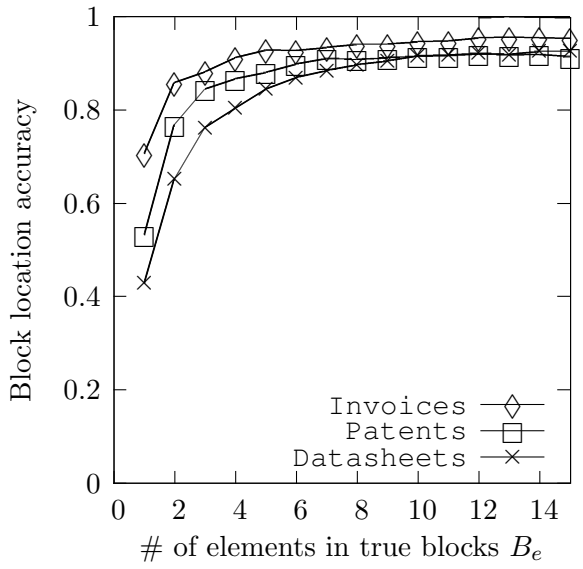


Figure 2.4: Block location accuracy vs. # of elements in true blocks B_e for the three datasets.

We chose upper limit of true blocks size (4 elements) performing a dedicated experiment: we evaluated the block location accuracy while varying the number of elements in the set of true blocks B_e. This experiment was performed with all our datasets (see Section 2.6.2); we can see in Figure 2.4 that the accuracy improvement with a number of elements greater then 4 is marginal.

- $H_{\text{BL}} = \textsf{Supervised} \implies$ Human intervention is always required. In this case, the system presents each document to the operator and processing proceeds as in $H_{\text{BL}} = \textsf{Semisupervised}$.

If the input wrapper W is empty, human intervention is always required irrespective of the value of H_{BL}. The reason is because, for each element, the set of true blocks is empty and hence the parameters have no values.

In this case the GUI (Figure 2.5) presents a document with no highlighted block and offers only the option of selecting the true block (i.e., there is no true block to confirm). Once the document has been processed by the operator, the selected true blocks are inserted into the respective sets. The processing of further documents associated with this wrapper will now proceed as above, based on the value of H_{BL}. In other words, the processing of the first document of a wrapper requires a human operator that points and clicks on the relevant information, hence allowing the system to generate the wrapper. The processing of further documents of

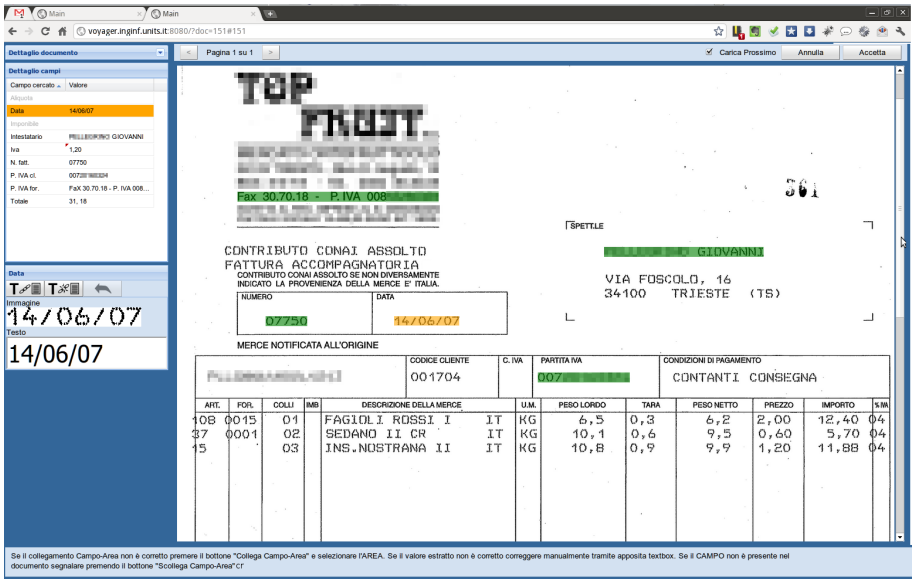


Figure 2.5: Block location GUI.

that wrapper may or may not require a human operator depending on the configured value for H_{BL}.

2.6 Experiments and Results

2.6.1 Prototype

We implemented PATO prototype according to the SaaS (software as a service) approach. Documents and results can be submitted and received, respectively, in batches with a programmatic REST interface. Operators interact with PATO through a browser-based GUI. We developed the back-end with Java Enterprise Edition 6 and used Glassfish as application server. We used the Google Web Toolkit framework (GWT) for the user interface and the open-source OCR CuneiForm 1.1.0.

All the experiments described in the next sections were executed on a quad core 2.33 GHz PC with 8 GB of RAM. The average time needed for choosing the wrapper for a document is 1.4 ms. The update of knowledge repository portion related to the wrapper choice stage takes, on the average, 2.1 s; most of this time is spent for retraining the SVM classifier. Regarding the block location stage, the average time for generating or updating a wrapper is 31.5 ms. The average time for locating an element on a document is about 135 ms. To place these figures in perspective, it suffices to note that the OCR execution of a single-page document takes about 4 s, whereas image preprocessing (binarization,

deskew) takes 2.4 s.

2.6.2 Dataset

A common problem in the research field of information extraction from printed documents is the lack of a reference testing dataset, which makes it impossible to compare results obtained by different research groups directly [66]. Following common practice, thus, we built three datasets:

- *invoices*: we collected a corpus of 415 real invoices issued by 43 different firms, i.e., with 43 different templates. The distribution of documents amongst firms is not uniform, the largest number of documents from the same firm being 80. This dataset is challenging because all documents were digitized after being handled by a corporate environment, thus they are quite noisy as they contain stamps, handwritten signatures, ink annotations, staples and so on. We defined a schema composed of 9 elements: `date`, `invoiceNumber`, `total`, `taxableAmount`, `vat`, `rate`, `customer`, `customerVatNumber`, `issuerVatNumber`

- *patents*: we collected 118 patents from 8 different countries which use different templates. We defined a schema composed of 10 elements: `classificationCode`, `fillingDate`, `title`, `abstract`, `representative`, `inventor`, `publicationDate`, `applicationNumber`, `priority`, `applicant`.

- *datasheets*: we collected 108 diode datasheets, produced by 10 different companies which use different templates. We defined a schema composed of 6 elements: `model`, `type`, `case`, `storageTemperature`, `weigth`, `thermalResistence`.

The invoices and a small portion of the other datasets were acquired at 300 DPI, while the remaining documents were born-digital. We manually constructed the ground truth for all the documents. An operator visually inspected each document and, for each element of the schema, manually selected the corresponding true block, if present. We made part of our dataset publicly available[1].

2.6.3 Experiments

The nature of the problem implies that the dataset has to be handled as a sequence rather than as a set: the results may depend on the specific order in which the documents are submitted to the system.

[1] http://machinelearning.inginf.units.it/data-and-tools/ghega-dataset (invoices are not included due to privacy concerns)

Based on these considerations, we performed 20 executions of the following procedure for each dataset: (i) we constructed a random sequence S with documents of the dataset; (ii) we submitted S to the system; (iii) we repeated the previous step 9 times, one for each possible combination of values for H_{WC} and H_{BL}. We hence performed 540 experiments.

As stated in previous sections, we selected values for the system-wide and wrapper-independent parameters after a few exploratory experiments on a small portion of the first dataset and left these values unchanged in all the experiments presented here.

In order to explore the trade-off between accuracy and automation level, we measured several quantitative indexes, as follows. We measured accuracy after the two salient workflow stages:

1. wrapper choice (WC) accuracy: the fraction of documents which are associated with the correct existing wrapper or that correctly lead to the generation of a new wrapper;

2. block location (BL) accuracy: the fraction of found blocks which correctly match the corresponding true blocks.

The block accuracy is a cumulative measure, i.e., it takes into account the effect of any possible error in the prior wrapper choice stage.

We estimated the automation level by counting, in each of the 540 experiments, the number of times that the system asked input from human operators. In particular, we counted the number of confirmations and the number of selections provided by the operators, i.e., the number of correct and of wrong suggestions, respectively. Input from human operators was actually simulated based on the ground truth.

We also cast the previous automation level results in terms of time required by human operators. To this end, we executed a further dedicated experiment for assessing the time required by each basic GUI operation, as follows. We selected a panel of 7 operators and a random sequence of 32 documents, one sequence for each dataset, the same sequence for all operators. We instrumented the system to measure the elapsed time for the processing steps of the operator and configured the system with $H_{\mathrm{WC}} = H_{\mathrm{BL}} = \mathsf{Supervised}$. Then, we asked each operator to process the sequence and collected the corresponding measurements. The results are in Table 2.2.

In order to obtain a baseline for these values and for the system as a whole, we also made a rough assessment of the time required by a traditional data entry procedure. For each dataset, we prepared a spreadsheet with the same schema as in the previous experiments and printed 10 documents selected at random from the dataset. Then, we

Interaction type	Time (s)
WC confirmation	5.24
WC selection	5.80
BL confirmation	3.45
BL selection	5.08

Table 2.2: Human interaction time (average time required to operators for each basic GUI operation).

asked each of the 7 operators to fill the spreadsheets starting from the printed documents. The average time for document turned out to be 89.23 s for the invoices documents, 50.9 s for the datasheets and 114.7 s for the patents. As will be shown in the next section, these values are significantly higher than the average time for document required by PATO, even in the configuration with maximal amount of operator involvement (i.e., $H_{\mathrm{WC}} = H_{\mathrm{BL}} = \mathsf{Supervised}$). We did not even attempt to translate the resulting estimates of time savings into monetary values because of the excessive number of factors that would be necessary for obtaining useful figures: labour cost, country, terms of service regarding accuracy and latency, pipeline structuring (e.g., web-based crowdsourcing or actual shipping of printed documents) and so on. Indirect but meaningful indications in this respect may be found on more specialized studies. In [59] the processing cost of a printed invoice has been estimated in about $13 per document and [89] reports that the number of printed paper forms which are handled yearly by Japanese public administration is greater than 2 billions. The economic impact of accountant personnel workload involved in generating and maintaining invoice wrappers for a widely adopted invoice recognition software is quantified in [91].

2.6.4 Results

Accuracy results averaged on the three datasets are shown in Table 2.3 for the accuracy and in Table 2.4 for the BL accuracy. The table contains one row for each combination of the system parameters H_{WC} and H_{BL}. The main finding is that block location accuracy is greater than 71% even in the most automated configuration. The experiments also demonstrate that less automated configurations indeed improve accuracy, as expected. Most importantly, even a moderate amount of operators' feedback in the block selection stage allows obtaining a block location accuracy consistently above 89%—it suffices to exclude the $H_{\mathrm{BL}} = \mathsf{Unsupervised}$ mode; please recall that $H_{\mathrm{BL}} = \mathsf{Semisupervised}$ involves the operator only when there are less than 5 examples for an element. It is worth pointing out also that the 100% accuracy in block location with $H_{\mathrm{BL}} = \mathsf{Supervised}$ could not be taken for granted in advance, as this accuracy index takes

H_{WC}	Accuracy		Interactions	
	Avg.	St. dev.	Conf.	Sel.
Unsupervised	72.72%	4.43%	0.00	0.00
Semisupervised	88.91%	5.58%	0.06	0.03
Supervised	100.00%	0.00%	0.90	0.10

Table 2.3: Average accuracy with standard deviation and number of human interactions per document for *wrapper choice* stage. Results are averaged on the three datasets.

H_{WC}	H_{BL}	Accuracy		Interactions	
		Avg.	St. dev.	Conf.	Sel.
Unsupervised	Unsupervised	71.23%	1.10%	0.00	0.67
	Semisupervised	89.11%	0.96%	1.77	1.14
	Supervised	100.00%	0.00%	4.94	1.70
Semisupervised	Unsupervised	74.06%	1.22%	0.00	0.71
	Semisupervised	91.59%	0.93%	1.94	1.15
	Supervised	100.00%	0.00%	5.16	1.52
Supervised	Unsupervised	75.98%	1.13%	0.00	0.68
	Semisupervised	92.66%	1.04%	1.84	1.08
	Supervised	100.00%	0.00%	5.10	1.41

Table 2.4: Average accuracy with standard deviation and number of human interactions per document for *block location* stage. Each row corresponds to a different combination for the automation level parameters H_{WC} and H_{BL}. Results are averaged on the three datasets.

into account any possible error in the previous wrapper choice stage. Indeed, the block location stage turns out to be quite robust with respect to errors in the wrapper choice stage: despite different accuracy levels of the wrapper choice stage, the block location accuracy is always quite high. It can also be seen, from the standard deviation of the accuracy (Table 2.3 and Table 2.4), that the system is robust with respect to the order in which documents are submitted.

These results cannot be compared directly to other earlier works, due to the lack of a common benchmark dataset (see previous section). However, it is interesting to note that, even in the most automated configuration, our numerical values are at least as good as those reported in earlier works focused on a static multi-source scenario for printed documents, e.g., [28, 49], even though we are coping with a much more challenging dynamic scenario.

Automation level results are also shown in Table 2.3, columns WC interaction and in Table 2.4, columns BL interaction. These columns

H_{WC}	H_{BL}	Interaction (s/doc)
Unsupervised	Unsupervised	3.40
	Semisupervised	11.89
	Supervised	25.71
Semisupervised	Unsupervised	4.14
	Semisupervised	13.05
	Supervised	26.06
Supervised	Unsupervised	9.19
	Semisupervised	17.56
	Supervised	30.49

Table 2.5: Number and time of human interactions per document for both wrapper choice and block location stage. Each row corresponds to a different combination for the automation level parameters H_{WC} and H_{BL}. Results are averaged on the three datasets.

show the average number of interactions required for each document, in each configuration. For example, with $H_{\mathrm{WC}} =$ Unsupervised and $H_{\mathrm{BL}} =$ Unsupervised, the operator is required to perform for each document, on the average, 0.67 block selections, no block or wrapper confirmation, no wrapper selection.

These figures are cast in terms of average human interaction time per document in Table 2.5, based on our measurements for WC/BL selection and confirmation (see previous section). Interestingly, the human processing time in supervised modes (25.71–30.49 s) is significantly smaller than 86.9 s, i.e., our estimate for the document processing time without our system (see previous section). In other words, our system may provide a practical advantage also in those scenario where perfect accuracy is required, since it can reduce the human processing time by 66% with respect to traditional data entry.

Data in Table 2.3 and Table 2.4 are averaged on the three datasets. Table 2.6 shows accuracy and time length of human interaction for each document, along with the traditional data entry processing time (baseline), separately for each dataset.

We remark again that in the field of information extraction from printed documents there does not exist any standard dataset benchmark. Our indexes, thus, cannot be compared directly to indexes obtained in other works. Keeping this fundamental problem in mind, though, it may be useful to note the values for block location accuracy reported in [49], as this is the only previous work encompassing both wrapper choice and generation in a (static) multi-source scenario. The cited work used a dataset composed of 923 documents of which 300 documents were used for training (we used 415 documents, with a training set of 14

Dataset	WC accuracy	BL accuracy	Interaction (s/doc)	Data entry (s/doc)
	$H_{\mathrm{WC}} = H_{\mathrm{BL}} = \mathsf{Semisupervised}$			
Invoices	90.03%	95.79%	13.51	89.23
Datasheets	86.17%	84.95%	8.39	53.02
Patents	87.50%	82.89%	15.70	118.45
	$H_{\mathrm{WC}} = \mathsf{Semisupervised}, H_{\mathrm{BL}} = \mathsf{Supervised}$			
Invoices	90.03%	100.00%	26.54	89.23
Datasheets	86.17%	100.00%	16.61	53.02
Patents	87.50%	100.00%	33.00	118.45

Table 2.6: Results for the three datasets with two combinations of H_{WC} and H_{BL} parameters.

documents). The reported block location accuracy is 80% and 75% for documents of known and unknown sources respectively.

2.7 Remarks

We have presented the design, implementation and experimental evaluation of a system (PATO) for information extraction from printed documents in a dynamic multi-source scenario. PATO assumes by design that the appearance of new sources is not a sort of exceptional event and is able to generate corresponding new wrappers: wrapper generation is supervised by the human operator, who interacts with the system quickly and simply, without the need of any dedicated IT skills. A crucial consequence of this assumption is that a wrapper must be generated using only information that can be extracted with a simple point-and-click GUI. Wrappers are indeed generated based on a maximum-likelihood approach applied on geometrical and textual properties of documents. PATO chooses the appropriate wrapper for a document based on a combination of two classifiers (k-NN and SVM) that operate on image-level features. The prototype allows offering the service on a cloud-based platform and may support crowdsourcing-based interaction modes.

We assessed the performance of PATO on a challenging dataset composed of more than 640 printed documents. The result are very satisfactory and suggest that our proposal may indeed constitute a viable approach to information extraction from printed documents. The system provides very good accuracy even in the most automated configuration. On the other hand, PATO can deliver perfect block location accuracy (i.e., 100% with our dataset) and, at the same time, reduce significantly the operators' engagement time with respect to traditional data entry.

Chapter 3

OCR error correction

3.1 Overview

The importance of the OCR errors for the quality of information extraction systems is demonstrated in [98] and [76], which quantify the negative impact of OCR errors. The experimental evaluation showed that the system presented in Chapter 2 is robust to OCR errors, that is, it is able to locate the correct information despite the presence of noise and characters misrecognition. However a system able to perform an automatic correction of these errors will allow the system presented in Chapter 2 to gain an improved accuracy.

We therefore propose a solution aimed at improving the accuracy of OCR-produced results on real-world printed documents. Our solution is tailored to environments in which the type of the information to be extracted is known, as in [18, 28, 73, 81, 6]. We aim at detecting and correcting OCR errors in the extracted information using knowledge about the type of each information (date, currency and so on).

The automatic correction of OCR errors is widely discussed in literature [61, 55, 106, 17], but is still an open research topic, expecially regarding the correction of text obtained from low quality or noisy printed documents. Traditionally, there are two main ways to detect OCR errors: dictionary lookup and character n-gram-matching [61, 55]. Other methods for automatic OCR errors correction, like the one described in [106] use topic models, which automatically detect and represent an article semantic context. In this work we propose a different approach based on the use of syntactic and semantic knowledge about the handled text that is often available, expecially in the information extraction systems of our interest.

Our approach is composed of two main steps: in the first step we

exploit domain-knowledge about possible OCR mistakes to generate a set of variants of the string extracted by the OCR. In the second step we perform a suite of syntactic and semantic checks to select from the correct string from the set of proposed ones. We also perform an aggregate semantic check in order to verify the correctness of two or more elements that are semantically linked to each other (e.g., total, taxable amount and vat in an invoice document). This approach is able to detect and correct OCR errors also in noisy documents without introducing errors.

As far as we know there is only one approach similar to the one here proposed, but tailored to a different application domain: [44] proposes an approach where a tree grammar, employed to define syntactically acceptable mathematical formulae, is used to detect and correct misrecognized mathematical formulae.

3.2 Our approach

3.2.1 System overview

Documents of our interest are typically electronic images of paper documents obtained with a scanner. A document is associated with a schema, as follows. The schema describes the information to be extracted from the document and consists of a set of typed elements e: for each element, the document contains zero or one value v.

For example, a schema could be `date`, `totalAmount`, `documentNumber`, respectively with types `date`, `currency` and `number`; a document with this schema could contain the values `"7/2/2011"`, `"23,79"` and no value for the respective elements.

Executing an OCR procedure on the electronic image of a document we obtain a set of strings $\{l_1, l_2, \ldots, l_n\}$. For each element e of the schema, we associate the candidate string l to that element. The system presented in the previous chapter can be used to perform the association between elements e and string l.

For each searched element, it may be $l \neq v$, because of the following reasons:

- l may contain v and **extra-text** that is not part of v. For example $l =$ `"date:21/12/2008"` while $v =$ `"21/12/2008"`.

- l may not contain v due to OCR errors. These errors can be of two types:

  - *segmentation error*: different line, word or character spacings lead to misrecognitions of white-spaces, causing segmen-

tation errors (e.g., l = "076 4352 056 C" while v = "0764352056C").

- *misrecognition of characters*: low print quality, dirt and font variations prevent an accurate recognition of characters (e.g., l = "|9,5SG" while v = "19,556" or l = "IOAS/0B127" while v = "105/08127").

While the segmentation and misrecognition problem may occur only with digitized documents, the extra-text problem may occur also with digitally born documents.

We propose a solution that uses a suite of syntactic and semantic checks in order to detect and correct these OCR-generated errors.

Our system is designed to be *modular* and *extensible*, so as to make it possible to augment and improve the domain-knowledge encoded in the module as well as to accommodate further application-specific rules beyond those currently embedded in the system. A high-level description of this step follows, full details are provided in the next sections.

For each element, we generate a set of values $\{v_1^*, v_2^*, \ldots, v_n^*\}$ that, due to OCR errors, might have lead to the extraction of l. This set is generated by applying to the extracted string l a number of possible substitutions as encoded in a predefined table of *substitution rules*. This table encodes a domain-knowledge about possible misrecognitions of characters. Then, we perform a suite of syntactic and semantic checks to exclude from the previous set all those values that do not satisfy at least one of these checks. We denote by V^* the resulting set of candidate values. These syntactic and semantic checks are encoded in boolean functions tailored to the type of the element to be extracted. We have implemented these functions for the following elements: `data`, `number`, `vatNumber`, `currency`, `fiscal code` (a unique id assigned to each person that lives in Italy, whose structure follows certain costraints [107]).

In addition to checks applied to individual elements, a check is performed also on groups of *correlated elements* whose values have to satisfy one or more joint conditions. For example, in our invoice scenario the group $\{\texttt{taxableAmount}, \texttt{vat}, \texttt{total}\}$ has to satisfy the condition $v_{\texttt{taxableAmount}} + v_{\texttt{vat}} = v_{\texttt{total}}$. Based on this observation, we introduced a semantic check on the coherency of the values for the elements in the group, as well as the ability to suggest a set $\mathcal{V}$ of further corrections in case the check is violated.

Finally, for each element we select a single value v_e^* that will be the proposed correction, as follows: if $\mathcal{V}$ contains one or more values for e the first one is selected, otherwise the first one contained in V_e^* is selected. As will be explained in the next sections, if $\mathcal{V}$ or V_e^* contain more than one value all have the same chance of being correct; the first one is chosen

and the system notify the operator about the remaining candidate values.

Our system is provides as output a qualitative evaluation of the proposed corrections, which is a sort of confidence level for the value associated with an element. There are three quality levels, low, medium and high, that are chosen based on the cardinality of V_e^* and of $\mathcal{V}$. We omit the details of the mapping for brevity. Using the quality for an element, we can decide whether to use that (possibly corrected) value automatically, or ask a confirmation to the operator. The operator is always involved when two or more equally likely corrections are selected. If the candidate string l, obtained from the initial OCR procedure, satisfies all checks, then no correction is attempted and an high quality level is given as output.

3.2.2 Single element

The workflow applied to each single element is shown in Figure 3.1 and discussed below. Algorithm 1 shows the corresponding pseudocode. As shown in Figure 3.1, our approach is based on components that may be extended as appropriate in a different application domain: the table of substitution rules and the functions $f^e_{\text{syntax}}(v)$ and $f^e_{\text{semantic}}(v)$ that implement syntactic and semantic checks on an element value v and are tailored to each type of element e. The table of *substitution rules* S is a set of pairs $s \to \mathcal{S}$, where s is a string and $\mathcal{S}$ is a set of strings. Each element of $\mathcal{S}$, say s', is a string visually similar to s, i.e., such that the OCR might have extracted s instead of the correct value s'. An example of substitution rule used in our system is `"1"` $\to$ {`"i"`, `"I"`}, which means that the OCR might extract the character `"1"` while in fact the correct character might be either `"i"` or `"I"`. Another example is `"11"` $\to$ {`"N"`}, which means that the extracted string might be `"11"` while the correct value might be `"N"`

In detail, the processing of each element is as follows.

1. The *string cleaning* stage removes from the input string l the characters that satisfy both the following conditions: the character cannot be present in v (according to f_{syntax}) and is not present in any left-side element of S.

2. The *substitution application* stage generates a set of candidate values V' applying all the substitution rules on the input string l'. For example, with l' = `"a1b1"` the substitutions can be:

   - *One-to-one*: for example, the rule `"1"` $\to$ `"i"` provokes the insertion in V' of the candidate values:

     {`"a1b1"`, `"aib1"`, `"a1bi"`, `"aibi"`}

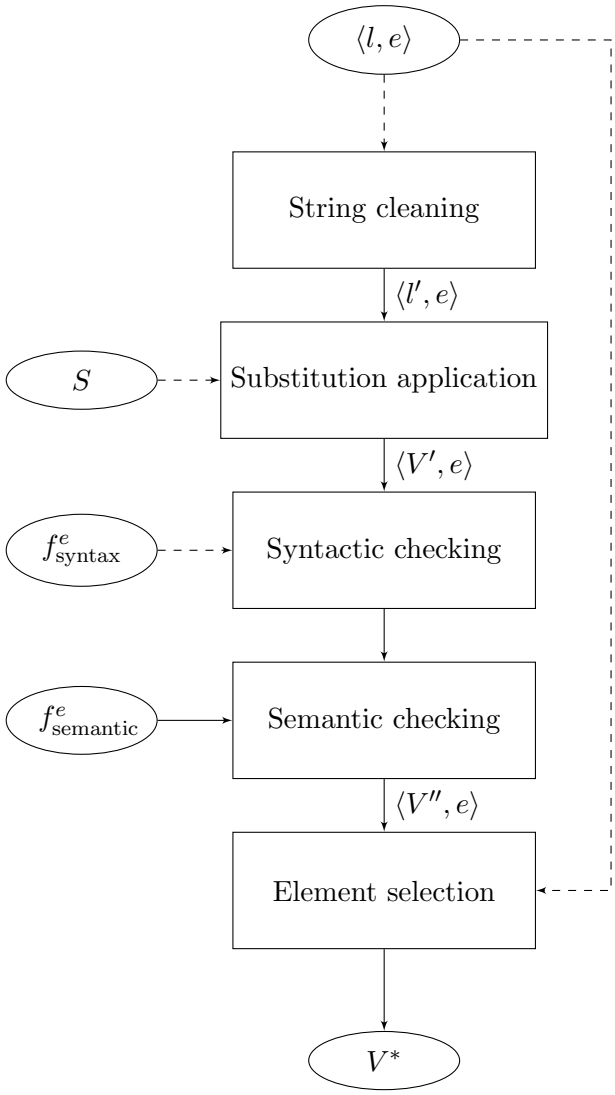


Figure 3.1: OCR error fixer workflow.

- *One-to-many*: for example, the rule `"1"` $\rightarrow$ {`"i"`, `"I"`} provokes the insertion in V' of the candidate values:

$$\{\texttt{"a1b1"}, \texttt{"a1bi"}, \texttt{"a1bI"}, \texttt{"aib1"}, \texttt{"aibi"}, \texttt{"aibI"}, \texttt{"aIb1"}, \texttt{"aIbi"}, \texttt{"aIbI"}\}$$

3. The *syntactic* and *semantic* checking stage removes from the set of candidate values V' those strings that do not satisfy one or both of the semantic checks for the element (joint checks applied to groups of correlated elements are discussed in the next section) and generates V''.

4. The *element selection* stage selects the strings with minimal distance from the extracted string l'. Since the Levenshtein distance gives the same weight to a character replacement and a character removal, we used a modified version of that distance (d_{L}) which give more weight to a character removal. This is done adding the length difference between strings to the original Levensthein distance.

If the final set of candidate values V^* is composed of more than one element, then the operator is asked to make the final choice. Otherwise, the operator may or may not be involved depending on the quality level assigned to the element, as outlined in the previous section.

3.2.3 Correlated element fixing

Given a group $E = \{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ of correlated elements, we introduce a joint semantic check encoded in a boolean function

$$f_{\text{semantic}}^{e_{i_1}, e_{i_2}, \ldots, e_{i_k}}(v_{i_1}^*, v_{i_2}^*, \ldots, v_{i_k}^*) \Rightarrow \{T, F\}$$

The joint conditions are used to provide further corrections: e.g., $v_{\texttt{vat}} = v_{\texttt{total}} - v_{\texttt{taxableAmount}}$.

Using the single element fixing algorithm we obtain, for each element e_i, a set of candidate values $V_{e_i}^*$. Using the sets $V_{e_i}^*$ we generate all the possible combination of values $\langle v_{e_{i_1}}^*, \ldots, v_{e_{i_k}}^* \rangle \in V_{e_{i_1}}^* \times \cdots \times V_{e_{i_k}}^*$.

Then we create a new set of tuple called $\mathcal{V}$ where we store all the combinations $\langle v_{e_{i_1}}^*, \ldots, v_{e_{i_k}}^* \rangle$ that satisfy the joint semantic check.

We also define k functions $\mathcal{F}_j(v_{e_{i_1}}^*, \ldots, v_{e_{i_j-1}}^*, v_{e_{i_j+1}}^*, \ldots, v_{e_{i_n}}^*) = \hat{v}_{e_{i_j}}$, i.e., having $k-1$ elements, the function $\mathcal{F}_j$ gives the j-th value $\hat{v}_{e_{i_j}}$ such that f_{semantic} is satisfied.

Using these functions, we can obtain a value $\hat{v}_{e_{i_j}}$ for at most one element e_{i_j} for which $V_{e_{i_j}}^* = \emptyset$.

Algorithm 1 Single element fixing pseudo-code

for e in $\{e_1, e_2, \ldots, e_n\}$ **do**
  $l_e :=$ extracted string for e
  $l'_e := f_{\text{cleaner}}(l_e)$
  $V'_e := f_{\text{substitutions}}(l'_e)$
  $V''_e := \emptyset$
  for v' in V'_e **do**
    if $f^e_{\text{syntactic}}(v')$ and $f^e_{\text{semantic}}(v')$ **then**
      $V''_e := V''_e \cup v'$
    end if
  end for
  $V^*_e := \emptyset$
  $d_{\min} := \min_{v'' \in V''_e} d_{\text{L}}(v'', l'_e)$
  for v'' in V''_e **do**
    if $d_{\text{L}}(v'', l'_e) = d_{\min}$ **then**
      add v'' to V^*_e
    end if
  end for
end for

In particular, in our scenario we have a single

$$E = \{\texttt{total}, \texttt{taxableAmount}, \texttt{vat}, \texttt{rate}\}$$

for which the joint semantic function is defined as follows (we omit the elements superscripts for ease of reading):

$$f(v^*_{\texttt{total}}, v^*_{\texttt{taxableAmount}}, v^*_{\texttt{vat}}, v^*_{\texttt{rate}}) =$$
$$= \begin{cases} T, & \text{if } v^*_{\texttt{total}} = v^*_{\texttt{taxableAmount}} + \\ & v^*_{\texttt{vat}} \wedge v^*_{\texttt{vat}} = \frac{v^*_{\texttt{taxableAmount}} * v^*_{\texttt{rate}})}{100} \\ F, & \text{otherwise} \end{cases}$$

Moreover, for the E described above we can define these functions:

$$\begin{aligned} \mathcal{F}_{\texttt{total}} &= v^*_{\texttt{taxableAmount}} + v^*_{\texttt{vat}} \\ \mathcal{F}_{\texttt{taxableAmount}} &= v^*_{\texttt{total}} - v^*_{\texttt{vat}} \\ \mathcal{F}_{\texttt{vat}} &= v^*_{\texttt{total}} - v^*_{\texttt{taxableAmount}} \\ \mathcal{F}_{\texttt{rate}} &= \frac{100 \cdot v^*_{\texttt{vat}}}{v^*_{\texttt{taxableAmount}}} \end{aligned}$$

Algorithm 2 Correlated element fixing algorithm

for each $\{e_{i_1}, e_{i_2}, \ldots, e_{i_k}\}$ **do**
  for $\langle v^*_{e_{i_1}}, \ldots, v^*_{e_{i_k}} \rangle$ in $V^*_{e_{i_1}} \times \cdots \times V^*_{e_{i_k}}$ **do**
    if $\exists j \mid V^*_{e_{i_j}} = \emptyset$ **then**
      $\hat{v}_{e_{i_j}} = \mathcal{F}_j(v^*_{e_{i_1}}, \ldots, v^*_{e_{i_j-1}}, v^*_{e_{i_j+1}}, \ldots, v^*_{e_{i_k}})$
      add $\langle v^*_{e_{i_1}}, \ldots, \hat{v}_{e_{i_j}}, \ldots, v^*_{e_{i_k}} \rangle$ to $\mathcal{V}$
    else
      if $f^{e_{i_1}, e_{i_2}, \ldots, e_{i_n}}_{semantic}(v^*_{e_{i_1}}, v^*_{e_{i_2}}, \ldots, v^*_{e_{i_k}}) = T$ **then**
        add $\langle v^*_{e_{i_1}}, \ldots, v^*_{e_{i_k}} \rangle$ to $\mathcal{V}$
      end if
    end if
  end for
end for

3.3 Experiments and results

3.3.1 Dataset

In order to assess our approach effectiveness we collected two real-world datasets: the first (D_1) composed by 591 values (extracted from 76 invoice documents) and the second (D_2) composed by 208 values (extracted from 37 invoice documents).

All documents belonging to the dataset were digitalized after being handled by a corporate environment, thus they are quite noisy with handwritten signatures, stamps, etc.; during the digitalization the pages were positioned in a variable way with respect to the scanner area, resulting in images whose content position is variable.

We defined a schema composed of 9 elements: `date`, `invoiceNumber`, `total`, `taxableAmount`, `vat`, `rate`, `customer`, `customerVatNumber`, `issuerVatNumber`.

The ground truth for the entire dataset was constructed using the defined schema. Each document image is converted to a set of values using an OCR system[1]. Then an operator inspected visually each document and, for each element of the schema, manually selected the corresponding string (i.e., l), if present, and wrote down the correct value (i.e., v).

The dataset D_2 was created with the purpose of testing our algorithm with low quality and dot-matrix printed documents. OCR performances on dot-matrix printed document is poor: a visual inspection of the dataset showed us that the OCR system introduced a sensible amount of errors, making the scenario highly challenging. In particular,

[1] CuneiForm 1.1.0, an open source document analysis and OCR system.

OCR → "R5/0 I/07"
v = "25/01/07"

(a)

OCR → "O1027eSO329"
v = "01027680329"

(b)

OCR → "QXOVANNX00"
v = "GIOVANNI**"

(c)

Figure 3.2: Examples of OCR errors; in particular, Figure 3.2(a) is a typical low quality dot matrix print.

approximately 55% of the strings recognized by the OCR system were wrong. Figure 3.2 shows some portions of documents that led to OCR errors.

3.3.2 Performance evaluation

The ground truth for this experiment is hence composed, for each document, of the following set of pairs:

$$L = \{\langle l_{e_1}, v_{e_1}\rangle, \langle l_{e_2}, v_{e_2}\rangle, \ldots, \langle l_{e_n}, v_{e_n}\rangle\}$$

Each pair associates the string l with the correct value v wrote down by the operator.

We applied our algorithm to each set L and counted the number of times when the output v^* was equal to v, i.e., the output was correct.

Table 3.1 and 3.2 shows the performance of our algorithm in both datasets. In particular, the tables show the percentage of correctly recognized values before (second column) and after (third column) the execution of our procedure.

Averaging these values we obtain the global percentage of correct values before and after the application of our algorithm. In dataset D_1 we increment correctness from 49.1% to 86.8%, while in dataset D_2 form 31.7% to 61.23%. In summary, our system is able to nearly double the percentage of correct values, also in a challenging low quality dataset.

Element	Correct before	Correct after	Total
`date`	46.05%	89.47%	76
`invoice number`	28.95%	55.26%	76
`vat number`	40.00%	88.89%	135
`taxable amount`	47.37%	90.79%	76
`total`	30.26%	85.53%	76
`vat`	57.89%	96.05%	76
`rate`	100.0%	100.0%	76

Table 3.1: Performance on dataset D_1

Element	Correct before	Correct after	Total
`date`	54.8%	77.4%	31
`invoice number`	46.1%	50.0%	26
`vat number`	19.12%	58.82%	68
`taxable amount`	46.4%	71.4%	28
`total`	43.4%	69.6%	23
`vat`	29.1%	45.8%	24
`rate`	50.0%	75.0%	8

Table 3.2: Performance on dataset D_2

It is important to notice that `vat number` and `total` are the elements with the greatest correctness increment in both datasets. The `vat number` structure follows certain constrains, and the great correctness improvement (44.2% in average) confirms the usefulness of semantic checks; similarly the correctness improvement of the `total` element (40.7% in average) confirms the usefulness of joint semantic checks.

3.4 Remarks

In this work we consider the problem of detecting and correcting OCR errors in an information extraction system. We propose an approach based on domain-knowledge about the possible misrecognition of characters to suggest corrections. We also propose a set of semantic and syntactic checks to validate the suggested corrections.

The system here proposed is able to nearly double the percentage of correctly identified values. These results have been obtained with a real-world dataset including a substantial amount of noise, as typically occurs when digitalizing printed documents previously handled by a corporate office. Despite the challenging dataset used in our experimental

evaluation, the obtained results are highly encouraging.

Future work will be devoted to extending further the generation of the corrections, possibly for allowing an automatic definition of the substitution rules.

Chapter 4

Textual document processing: RegEx generation

4.1 Overivew

Systems presented in Chapter 2 and 3 deals with weakly-structured documents: the structure is given by the geometrical position of text on the page and does play a role in the information extraction process. When the documents under analysis are mere text sequences, a more common approach is the use of regular expressions to perform information extraction. Indeed a regular expression is a means for specifying string patterns concisely. Such a specification may be used by a specialized engine for extracting the strings matching the specification from a data stream.

Regular expressions are a long-established technique for a large variety of textual document processing applications [99] and continue to be a routinely used tool due to their expressiveness and flexibility [23]. Regular expressions have become an essential device in broadly different application domains, including construction of XML schemas [19, 20], extraction of bibliographic citations [31], network packets rewriting [53], network traffic classification [101, 16], signal processing hardware design [96], malware [83, 32] and phishing detection [85] and so on.

Constructing a regular expression suitable for a specific task is a tedious and error-prone process, which requires specialized skills including familiarity with the formalism used by practical engines. For this reason, several approaches for generating regular expressions automatically have been proposed in the literature, with varying degrees of practical applicability (see next section for a detailed discussion). In this work we focus on text-extraction tasks and describe the design, implementation and experimental evaluation of a system based on genetic programming

(GP) for the automatic generation of regular expressions. The user is required to describe the desired task by providing a set of positive examples, in the form of text lines in which each line is accompanied by the string to be extracted, and an optional set of negative examples, i.e., of text lines from which no string has to be extracted. The system uses these examples as learning corpus for driving the evolutionary search for a regular expression suitable for the specified task. The regular expression generated by the system is suitable for use with widespread and popular engines such as libraries of Java, PHP, Perl and so on. It is important to point out that all the user has to provide is a set of examples. In particular, the user need not provide any initial regular expression or hints about structure or symbols of the target expression. Usage of the system, thus, requires neither familiarity with GP nor with regular expressions syntax.

Essential components of our implementation include the following. First, the fitness of individuals is based on the edit distance between each detected string and the corresponding target string. Several earlier works use a fitness based on the number of examples extracted correctly (see Section 5.2), but, as it turned out from our experiments, such a fitness definition is not adequate for this task. Second, we incorporate in the fitness definition a function of the size of the individual, in order to control bloating and obtain more readable results. Third, individuals are generated so as to make sure that each individual represents a syntactically correct expression.

We performed an extensive experimental evaluation of our proposal on 12 different extraction tasks: email addresses, IP addresses, MAC (Ethernet card-level) addresses, web URLs, HTML headings, Italian Social Security Numbers, phone numbers, HREF attributes, Twitter hashtags and citations. All these datasets were not generated synthetically, except for one: the Italian Social Security Numbers dataset. We obtained very good results for precision and recall in all the experiments. Some of these datasets were used by earlier state-of-the-art proposals and our results compare very favourably even to all these baseline results.

We believe these results may be practically relevant also because we obtained very good figures for precision and recall even with just a few tens of examples and the time required for generating a regular expression is in the order of minutes.

It seems reasonable to claim, thus, that the system may be a practical surrogate for the specific skills required for generating regular expressions, at least in extraction problems similar to those analysed in our evaluation.

A prototype of our system is publicly available at http://regex.inginf.units.it.

4.2 Related work

The problem of synthesizing a regular expression from a set of examples is long-established (e.g., [24]) and has been studied from several points of view. We restrict our discussion to evolutionary solutions and to recent proposals focussed on practical application domains of text-extraction.

An evolutionary approach based on grammatical evolution, i.e., a grammar-based genetic algorithm for generating programs written in languages specified with the Backus-Naur Form, is proposed in [29] and assessed on the extraction of hyperlinks from HTML files. The approach takes a set of examples in the form of text lines as input: a positive example is a text line from which some string has to be extracted and a negative example is a line from which no string has to be extracted. The cited work, thus, considers a *flagging* problem: a positive example is handled correctly when some string is extracted, irrespective of the string. We consider instead a more difficult extraction problem: in our case a positive example consists of a text line paired with a substring in that line; a positive example is handled correctly only when exactly that substring is extracted. We included in our experimental evaluation the dataset of [29]. Interestingly, our results improve those of the cited work even in terms of flagging precision and recall.

Problem and fitness definition in [10] and [46] are more similar to ours. The proposal in [10] applies a genetic algorithm for evolving regular expressions in several populations, followed by a composition module that composes two given regular expressions in several predefined ways and selects the composition which scores better on a validation set. The criteria for choosing from the final populations the two specific expressions to be input to the composition module are not given. The proposal is assessed in the context of web data extraction, in particular URLs and phone numbers. According to the authors, when applied to real web documents, the generated expressions are often not able to extract essential URLs components. A more direct application of genetic algorithms is presented in [46], which proposes to restrict the search space by selecting the symbol alphabet based on a preliminary frequency string analysis on a subset of the corpus. The approach is applied to URL extraction, but no details are given about size and composition of training and testing set.

Concerning evolutionary approaches based on genetic programming, the automatic induction of deterministic finite automata from examples was proposed in [37], whereas the generation of regular expressions was proposed in [97] and applied to the Tomita benchmark languages [100]. Stochastic regular expressions, also applied to the Tomita languages, were considered in [88]. Our approach follows the same lines of these

works, in that regular expressions are directly encoded as program trees. On the other hand, the computing power available today enable us to place much stronger emphasis on real-world text processing problems, with regular expressions suitable to be input to widespread engines such as Java, PHP and so on.

Concerning recent proposals focussed on text extraction, an active learning approach is explored in [108]. The application domain is criminal justice information systems and the main focus is minimizing the manual effort required by operators. Starting from a single positive example, human operators are introduced in the active learning loop in order to manually prune irrelevant candidate examples generated by the learning procedure. The approach is assessed on datasets with training set larger than the corresponding testing set—in our experiments the training set is a small fraction of the testing set. The cited work proposes an algorithm that may generate only *reduced* regular expressions, i.e., a restricted form of regular expressions not including, for example, the Kleen operator used for specifying zero or more occurrences of the previous string (e.g., "`a*`" means zero or more occurrences of the "`a`" character). Similar constraints characterize the learning algorithm proposed in [39]. This limitation is not present in the active learning algorithm proposed in [58], which requires a single positive example and an external oracle able to respond to membership queries about candidate expressions—the role played by human operators in the previous works. This algorithm is provably able to generate arbitrarily complex regular expressions—not including the union operator "`|`"—in polynomial time. No experimental evaluation is provided.

An approach that may be applied to a wide range of practical cases is proposed in [67]. This proposal requires a labelled set of examples and an initial regular expression that has to be prepared with some domain knowledge—which of course implies the presence of a skilled user. The algorithm applies successive transformations to the starting expression, for example by adding terms that should not be matched, until reaching a local optimum in terms of precision and recall. The proposal is assessed on regular expressions for extracting phone numbers, university course names, software names, URLs. These datasets were publicly available and we included some of them in our experimental evaluation. Another approach based on successive transformations, which also relies on an initial regular expression, is proposed in [7]. The main focus here is the ability to cope with a potentially large alphabet over noisy data. The requirement of an initial regular expression is not present in [23], which is based on the identification in the training corpus of relevant patterns at different granularity, i.e., either tokens or characters. The most suitable of these patterns are then selected and combined into a single regular

expression. This proposal is assessed on several business-related text extraction tasks, i.e., phone numbers, invoice numbers, SWIFT codes and some of the datasets in [67] (we included these datasets in our evaluation).

As a final remark we note that the earlier proposals more promising for practical text-extraction applications are *not* based on evolutionary approaches (i.e, [67, 7, 23]). Our experiments, though, exhibit precision and recall that compare very favourably to all these works.

Automatic generation of regular expressions from examples is an active research area also in application domains very different from text extraction, in particular, gene classification in biological research [62]. The algorithm proposed in the cited work is tailored to the specific application domain, i.e., extraction of patterns (mRNA sequences) with biological significance which cannot be annotated in advance. Our approach focusses on a radically different scenario, because we require that each positive example is annotated with the exact substring which is to be identified.

Automatic generation of an appropriate schema definition from a given set of XML documents is proposed in [19, 20]. These works develop probabilistic algorithms able to generate subclasses of regular expressions that, as shown by the authors, suffice for the specific application domain—e.g., expressions in which each alphabet symbol occurs a small number of times.

Automatic generations of regular expressions to be used for spam classification is described in [86]. The algorithm proposed in the cited work is trained by examples of textual spam messages paired with a regular expression generated by a human expert for flagging those messages as spam.

4.3 Our approach

4.3.1 User experience

The user provides a set of examples, each composed by a pair of strings $\langle t, s \rangle$ where t is a text line and s is the substring of t that must be extracted by the regular expression. A pair where s is empty, meaning that no string must be extracted from t, is a negative example.

The system generates a regular expression fully compatible with all major regular expression engines, including those of Java, Perl and PHP. The generated expression is not compatible with the JavaScript engines included in popular browsers, for reasons discussed below. However, the expression may be made compatible with JavaScript by means of a simple mechanical transformation [43] and our system is able to execute this transformation automatically.

We remark that the user should not provide any further information like an initial regular expression, a restricted set of important words, or hints about the structure of the example. Moreover, our approach does not required any knowledge about the syntax of the regular expressions or about the GP algorithm. As pointed out in the introduction, a prototype is available at http://regex.inginf.units.it.

4.3.2 Implementation

Every individual of the genetic programming (GP) search process is a tree τ. The *terminal set* consists of: (i) a large alphabet of *constants* including common characters and punctuation symbols, (ii) the numerical and alphabetical *ranges*, (iii) two common *predefined character classes*, i.e., "`\w`" and "`\d`", (iv) the *wildcard character* "`.`". Members of the terminal set are listed in Table 4.1, in which the Label column is the string that represents the corresponding node in the tree.

The *function set* consists of the regular expressions operators listed in Table 4.1: (i) the *possessive quantifiers* "*star*", "*plus*", "*question mark*", (ii) the *non-capturing group*, (iii) the *character class* and *negated character class*, (iv) the *concatenator*, that is a binary node that concatenates its children, (v) and the ternary *possessive quantifiers* "*repetition*". Labels of function set elements are *templates* used for transforming the corresponding node and its children into (part of) a regular expression. For example, a node of type "possessive question mark" will be transformed into a string composed of the string associated with the child node followed by the characters "`?+`". The string associated with each child node will be constructed in the same way, leaf nodes being associated with their respective labels as listed in Table 4.2.

A tree τ is transformed into a string R_τ which represents a regular expression by means of a depth-first post order visit. In detail, $R_\tau :=$ NODE2REGEX(ROOT(τ)), where the function NODE2REGEX is defined in Algorithm 3: CHILD(N, i) denotes the i-th child of node N and REPLACE(t_1, t_2, t_3) is a function that substitutes string t_2 with string t_3 in string t_1. A simple example is shown in Fig. 4.1.

Upon generation of a new candidate individual, the syntactic correctness of the corresponding candidate expression is checked. If the check fails, the candidate individual is discarded and a new one is generated. The GP search is implemented by a software developed in our lab. The software is written in Java and can run different GP searches in parallel on different machines.

We used a *fitness* function that implements a multiobjective optimization, minimizing: (i) the sum of the Levenshtein distances (also called *edit distances*) between each detected string and the correspond-

Node Type	Arity	Label
possessive star	1	“c_1`*+`”
possessive plus		“c_1`++`”
possessive question mark		“c_1`?+`”
non-capturing group		“`(`c_1`)`”
character class		“`[`c_1`]`”
negated character class		“`[^`c_1`]`”
concatenator	2	“c_1c_2”
possessive repetition	3	“c_1`{`c_2`,`c_3`}+`”

Table 4.1: Function set

Node Type	Labels
constants	“`a`”,...,“`z`”,“`A`”,...,“`Z`”, “`0`”,...,“`9`”,“`@`”,“`#`”,...
ranges	“`a-z`”,“`A-Z`”,“`0-9`”
predefined character classes	“`\w`”,“`\d`”
wildcard	“`.`”

Table 4.2: Terminal set

Algorithm 3 Transformation function from node N to regular expression R_N.

function NODE2REGEX(N)
  R_N := LABEL(N)
  if N is leaf **then**
    return R_N
  else
    for $i := 1$; $i \leq$ ARITY(N); i++ **do**
      N_C := CHILD(N, i)
      R_N := REPLACE(R_N,“c_i”, NODE2REGEX(N_C))
    end for
    return R_N
  end if
end function

Quantifier	Greedy	Lazy	Possessive
0 or more times	`*`	`*?`	`*+`
1 or more times	`+`	`+?`	`++`
0 or 1 time	`?`	`??`	`?+`
from m to n times	`{m,n}`	`{m,n}?`	`{m,n}+`

Table 4.3: Sample quantifiers

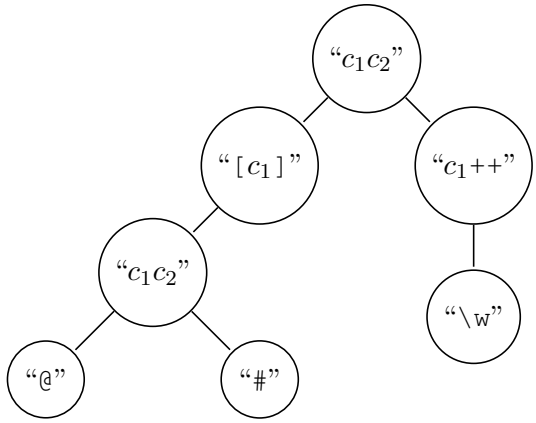


Figure 4.1: Tree representation of the “`[@#]\w++`” regular expression.

ing desired string, and (ii) the length of the regular expression. In detail, we defined the fitnesses $f_d(R)$ and $f_l(R)$ of an individual R as follows:

$$f_d(R) = \sum_{i=1}^{n} d(s_i, R(t_i)) \tag{4.1}$$

$$f_l(R) = l(R) \tag{4.2}$$

where: (i) t_i is the i-th example text line in a set of n given examples, (ii) s_i is the substring to be found in t_i, (iii) $R(t_i)$ is the string extracted by the individual R for the example t_i, (iv) $d(t', t'')$ is the Levenshtein distance between strings t' and t'', (v) $l(R)$ is the number of characters in the individual R—i.e., the length of the regular expression represented by that individual. The multi-objective optimization is performed by a *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) [36].

We remark that the fitness is *not* defined in terms of precision and recall, which are the performance metrics that really matter in the final result. Other prior works attempt to minimize the number of unmatched strings in the training corpus, thereby focussing more directly on precision and recall [62, 29]. Our early experiments along this line did not lead to satisfactory results. Looking at the generated individuals, we found that this approach tends to be excessively selective, in the sense that individuals failing to match just a few characters are as important in the next evolutionary step as those that are totally wrong. We thus decided to use the Levenshtein distance (along the lines of [10, 46]) and obtained very good results. A more systematic comparison between different fitness definitions is given in the experimental evaluation.

4.3.3 Observations

The choice of function set and terminal set has been influenced by the results of our early experiments, as follows. Regular expressions may include *quantifiers*, i.e., metacharacters that describe how many times a given group of characters shall repeat to be considered a match. Quantifiers can be grouped by their behaviour in three macro groups (Table 4.3): *greedy*, when they return the largest matching string, *lazy*, when they return the minimal match, and *possessive*, that are very similar to greedy quantifiers except that a possessive quantifier does not attempt to backtrack when a match fails. In other words, once the engine reaches the end of a candidate string without finding a match, a greedy quantifier would backtrack and analyse the string again, whereas a possessive quantifier will continue the analysis from the end of the candidate string just analysed. Since greedy and lazy quantifiers have worst case exponential complexity, we decided to generate individuals that include only possessive quantifiers.

This design choice has been corroborated by the results of early experiments in which we allowed individuals to include either greedy or lazy quantifiers. The execution time of these experiments was way too long to be practical—in the order of several tens of hours for generating a regular expression, as opposed to the minutes or few tens of minutes typically required when only possessive quantifiers are allowed (Section 5.5). Allowing regular expressions to contain only possessive quantifiers lead to results that cannot be handled by JavaScript engines included in major browsers. However, as pointed out in Section 5.4, a simple mechanical transformation—which consists in replacing each possessive quantifier with an equivalent expression composed of group operators and a greedy quantifier—makes the resulting expression compatible with JavaScript.

4.4 Experiments

4.4.1 Extraction tasks and datasets

We considered 12 different datasets, 2 of which are taken from [67], and [23] and other 2 are taken from [29]. We made our best to include in the evaluation all earlier proposals that address our problem. In this respect, it is useful to remark that our setting is more challenging than some of these works: (i) the proposal in [67] improves a regular expression initially provided by the user, whereas in our case the user does not provide any hint about the regular expression to be constructed; and, (ii) the proposal in [29] counts the numbers of examples in which a match has been found, irrespective of the content of the matched string—a flagging

problem. We count instead the number of examples in which exactly the searched string has been matched.

Each dataset corresponds to an extraction task, as described below, and we manually labelled all the data. The size of each dataset, including its composition in terms of number of positive and negative samples, is given in Table 4.5. A list of the datasets follows.

ReLIE URL Extract URLs from a collection of 50,000 web-pages obtained from the publicly available University of Michigan Web page collection [68] (used by [67]).

ReLIE Phone Number Extract phone numbers from a collection of 10,000 emails obtained from the publicly available Enron email collection [78] (used by [67, 23]).

Cetinkaya HREF Extract the HTML attribute HREF from the HTML source of a set of 3 web pages (used by [29]).

Cetinkaya URL Extract URLs from a set of 3 web pages (used by [29]).

Twitter Hashtag/Cite Extract hashtags and citations from a big corpus of Twitter messages collected using the Twitter Streaming API[1]; a superset of this corpus has been used in [72].

Twitter URL Extract URLs from a subset of the corpus used in the previous task.

Log IP Extract the IP addresses from a firewall log. These log were collected from our lab gateway server running the vuurmuur[2] firewall software.

Italian SSN Extract Italian SSNs[3] from a text corpus partly composed of synthetically generated examples including some form of noise, and partly obtained by OCR processing of low quality printed documents, mostly produced by dot-matrix printers. These documents were invoices issued by sixty different Italian dealers and have been used in [74].

Email Header IP Extract the IP addresses from the headers of an email corpus composed of 50 email collected from personal mail boxes of our lab staff. This task is more challenging than extracting IP addresses from a server log because email headers typically contain strings closely resembling to IP addresses, such as serial numbers, unique identification numbers or timestamps.

[1] https://dev.twitter.com/docs/streaming-apis
[2] http://www.vuurmuur.org/
[3] http://it.wikipedia.org/wiki/Codice_fiscale

Website Email Extract the email addresses from the HTML source of the address book page obtained from the website of a local nonprofit association.

Log MAC Extract the MAC (Ethernet card) addresses from the same log used in the *Log IP* task.

Website Heading Extract the HTML headings from the HTML source of a set of pages taken from Wikipedia and W3C web sites.

4.4.2 Methodology

We executed each experiment as follows:

1. We split the dataset in three subsets selected randomly: a *training* set, a *validation* set and a *testing* set. The training set and the validation set are balanced, i.e., the number of positive examples is always the same as the number of negative examples. Those sets are used as *learning corpus*, as described below.

2. We executed a GP search as follows: (i) we ran J different and independent GP evolutions (*jobs*), each on the training set (without the examples in the validation set) and with the GP-related parameters set as in Table 5.2; (ii) we selected the individual with the best fitness on the training set for each job; (iii) among the resulting set of J individuals, we selected the one with the best F-measure on the validation set and used this individual as the final regular expression R of the GP search.

3. We evaluated precision, recall and F-measure of R on the testing set. In detail, we count an *extraction* when some (non empty) string has been extracted from an example and a *correct extraction* when exactly the (non empty) string associated with a positive example has been extracted. Accordingly, the *precision* of a regular expression is the ratio between number of correct extractions and number of extractions; the *recall* is the ratio between number of correct extractions and number of positive examples; *F-measure* is the harmonic mean of precision and recall.

4. We executed each experiment with 5-fold cross-validation, i.e., we repeated steps 1–3 five times.

5. We averaged the results for precision, recall and F-measure across the five folds.

Parameter	Settings
Population size	500
Selection	Tournament of size 7
Initialization method	Ramped half-and-half
Initialization depths	1-5 levels
Maximum depth after crossover	15
Reproduction rate	10%
Crossover rate	80%
Mutation rate	10%
Number of generations	1000

Table 4.4: GP parameters

Task	Examples	Positive	Negative
ReLIE URL	3877	2820	1057
ReLIE Phone Number	41832	4097	37735
Cetinkaya HREF	3416	211	3205
Cetinkaya URL	1233	466	767
Twitter Hashtag/Cite	50000	34879	15121
Twitter URL	5300	2200	3100
Log IP	10000	5000	5000
Italian SSN	5507	2783	2724
Email Header IP	2207	480	1728
Website Email	25590	1095	24495
Log MAC	10000	5000	5000
Website Heading	49513	566	48947

Table 4.5: Dataset compositions

4.4.3 Results

We executed a first suite of experiments with a learning corpus size of 100 elements, 50 training examples and 50 validation examples, and $J = 128$ jobs. The learning corpus is always a small portion of the full dataset, around 1-4% except for the Cetinkaya URL task in which it is 8.1%. The results were very good in all tasks as we always obtained values of precision, recall, and F-measure around or higher than 90% (Fig. 4.2). The only exceptions are the precision indexes for Cetinkaya HREF and the ReLIE Phone precision. However, even these results constitute a significant improvement over earlier works as discussed in detail in the following.

Tasks ReLIE URL and ReLIE Phone Number have been used in earlier relevant works [67, 23]. We repeated our experiments with different sizes for the training set (as clarified in more detail below) and plotted the average F-measure of the generated expressions on the testing set against

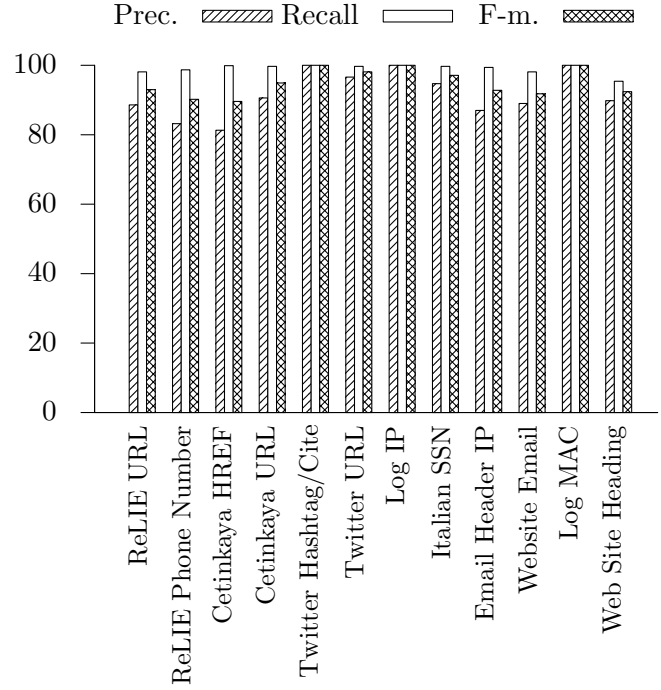
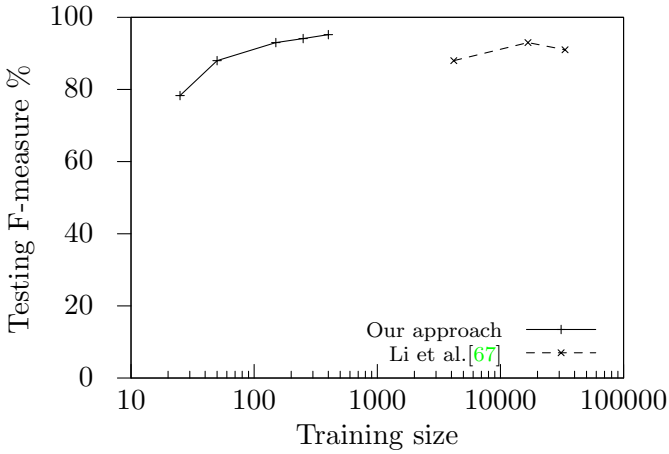
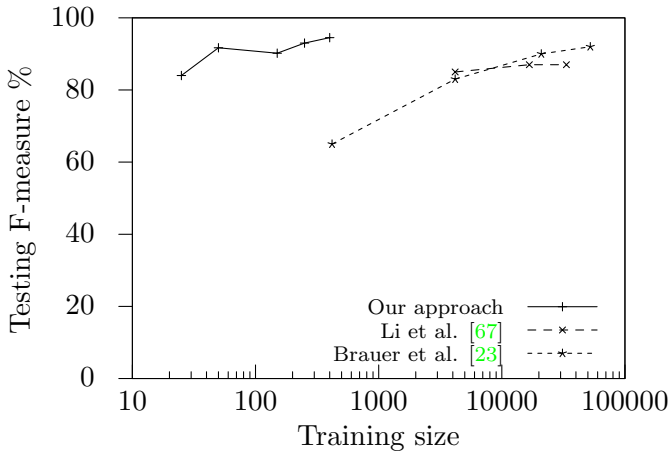


Figure 4.2: Experiment results, in terms of Precision, Recall and F-Measure, of each task



(a) ReLIE URL task



(b) ReLIE Phone Number task

Figure 4.3: Analysis of tasks ReLIE URL and ReLIE Phone Number. Performance comparison between our approach and earlier state-of-the-art proposals.

the training set size. The results are in Fig. 5.2(a) for ReLIE URL and in Fig. 5.2(b) for ReLIE Phone Number. The figures show also curves for the corresponding F-measure values as reported from the cited works. It seems fair to claim an evident superiority of our approach—note the logarithmic scale on the x-axis.

The performance indexes of our approach are obtained, as described in the previous section, as the average performance of the best expressions generated in each of the five folds, where the best expression for each fold is chosen by evaluating $J = 128$ individuals on the validation set. We analyzed *all* the 5×128 individuals that compose the final populations of the five folds and reported the corresponding performance distributions in Fig. 4.4 and Fig. 4.5 (learning set with 100 examples and $J = 128$, i.e., the experiment in Fig. 4.2). It can be seen that the very good performance that we obtain is not the result of a bunch of lucky individuals: our approach manage to generate systematically a number of different expressions with high values of precision, recall and F-measure.

Task	Dataset			Results (%)			Time
	Learn.	%	Train.	Prec.	Recall	F-m.	(min)
ReLIE URL	25	0.7	12	77.3	82.5	78.3	2
	50	1.3	25	79.9	98.1	88.0	4
	100	2.6	50	88.6	98.1	93.0	6
	250	6.4	150	89.7	99.0	94.1	10
	400	10.3	300	92.0	98.6	95.2	23
ReLIE Phone Number	25	0.1	12	80.9	90.9	84.0	2
	50	0.1	25	85.4	99.2	91.7	5
	100	0.2	50	83.2	98.7	90.2	7
	250	0.6	150	87.7	99.1	93.0	11
	400	1.0	300	90.2	99.1	94.5	28
Cetinkaya HREF	25	0.7	12	34.5	94.8	46.9	5
	50	1.5	25	72.2	94.4	81.6	10
	100	2.9	50	81.3	99.9	89.6	17
	250	7.3	150	85.6	99.2	91.8	30
	400	11.7	300	88.1	100.0	93.5	41
Cetinkaya URL	25	2.0	12	79.4	89.6	83.4	3
	50	4.1	25	87.6	98.4	92.7	7
	100	8.1	50	90.6	99.7	94.9	12
	250	22.3	150	95.0	99.8	97.3	22
	400	32.4	300	97.1	99.8	98.5	29

Twitter Hashtag/Cite	25	0.1	12	98.7	91.2	94.8	1
	50	0.1	25	99.1	95.6	97.3	2
	100	0.2	50	100.0	100.0	100.0	3
	250	0.5	150	99.9	100.0	100.0	8
	400	0.8	300	99.8	99.9	99.9	13
Twitter URL	25	0.5	12	95.4	99.5	97.3	1
	50	0.9	25	97.3	99.4	98.3	2
	100	1.9	50	96.6	99.7	98.1	7
	250	4.7	150	96.5	99.6	98.0	12
	400	7.5	300	97.4	99.4	98.4	24
Log IP	25	0.3	12	100.0	100.0	100.0	4
	50	0.5	25	100.0	100.0	100.0	7
	100	1.0	50	100.0	100.0	100.0	9
	250	2.5	150	100.0	100.0	100.0	7
	400	4.0	300	100.0	100.0	100.0	30
Italian SSN	25	0.5	12	95.6	99.6	97.6	1
	50	0.9	25	90.7	99.7	94.9	2
	100	1.8	50	94.7	99.7	97.1	2
	250	4.5	150	98.6	99.7	99.2	3
	400	7.3	300	98.5	99.6	99.1	6
Email Header IP	25	1.1	12	84.2	99.8	91.3	2
	50	2.3	25	86.1	99.4	92.4	4
	100	4.5	50	87.0	99.4	92.8	6
	250	11.3	150	89.5	98.1	93.6	9
	400	18.1	300	89.8	99.9	94.6	20
Website Email	25	0.1	12	75.3	99.2	81.0	2
	50	0.2	25	88.3	99.8	92.3	5
	100	0.4	50	89.0	98.1	91.8	7
	250	1.0	150	99.1	100.0	99.6	10
	400	1.6	300	99.1	100.0	99.6	23
Log MAC	25	0.3	12	100.0	100.0	100.0	4
	50	0.5	25	100.0	100.0	100.0	7
	100	1.0	50	100.0	100.0	100.0	10
	250	2.5	150	100.0	100.0	100.0	19
	400	4.0	300	100.0	100.0	100.0	29
Website Heading	25	0.1	12	79.9	100.0	88.7	6
	50	0.1	25	72.4	91.4	78.7	10
	100	0.2	50	89.8	95.4	92.4	15
	250	0.5	150	90.6	89.9	89.2	28
	400	0.8	300	92.7	100.0	96.2	42

Table 4.6: Experiment results with different learning size

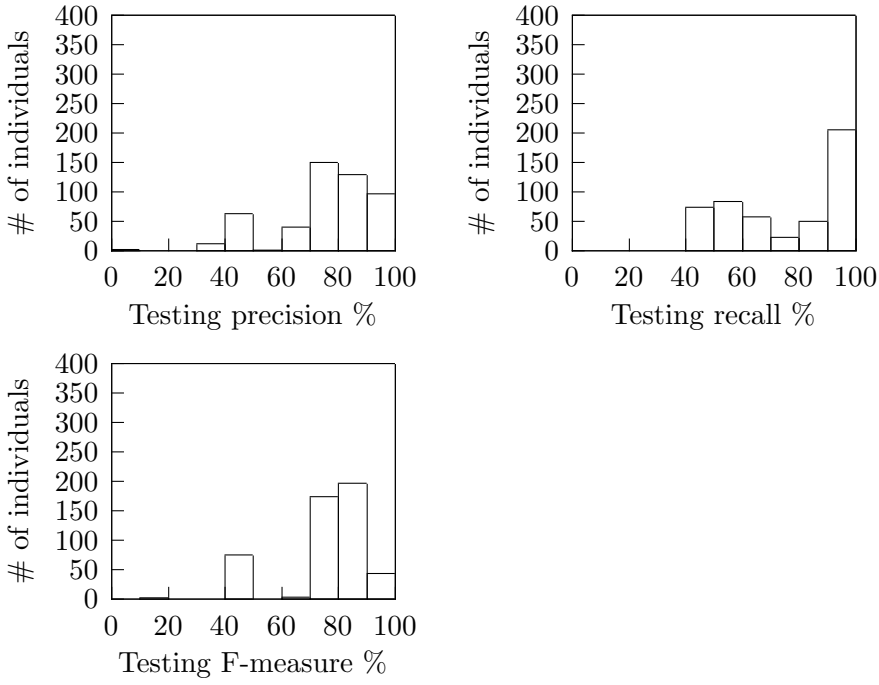


Figure 4.4: Distributions of precision, recall and F-measure on the testing set of ReLIE URL task.

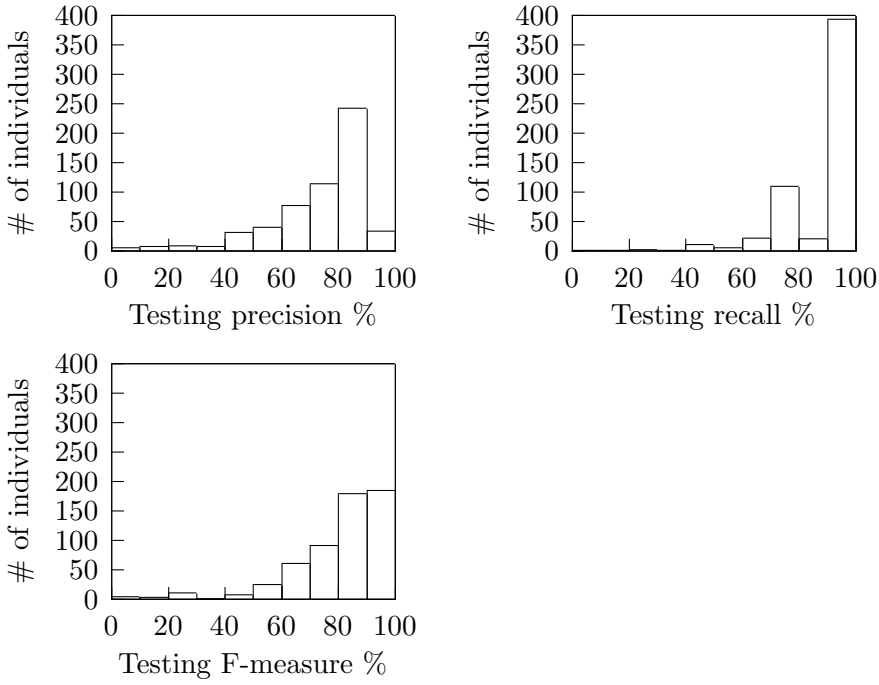


Figure 4.5: Distributions of precision, recall and F-measure on the testing set of ReLIE Phone Number task.

The datasets of tasks Cetinkaya HREF and Cetinkaya URL were also used in earlier relevant works in the context of a flagging problem [29]: a positive example is counted as correct when some string is extracted, irrespective of the string—an extraction problem simpler than ours. We assessed the performance of our result and of the regular expressions described in [29] according to this metric—i.e., we used all these expressions for solving a flagging problem on our testing set. The results are in Table 4.7. Our results exhibit better performance, which is interesting because: (i) the regular expressions in [29] were generated with 266 and 232 learning examples for the two tasks, whereas our result used 100 learning examples; (ii) our GP search aimed at optimizing a different (stronger) metric.

Having ascertained the good performance of the previous configuration, we investigated other dimensions of the design space in order to gain insights into the relation between quality of the generated expressions and size of the training set. We executed a large suite of experiments by varying the size of the learning set, as summarized in Table 4.6. This table reports, for each task, the number of learning examples, the percentage of the learning corpus with respect to the full dataset and the number of training examples. The rows with 100 learning examples are a duplicate of the previous configuration provided for clarity. It can be seen that the quality of the generated expression is very good in nearly all cases, even when the learning corpus is very small. Not surprisingly, for some tasks a learning corpus composed of only 25–50 examples turns out to be excessively small—e.g., Cetinkaya HREF. Even in these cases, however, enlarging the learning corpus does improve performance and 100 examples always suffice to achieve F-measure greater than 90%.

The table also reports the average execution time for each fold. We executed our experiments on 4 identical machines running in parallel, each powered with a quad-core Intel Xeon X3323 (2.53 GHz) and 2GB of RAM. Execution time is in the order of a few minutes, which seems practical. Indeed, although constructing the learning corpus is not immediate, the size of such a corpus is sufficiently small to be constructed in a matter of minutes as well. Most importantly, though, this job does not require any specific skills to be accomplished.

We also explored the possibility of reducing the number of jobs $J = 128$, in order to save computing resources. We repeated each of the experiments in Table 4.6 twice, with $J = 64$ and $J = 32$. We found that performance does not degrade significantly even when the number of jobs drops from 128 to 32—which roughly corresponds to dividing the execution time in Table 4.6 by four. In this perspective, we decided to set $J = 32$ in the prototype of our system available at http://regex.inginf.units.it.

Approach	Cetinkaya HREF	Cetinkaya URL
Cetinkaya [29]	99.97	76.07
Our approach	100.00	99.64

Table 4.7: Comparison between our approach and the one presented in [29] (flagging-based metric)

Task	Regular expression
Twitter Hashtag/Cite	`[@#]\w++`
Twitter URL	`\w++[^\w]*+\w\.\w\w[^#]\w*+`
Log IP	`\d++\.\d++\.\d++\.\d++`
Italian SSN	`([A-Z]{4,8}+(\w\w\w)*+[A-Z])*+`
Email Header IP	`\d*+\.\d*+\.\d*+\.\d*+`
Website Email	`(\-?+\w*+@*+\.*+\w++)*+`
Log MAC	`\w*+:\w*+:\w\w:\w\w:\w\w:\w\w`
Website Heading	`\<h[^X]*+`
ReLIE URL	`((\w++:)?+/*+\w++\.[a-z]\w([^1]\w)?+\w(\.([^1]\w++)++)?+)++`
ReLIE Phone Number	`([^\)]\d)++[^:][^:]\d++[^:]\d\d[^:]\d`
Cetinkaya HREF	`h[r][^\.]*+(([^1][^h][^1]*+\w*+[^1])*+)*+/*+(\.*+\w\w*+/*+[^1])*+\w*+`
Cetinkaya URL	`([/\w:]*+\.([^:][/\w\.]++)*+)`

Table 4.8: Regular expressions obtained with a training set of 50 elements. For each task, we report only the shortest expression among those obtained in the five folds.

We believe that our fitness definition plays a crucial role in determining the very good results. In order to gain further insights into this issue, we executed further experiments with different fitness definitions. First, we defined a linear combination of the objectives in Eqn. (5.1) and (4.2):

$$f(R) = \sum_{i=1}^{n} d(s_i, R(t_i)) + \alpha l(R) \tag{4.3}$$

Next, we focussed on the experiment of the Twitter URL task with learning corpus of 400 examples and executed this experiment with the following fitness definitions.

MO [*Edit*, *Length*] the multi-objective fitness function of our approach (Section 4.3.2).

MO [*Edit*, *Depth*] a multi-objective fitness function in which the length of the regular expression is replaced by the *depth* of the tree representing the individual.

Edit + α*Length* a linear combination of the objectives, with varying values for the α parameter (Eqn. 4.3);

Edit + α*Depth* the same as the previous definition, but using the depth of the tree instead of the length of the expression;

Errors a set of four fitness definitions obtained from the four above by counting the number of missed examples rather than the sum of the edit distances between each detected expression and the corresponding example.

The results are given in Table 4.9. We omitted the results of the experiments with fitness functions based on the number of missed examples (*Errors* in the above list) as they all exhibited precision and recall equal to zero. This analysis has three key outcomes. First, fitness definitions aimed at minimizing the number of missed examples do not work. Indeed, this observation is perhaps the reason why the earlier approaches shown in the Fig. 5.2(a) and Fig. 5.2(b) need a much larger training set. Second, when minimizing the sum of the edit distances, the various fitness flavours have essentially no effect on precision and recall, but they do have a strong impact on the complexity, and thus on readability, of the generated expression. Third, a multi-objective framework avoids the problem of estimating the linearisation coefficients, but a broad range of values for α provide expressions that are shorter and of comparable quality.

Finally, we show a sample of the expressions generated by our system in Table 4.8. The table has one row for each of the previous experiments

Fitness	α	Prec. %	Recall %	F-m. %	l
MO [Edit, Length]		97.39	99.47	98.42	54
MO [Edit, Depth]		97.72	99.63	98.67	150
Edit		97.41	98.50	97.05	285
Edit + αLength	0.01	97.40	99.50	98.44	30
Edit + αLength	0.10	97.36	99.50	98.42	28
Edit + αLength	1.00	97.36	99.50	98.42	28
Edit + αDepth	0.01	97.41	99.50	98.44	51
Edit + αDepth	0.10	97.67	99.50	98.58	55
Edit + αDepth	1.00	97.41	99.50	98.44	47

Table 4.9: Performance indexes and average length l of the resulting regular expressions for different fitness functions and values for the α parameter used for weighing the two objectives. MO indicates a Multi-Objective approach.

with training set of 50 elements. Each row shows the shortest expression generated across the corresponding five folds. The expressions have not been manipulated and are exactly as generated by our machinery.

4.5 Remarks

We have proposed an approach for the automatic generation of regular expressions for text extraction implemented with genetic programming (GP). The approach requires only a set of labelled examples for describing the extraction task and it does not require any hint about the regular expression that solves that task. No specific skills about regular expressions are thus required by users.

We assessed our proposal on 12 datasets from different application domains. The results in terms of precision and recall are very good, even if compared to earlier state-of-the-art proposals. The training corpus was small, in a relative sense (compared to the size of the testing set), in an absolute sense and in comparison to earlier proposals. The execution time is sufficiently short to make the approach practical.

Key ingredients of our approach are: (i) a multi-objective fitness function based on the edit distance and the length of the candidate regular expression, (ii) the enforcement of syntactical and semantic constraints on all the individuals constructed during the evolution, (iii) the choice of speeding up fitness evaluation by constructing individuals that may include only possessive quantifiers.

Although our approach has certainly to be investigated further on other datasets and application domains, we believe that our results are highly promising toward the achievement of a a practical surrogate for the specific skills required for generating regular expressions, and significant

as a demonstration of what can be achieved with GP-based approaches on contemporary IT technology.

Chapter 5

Structured document processing: DTD generation

5.1 Overview

The eXtensible Markup Language (XML) is a markup language for encoding data in a format which is both human-readable and machine-readable. XML *documents* consist of Unicode text and can represent arbitrary data structures. Although XML documents may be produced and/or consumed also by humans, in practice XML is widely used in machine-to-machine interaction, especially on the web.

Practical applications may impose specific encoding constraints on the data to be exchanged and these constraints take the form of a *schema*. Schemas are specified in a *schema language*, the most widely used schema languages being those used in *Document Type Definitions* (DTD) [103] and XML Schema Definitions (XSD) [104]. Schemas provide human users with a conceptual description of the data contained in XML documents and, most importantly, greatly facilitate automated processing. For example, availability of the expected schema for a given XML document allows machine consumers to validate input data automatically. In fact, unvalidated input data from web requests is a very important and pervasive class of security vulnerabilities in web applications.

XML documents are not required to refer their schema, however. Although the presence of schemas constitutes an important advantage, a large portion of XML documents found in the wild actually does not refer any schema [75, 8]. Furthermore, even when XML documents do refer a schema, the schema is often unavailable (e.g., it has been moved to another web site) or incomplete. For example, the DTD schema for

requests and responses to the OpenStreetMap API[1] is stated to be "incomplete", yet made available by the organization itself who defined and primarily uses the corresponding XML documents.

For these reasons, the need arose for methods capable of generating and maintaining good-quality schemas from existing XML data, in order to allow more effective production, processing and consumptions of XML encoded data [42]. In this chapter, we propose the design, implementation and experimental evaluation of a tool based on Genetic Programming (GP) for generating DTD schemas automatically. Our tool, which we called GP-DEI (Genetic Programming DTD Evolutionary Inferer), takes as input one or more XML documents and automatically produces a DTD schema which validates the input documents. Usage of the GP-DEI requires neither familiarity with GP nor with DTD or XML syntaxes. Our software is publicly available on our lab web site[2], along with the dataset used for its experimental evaluation.

The contribution of this work includes: (i) a way of encoding DTD element type declarations as trees that is suitable for a GP-based DTD schema generation; (ii) a set of multi-objective fitness definitions which allow obtaining a DTD which allow generating practically useful DTDs.

We performed an extensive experimental evaluation of our tool on a large collection of several sets of real world XML documents, including a portion of a dataset previously used in [19]. We compared a DTD generated by our tool against the real counterpart and we assessed the ability of GP-DEI to generate a DTD which, when used to validate corrupted XML documents, correctly detect wrong XML elements.

5.2 Related work

We are not aware of any evolutionary-based approach for inferring DTD schemas automatically from a set of examples; yet, there are some works on automatic generation of DTDs. The approach in [45] generates a DTD based on a sequence of induction steps. The proposed algorithm finds sequential and choice patterns in the input data, performs a form of factorization and generalization, and finally applies a Minimal Description Length principle. The approach in [19] is based on the observation that content models in DTDs contain large alphabets but every alphabet symbol occurs only a small number of times. Moreover, certain restricted forms of regular expressions suffice to represent DTDs, i.e., single occurrence regular expression (SORE) and chain regular expression (CHAREs). The approach proposes a learning algorithm that

[1] http://wiki.openstreetmap.org/wiki/API_v0.6/DTD, visited in November 2012
[2] Obscured due to double blind review.

automatically infers the latter based on a set of examples.

Use of spanning graphs is proposed in [79]. The XML input documents are converted into document trees, then all trees are merged in a spanning graph. The conversion from the final spanning graph into the DTD form is performed by applying a set of heuristic rules. It is also possible to perform a relaxation of the generated DTD according to a parameter specified by the user. A variation to this approach is proposed in [77], where a restricted content model is inferred from each element and some heuristic rules are applied to the merging procedure for generating the spanning graph. The algorithm proposed in [92] is based on converting XML documents to an entity-relationship model, which is done by extracting semantics information from the input documents, like cardinalities among entities and parent-child relationship.

The use of the DTDs goes beyond the document validation, therefore a tool able to infer DTDs from a set of XML documents could be useful in other application domains. For instance in [102] the DTDs are used to automatically create tables definition in a relational data base. A method for generating XForms and XSL code automatically starting from a DTD, in order to reduce development cost for coding forms, is presented in [64].

Since a DTD expression is a form of regular expression, as will be discussed in section 5.3, our approach, for each element in the documents, tries to find a suitable regular expression. In literature the problem of learning a regular expressions from a set of examples is long-established (e.g., [24]) and has been studied from several points of view. For example, [39] provides a learning algorithm for finite unions of pairwise union-free regular expressions, and two approaches for generating regular expressions based on Genetic Programming are proposed in [11, 97].

Finally, the inference of XML Schemas (i.e., XSD, which are not addressed in this work) from examples is addressed in [34, 52, 77, 21].

5.3 XML and DTD

An XML document is a string of characters which are either markup or content: the former describes the structure of the information while the latter is the information itself. Markup constructs are known as *tags*, which can be of three types: start tags (e.g., `<author>`), end tags (e.g., `</author>`) and empty-element tags (e.g., `<reviewed/>`). Start tags and empty-element tags may include zero or more pairs of named values known as *attributes* (e.g., `<paper doi="10.1145/2330784.2331000">`). A tag can have at most one attribute with a given name. The portion of a document between a start tag and the first matching end tag (tags included) is an *element*. Each element may include other

markups, i.e., other elements. An XML document may easily be transformed into a tree in which each node corresponds to an element and the node descendants corresponds to the elements contained in the element. An example is in Figure 5.1-left.

The XML specification does not impose any constraints on: (i) the element names that may be present in an XML document; (ii) the content of the elements; (iii) the attribute names that may be present in the elements; (iv) the value of the attributes. Such constraints may be described in a *schema*, an external document which can be written using different schema languages and can be possibly referred by an XML document. An XML document that is both syntactically correct and conforms to its schema is said to be *valid*.

A widely used schema language is the one used in Document Type Definitions (DTD). A DTD is composed by a set of *element type declarations*, a set of *attribute type declarations* and possibly further constructs (entity declarations and notation declarations) that are rarely used and not addressed in this work.

An element type declaration defines an element and its possible content. The content is defined as: (i) `EMPTY`, which specifies that the element cannot have any content, (ii) `ANY`, which specifies that no constraints exist for the element content, or (iii) an *expression* e, which is a form of reduced regular expression over the alphabet A of element names allowed by the DTD.

Expression e can be defined with a context free grammar: $e :=$ `#PCDATA`, $e := a$, $e :=$ `(`e_1`,` $\ldots$ `,`e_n`)`, $e :=$ `(`e_1`|` $\ldots$ `|`e_n`)`, $e := e$`?`, $e := e$`+`, $e := e$`*`, where: (i) `#PCDATA` is text content ; (ii) $a \in A$ is an element name; (iii) the list operator `,` specifies that an element content must include all the given expressions, in the exact order; (iv) the choice operator `|` specifies that an element content must include exactly one of the given expressions; (v) the `?`, `+` and `*` quantifiers specify that the given expression must appear respectively 0 or 1 times, 1 or more times, 0 or more times.

An example is in Figure 5.1-right, which shows a DTD describing a set of constraints satisfied by the XML document on the left.

An attribute type declaration defines the possible attributes for a given element. The declaration simply consists of a list of triplets, each describing: (i) the name of the attribute, (ii) its data type or an enumeration of its possible values, and (iii) an indication of whether the attribute is required (`#REQUIRED`), optional (`#IMPLIED`) or has a fixed value (`#FIXED`)—the latter being accompanied by an actual value. Figure 5.1-right contains 2 attribute type declarations.

```
<dblp>
  <inproceedings doi="10.1145/2330784.23300">
    <author>Alice Anderssen</author>
    <author>Bob Bolder</author>
    <author>Chris Cage</author>
    <author>Dimitri Dizokulos</author>
    <author>Eric Egon</author>
    <author>Frank Fanluc</author>
    <title>
      Automatic generation of foo from bar
    </title>
    <year>2012</year>
  </inproceedings>
  <article doi="10.1016/0304-3975(86)9008">
    <author>Gerard Berry</author>
    <author>Ravi Sethi</author>
    <title>
      From regular expressions to
      deterministic automata
    </title>
    <year>1986</year>
  </article>
</dblp>
```

```
<!ELEMENT dblp(inproceedings|article)*>
<!ELEMENT inproceedings(author+|title|year?)>
<!ELEMENT article(author+|title|year?)>
<!ELEMENT author(#PCDATA)>
<!ELEMENT title(#PCDATA)>
<!ELEMENT year(#PCDATA)>
<!ATTLIST inproceedings
  doi   CDATA #REQUIRED
  venue CDATA #IMPLIED>
<!ATTLIST article
  doi   CDATA #REQUIRED>
```

Figure 5.1: An XML document (left) and a DTD (right) which validates the XML document.

5.4 Our approach

Our tool, named GP-DEI, takes a set of XML documents and generates a DTD. The output DTD includes *all and only* the element type declarations and the attribute list declarations which allow to validate the input documents. The tool has the following limitations: (i) generated DTDs does not include other DTD artifacts (as entity declarations and notation declarations) and (ii) generated attribute list declarations include only one data type (`CDATA`) and only the `#REQUIRED` and `#FIXED` keywords. These limitations do not prevent GP-DEI from generating DTDs that are useful in the vast majority of practical cases, though.

GP-DEI operates in three steps: (i) pre-processing of the documents, (ii) evolutionary generation of the schema, (iii) post-processing of the schema which produces a syntactically correct DTD.

5.4.1 Pre-processing

In this step, GP-DEI parses the input XML documents and produces two sets of *training corpora*: (i) a set τ_E which will be used for generating the *element type* declarations, and (ii) a set τ_A which will be used for generating the *attribute type* declarations. The set τ_E contains one training corpus E_e for each element name e which occurs at least once in the input documents. A training corpus E_e is generated as follows: (i) parse each XML input document d and, (ii) for each found element with name e in d, adds the sequence of e children names to E_e; in case e contains

Element name e	Character c_e	Training corpus E_e
dblp	a	{bf}
inproceedings	b	{ccccccde}
author	c	{C, C, C, C, C, C, C, C}
title	d	{C, C}
year	e	{C, C}
article	f	{ccde}

Table 5.1: Part of the result of pre-processing step, including the set τ_E (third column) built from the document in Figure 5.1. Character C is the character associated with #text during pre-processing.

textual content, the name #text is included in the sequence (this name is used only by our tool and has not any XML/DTD meaning). The set τ_A is built in the same way, except that each training corpus A_e contains the sequences of attribute names instead of children element names.

A further pre-processing step is then performed on τ_E and τ_A, as follows: (i) we associate each element name e with a single (globally unique) unicode character c_e, (ii) we associate each attribute name a with a single (globally unique) unicode character c_a, (iii) for each E_e, we replace each sequence of children element names with a string obtained by concatenating the corresponding associated characters, (iv) we repeat the previous step for each A_e and, finally, (v) we replace in each E_e string all character repetitions of three or more times with a repetition of two characters (e.g., aaa→aa, bbbb→bb). As an example, Table 5.1 shows the set τ_E (third column) built from the input document of Figure 5.1.

5.4.2 Expressions generation

Since a DTD element type declaration is a form of regular expression (Section 5.3), we generate a regular expression R_e for each element name e in the set τ_E. We generate regular expressions with a GP-based procedure similar to the one described in [11].

The *terminal set* varies for each element name e and consists of all the characters appearing in E_e. The *function set* consists of the following regular expressions operators: (i) the *possessive quantifiers* *+, ?+ and ++, (ii) the *non-capturing group*, (iii) the *concatenator*, that is a binary node that concatenates its children, (iv) and the *choice* operator |.

We use two fitness functions to be minimized: (i) the sum of the Levenshtein distances between each example string and the corresponding matched portion of the string, and (ii) the number of optional quantifiers (*+ and ?+) in the regular expression. More in the detail, we define the

fitnesses $f_d(R)$ and $f_c(R)$ of an individual R as follows:

$$f_d(R) = \sum_{i=1}^{|E_e|} d(t_i, R(t_i)) \quad (5.1)$$

$$f_c(R) = s(R) + q(R) \quad (5.2)$$

where t_i is the i-th element of E_e, $R(t_i)$ is the string matched by the individual R in t_i, $d(t', t'')$ is the Levenshtein distance between strings t' and t'', $s(R)$ is the number of `*+` quantifiers in R and $q(R)$ is the number of `?+` quantifiers in R. We use a multi-objective NSGA-II based optimization to drive the GP-search.

The choice of the fitness function in Equation 5.2 was made to reduce the bloating in the regular expressions: an optional element may be useless, increases the length of regular expression and can also allow the regular expression to match strings never seen in the training set, which may result in producing DTDs which are "too general". Therefore, we give an advantage to individuals containing a smaller number of optional quantifiers.

It is important to point out that we represent each element name with only one character in order to equally weigh the errors on all elements independently by their name length. If element names were represented in their native form, the fitness function in Equation 5.1 would penalize errors on elements with a longer name. Moreover, compressing example strings speeds up the evaluation of the individual, since the regular expression processor takes more time to analyze longer strings.

Regarding the generation of the attribute list declarations, we do not use an evolutionary approach. Instead, for each e, we check the presence of an attribute character c_a in all the strings s of A_e and, if c_a occurs in each s, we set a as required (i.e., `#REQUIRED`), otherwise, we set a as optional (i.e., `#IMPLIED`). We finally set all attributes data type as `CDATA`.

5.4.3 Post-processing

In this step, we transform each regular expression R_e obtained in the previous step in an equivalent DTD markup declaration for e, as follows: (i) remove useless parts of R_e—e.g., redundant parentheses or element repetitions, (ii) replace each single character representation c_e with its corresponding element name e, and (iii) convert R_e to the corresponding DTD element declaration according to DTD syntax—e.g., replace concatenations with the list operator `,`.

Regarding the attributes declaration, we generate a DTD declaration by replacing each single character representation c_a with the corresponding attribute name a.

5.5 Experiments

5.5.1 Datasets

We assessed our approach on a number of XML documents, that we grouped based on their origin, as follows:

Mondial a single XML document (1.8 MB) containing information on various countries used in [19];

Genetic a single XML document (683 MB) representing a protein sequence used in [19];

SVG a set of 10 XML documents (1.7 MB, globally), each containing a copyright-free image represented in SVG and collected by searching on Google Images;

OpenStreetMap a set of 20 XML documents (12 GB, globally) containing free geographic data and mapping of the Italian administrative regions, downloaded from OpenStreetMap[3].

RSS a set of 10 XML documents (278 KB, globally), each containing a Rich Site Summary (RSS) files, obtained from an online scientific library[4].

The datasets are publicly available on our lab web site[5]. For ease of discussion, all the results are grouped as above.

5.5.2 Methodology

For each XML document in a group, we generated a DTD as follows: (i) we executed the pre-processing described in Section 5.4.1 and obtained τ_E and τ_A; (ii) for each E_e in τ_E, we executed 8 different and independent GP evolutions (*jobs*) with the GP-related parameters set as in Table 5.2 using E_e as training corpus; (iii) we selected the individual R_e with the best fitnesses on the training corpus among the best individuals of each job; (iv) for each A_e in τ_A, we determinated which attributes are required or implied for e (see Section 5.4.2) (v) we transformed each R_e into a DTD element declaration through the post-processing described in Section 5.4.3; (vi) we obtained the DTD attribute list declarations through the post-processing described in Section 5.4.3.

[3] http://download.gfoss.it/osm/osm/
[4] http://ieeexplore.ieee.org/
[5] http://machinelearning.inginf.units.it/data-and-tools

Parameter	Settings
Population size	500
Selection	Tournament of size 7
Initialization method	Ramped half-and-half
Initialization depths	1–5 levels
Maximum depth after crossover	15
Reproduction rate	10%
Crossover rate	80%
Mutation rate	10%
Number of generations	200

Table 5.2: GP parameters

A *validation task* consists in applying a standard XML validator [6] on a pair $\langle d, D \rangle$. If a validation task does not terminate correctly, the validator indicates the number of errors found on d using the DTD D. We executed a number of validation tasks, as explained in the next section.

5.5.3 Results

The salient results are summarized in Table 5.4. The table provides: (i) number of XML elements in each XML document; (ii) time for pre-processing each XML document d; (iii) time for generating the corresponding DTD D_d (not including pre-processing); (iv) number of errors found by the validator using $\langle d, D_d \rangle$. The values are averaged across all documents in the same group. The key, important result is that each generated DTD indeed validates correctly the corresponding document. The time taken by the pre-processing step is clearly proportional to the number of elements in the input documents. Otherwise, the generation time is deeply influenced by the complexity of the DTD we aim to infer—i.e., the OpenStreetMap dataset, although is the biggest one in terms of elements, has quite simple structure and take less time to find an optimal solution.

In the next suite of experiments we investigated the generalization ability of our approach. For each document d in groups SVG, OpenStreetMap and RSS, we executed a validation task by using D_d on each of the other documents in the group. We measured the percentage of elements for which a validation error has been found, averaged across all validation tasks on each group. We obtained 0 errors for OpenStreetMap and RSS documents and an error rate of 22.7% for SVG group. In other words, for documents in groups OpenStreetMap and RSS, one single

[6] http://www.saxproject.org/apidoc/org/xml/sax/XMLReader.html#setFeature(java.lang.String,boolean)

Name	# docs	Elements	
		avg	sd
Mondial	1	22,423	-
Genetic	1	21,305,818	-
SVG	10	290	335
OpenStreetMap	20	7,343,984	7,758,557
RSS	10	244	66

Table 5.3: DTD datasets size

Name	Preprocess (s)		Generation (s)		Errors	
	avg	sd	avg	sd	avg	sd
Mondial	1.0	-	241.0	-	0.0	-
Genetic	1,078.0	-	897.0	-	0.0	-
SVG	0.1	0.0	6.8	6.3	0.0	0.0
OpenStreetMap	144.9	151.6	6.7	0.5	0.0	0.0
RSS	0.1	0.0	17.7	2.0	0.0	0.0

Table 5.4: DTD generation results

document suffices to infer a DTD suitable for validating every other document. This interesting outcome does not occur with SVG documents. We believe the reason is because the set of element names that may potentially be found in a SVG file is very large and a single SVG is unlikely to contain all those element names.

To gain insights into this issue, we investigated how many files are needed for inferring a DTD suitable for validating any SVG document. We focussed on documents in the SVG group and proceeded as follows: (i) we sorted the documents based on the number of contained XML elements, in increasing order; (ii) we chose the first $d_1, \ldots, d_n$ documents; (iii) we generated a DTD $D_{d_1,\ldots,d_n}$ from these n documents; (iv) we executed a validation task on each document in the group using $D_{d_1,\ldots,d_n}$. Figure 5.2 plots the percentage of elements that are validated correctly against the training set size, expressed as a percentage of number of elements with respect to all SVG documents.

We can see that using DTD D_{d_1}—which is generated only with the smallest document (corresponding to 0.27% of the training set)—we can correctly validate slightly more than the 90% of our dataset. Using a DTD $D_{d_1,\ldots,d_6}$—generated with the first six smaller documents (corresponding to the 9.6% of the training set)—we can reach the 95%.

Having ascertained the ability of our approach to indeed generate useful DTDs, we investigated its robustness when a generated DTD has to validate a corrupted XML—a generated DTD D_d which is not "too

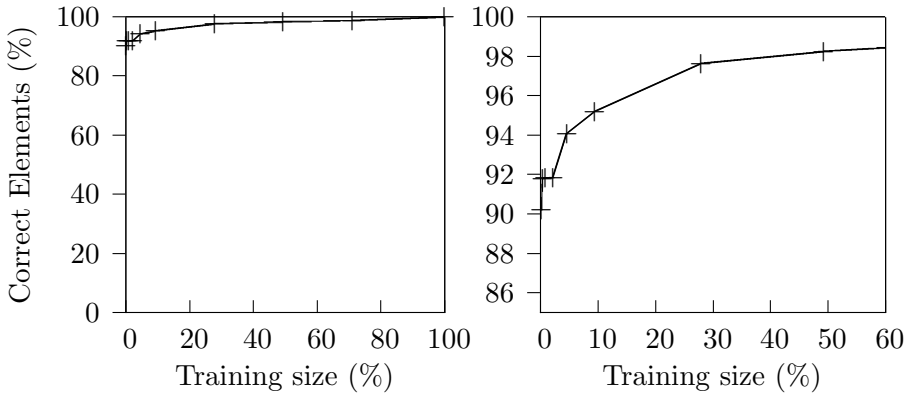


Figure 5.2: Generalization ability for SVG task (detail in the rightmost figure).

general", should be able to detect the corrupted elements. We focussed on the SVG group and proceeded as follows: (i) we generated ten DTD $D_{d_1}, \ldots, D_{d_{10}}$, one for each document of the group, (ii) we generated a new set $d'_1, \ldots, d'_{10}$ of corrupted SVG files, (iii) for each D_{d_i} and each d'_j, we executed a validation task.

The corrupted SVG files are generated using the following procedure: for each element of the document we randomly select one of these operation: (i) the element remains unmodified, (ii) the element is removed from the document, or (iii) the element is replaced with an empty element whose name is randomly selected from the DTD D_{SVG} defined in the SVG specification[7]. We define the corruption rate r for a corrupted document d'_i as ratio between the number of errors raised by running a validation task on $\langle d'_i, D_{\text{SVG}} \rangle$ and the number of elements in d'_i. The average corruption rate over all the 10 corrupted documents is 64.4%.

We assess GP-DEI performances on corrupted documents using false positive rate (FPR) and false negative rate (FNR), where a positive is a validated element. The former (FPR) is the ratio between the number of elements which are validated by D_i and are not validated by D_{SVG} and the number of elements which are not validated by D_{SVG}. The latter (FNR) is the ratio between the number of elements which are not validated by D_i and are validated by D_{SVG} and the number of elements which are validated by D_{SVG}. Table 5.5 shows the results obtained for each document used as input: each row corresponds to a DTD D_{d_i} and reports the number of element in d_i and the FPR and FNR averaged over $d'_1, \ldots, d'_{10}$.

[7] http://www.w3.org/Graphics/SVG/1.1/DTD/svg11.dtd

DTD D_{d_i}	Elements in d_i	FPR (%)	FNR (%)
D_{d_1}	4	29.3	28.3
D_{d_2}	8	25.6	16.7
D_{d_3}	16	12.7	20.2
D_{d_4}	38	16.5	15.3
D_{d_5}	74	12.8	18.1
D_{d_6}	143	12.7	17.5
D_{d_7}	548	20.8	19.7
D_{d_8}	640	17.1	18.6
D_{d_9}	645	9.8	18.0
$D_{d_{10}}$	864	13.7	16.1
Average		17.1	18.8

Table 5.5: Performances in presence of corrupted documents.

Original	Generated
`<!ELEMENT rss (channel)>`	`<!ELEMENT rss (channel)+>`
`<!ATTLIST rss version CDATA #FIXED "2.0">`	`<!ATTLIST rss version CDATA #REQUIRED>`
`<!ELEMENT channel (title | description | link & | language | item+ | rating? | image? | textinput? | copyright? | pubDate? | lastBuildDate | docs? | managingEditor? | skipDays?)*>`	`<!ELEMENT channel((title,link)| (((day,item+)|(month|year)) |(year|description)))+`
`<!ELEMENT title (#PCDATA)>`	`<!ELEMENT title(#PCDATA)>`
`<!ELEMENT description (#PCDATA) >`	`<!ELEMENT description(#PCDATA)>`
`<!ELEMENT link (#PCDATA) >`	`<!ELEMENT link(#PCDATA) >`

Table 5.6: Comparison between a portion of D_{RSS} and one DTD generated by GP-DEI.

Finally, in Table 5.6 we report a comparison between the DTD generated by GP-DEI for the first document in the RSS group and the Netscape DTD for RSS D_{RSS} found in the wild[8]. It is interesting to note that, as pointed out in Section 5.1, (i) D_{RSS} is no more available at the original URL (we refer to a copy) and (ii) our RSS documents (which we found in the wild) are not validated by D_{RSS}.

5.6 Remarks

We have proposed an approach for the automatic inference of DTDs with Genetic Programming. The approach requires only a set of XML documents to generate a DTD able to validate these documents. No

[8] http://web.archive.org/web/200012040847/http://my.netscape.com/publish/formats/rss-0.91.dtd

specific knowledge about the DTD and XML syntaxes nor any kind of supervision are required.

We assessed our proposal on 5 different dataset collected from different application domains. We verified that our tool generates DTD schemas which always correctly describe the input documents. We investigated the ability of our approach to generate schemas which may validate also similar documents and the robustness of our algorithm when a generated DTD has to validate a corrupted XML. Although our approach has to be investigated on other datasets, the results are highly promising and GP-DEI offers a practical solution to the problem of synthesizing a schema for a set of XML documents.

Chapter 6

Web Security

6.1 Analysis of Public Administrations Web Sites

The web is increasingly becoming one of the fundamental means for accessing government and public services. On the other hand, the web is also plagued by ICT-based security incidents at all levels: user devices, browsers, networks, server infrastructure [2]. An effective strategy for tackling Internet security threats is thus essential for fueling innovation and diffusion of e-government initiatives [60]. Indeed, it is not surprising that one of the seven goals of the European Digital Agenda is "enhance trust and security" [1].

A peculiar form of Internet security threats consists in the illegitimate modification of a web site that offers a public service. These risks are web-specific and have no counterpart in the traditional, i.e., "physical", access to public services. A common form of illegitimate modification is the *defacement*, i.e., the replacement of the original page with an entirely different page carrying political messages, offensive content and alike [13, 12]. In this case the user perceives immediately that the service cannot be used, thus the main effect of the attack is on the availability and utility of the application.

Other forms of attacks are aimed at remaining undetected by users and may take different forms:

a) Subtle changes aimed at injecting malware in browsers by exploiting software vulnerabilities [87].

b) Addition of new pages at URLs at which no page is supposed to exist. These pages are essentially defacements, except that they do not replace the original content and are visible only to users that know their URL. Attacks of this form are meant to be merely a proof

of the ability of the hacker. A significant fraction of the defacements archived in Zone-H[1] falls in this category.

c) Addition of illegitimate content aimed at deliberately manipulating search results—*search spam* [47]. The illegitimate content consist of links to the pages to be fraudulently promoted. The links may be added on existing pages or on pages created explicitly and solely to this purpose, at URLs at which no page is supposed to exist.

d) Modification of the site aimed at redirecting the browser to a site chosen by the attacker only when the user comes from a page returned by a search engine—*search redirection*. The relevance and diffusion of these attacks have been illustrated recently in the context of illegal drug trade [65].

Attacks of these forms are hard to detect because site administrators will likely never see any anomaly in the served content. Their effects on web sites of public interest may be very dangerous. Even leaving the malware injection threat aside, careful exploitation of the other forms of attack may create a very odd scenario: pages hosted on a trusted site that serve content fully controlled by attackers, tailored to the navigation path followed by users, visible only to certain users. An analogy with the physical world may illustrate the issue more clearly: when entering into the building of a public administration, one would not expect to find offices that are not supposed to exist and are visible only to certain citizens, perhaps depending on where they come from. Unfortunately, this is exactly what it could happen in web sites of public administrations. It is important to point out that HTTPS—the main and ubiquitous line of defense in sensitive web sites—does *not* provide any defense in this respect. HTTPS ensures secrecy, integrity and authentication by means of cryptographic techniques. The problem is, the server site is authenticated as a whole—any page coming from that site appears as being legitimate.

In this work we attempted to assess the ability of Italian public administration to be in full control of the respective web sites. We examined several thousands sites, including all local governments and universities, in search of evidence of attacks of categories c and d—a quick look at Zone-H will reveal that attacks of category b occur more or less routinely. We defined a methodology based on carefully constructed search engine queries for identifying pages that could be the results of those attacks, and inspection of those pages for filtering false positives out. We found that approximately 1.16% of the analyzed public administration domains

[1] http://www.zone-h.org

contains content that admittedly is not supposed to be there. Although these contents do not constitute an immediate threat to citizens, this result is not very encouraging and somewhat surprising. To place this result in perspective, we observe that a state-of-the-art system recently developed for efficiently searching malicious web pages, manages to construct a stream of URLs in which 1.34% of them identify malicious pages and this system improves earlier strategies by one order of magnitude [56]. While our results *cannot* be compared directly to those of the cited paper (the cited paper proposes a method for finding *only* the pages that distribute malware, i.e., attacks of category a, while we analyze *all* the web sites of public administrations and found compromised sites), it seems fair to claim that our results are indeed surprising. Besides, as clarified in the next sections, we could inspect just a few of the possible attack signatures, hence our data are very conservative. We believe that our analysis allows gaining useful insights into this novel and peculiar threat.

6.1.1 Our methodology

Preliminary Observations

Performing a full crawl of all the web sites of interest is clearly not feasible, even leaving aside the problem of discriminating between legitimate and illegitimate content. Moreover, a full crawl would not highlight search redirection attacks: in these attacks the fraudulent code on a compromised server identifies the requests to be redirected based on the HTTP request header, whose value identifies the page containing the link followed by the user—we would have to repeat the full crawl with differing values for such header, one indicating a click on a Google result page, another indicating an URL typed directly and so on.

For these reasons, we defined a methodology that may only search for a predefined set of attack signatures but can be implemented with moderate effort. As described in full detail in the next sections, the basic idea consists in querying search engines for the presence of certain words in the target sites. These words are chosen so as to be unlikely to be found in legitimate pages at those sites. The results provided by the search engines are then analyzed carefully so as to filter false positives out, i.e., to identify pages that are indeed fraudulent. Of course, this methodology cannot provide a full coverage of intrusions—querying search engines for *all* words or sentences that "should not be found" in legitimate pages is not feasible. Consequently, our analysis can only provide a partial view and conservative estimate of this phenomenon. On the other hand, we believe the results are indeed useful.

Pharmacy (W_F)	Illegal Drugs (W_D)
viagra	pills
cialis	marijuana
propecia	bong
zithromax	crack
doxycycline	cocaine
clomid	cannabis
levitra	lsd
nolvadex	heroin
lexapro	stoned
amoxil	opium
prednisone	koks
lasix	morphine
silagra	narcotic
tadalis	stimulant
zenegra	reefer

Table 6.1: Lists of target words that we used in our searches (we preferred to omit the Pornography list W_P).

Data Collection

We constructed a list D containing 5965 domains belonging to Italian local government administrations (municipalities, provinces, counties) and universities. Domains belonging to public administrations were obtained from a specialized web site[2] whereas those belonging to Universities were downloaded from "Ministry of Instruction, University and Research" web site. We compiled three lists W_F, W_P, W_D containing *target words* in three categories—pharmacy, pornography, illegal drugs—as follows. We initialized W_F with all the best seller drugs of an on-line shop[3]; W_P and W_D with all the blacklisted words in the parental filtering software Dansguardian[4]. Then, we removed all the non-word items (phrases, URLs, etc.) and performed a web search using the word as query and annotated the number of obtained results. Finally, we retained in each list only the 15 words with the highest number of results (see Table 6.1).

We generated two lists Q_{Bing} and Q_{Yahoo} of *search queries* as follows. For each domain d in D and for each word $w \in W_F \cup W_P \cup W_D$, we added to Q_{Bing} the search query `"site:`d w`"`. The `"site:`d`"` query portion instructs the search engine to include only results in the domain d. For each domain d in D and for each word list W_F, W_P, W_D, we added to

[2] http://www.comuni-italiani.it
[3] http://www.drugs-medshop.com
[4] http://www.dansguardian.org

Q_{Yahoo} the search query "`site:`d w_1 `OR ... OR` w_{15}", where $w_i, i \in [1, 15]$ is the set of words in the word list (i.e., 3 queries for each domain d). The "`OR`" keyword instructs the search engine to find web pages containing at least one of the words in the query. While the Bing search API can be used free of charge, usage of the Yahoo search API is charged on a per-query basis. For this reason, we constructed Q_{Yahoo} so as to search for several words at once.

We submitted each query in Q_{Yahoo} to the Yahoo search API and each query in Q_{Bing} to the Bing search API. We kept the first 50 results returned by each query. Using these results we compiled a list R where each element is a tuple $\langle d, W, u_{\text{SE}} \rangle$: d is the queried domain, W is the set of words contained in the query, u_{SE} is the URL returned by the query.

We obtained a list composed of 9459 elements: 6003 returned from the Yahoo API, 3456 from the Bing API. We found that 2305 results were obtained from both engines. We merged those duplicate items by setting the W value to the single word obtained from the Bing API and obtained a set R of 7154 elements, corresponding to 695 different domains.

We then collected additional data for each R element $\langle d, W, u_{\text{SE}} \rangle$, as follows. First, we performed 4 HTTP GET requests for each u_{SE}: 3 with the HTTP `Referrer` header set as if the request were generated by a user clicking on a search result obtained from one of the three major search engines (Google, Yahoo, Bing); 1 without including such header. For each GET request, we followed all redirections and saved the *landing URL* of the final web page $(u_{\text{direct}}, u_{\text{Google}}, u_{\text{Yahoo}}, u_{\text{Bing}})$ as well as the corresponding image snapshot $(\mathcal{I}_{\text{direct}}, \mathcal{I}_{\text{Google}}, \mathcal{I}_{\text{Yahoo}}, \mathcal{I}_{\text{Bing}})$.

Second, we associated each element with a single target word $w_a \in W$ as follows. We saved all the DOM trees obtained after the rendering of each landing URL. For elements obtained only from the Yahoo API we took the DOM tree obtained after the rendering of u_{Yahoo}, whereas for all the other elements we took the DOM tree after the rendering of u_{Bing}. We removed from the selected DOM tree: (i) all `script` and `style` HTML elements (along with their content) and (ii) all the HTML tags, hence obtaining a plain text string t which contains all the text rendered in the corresponding web page. We chose as w_a the first word in W found in t. In several cases we could not find in t any word in W, which may be an artifact of our procedure for choosing w_a but also of a change in the web site that had removed all words in W and had not yet been indexed by the search engine.

Data Analysis

At this point we analyzed the elements in R to determine whether the corresponding content was legitimate or contained evidence of an attack.

To this end we defined four categories, as described below. The analysis required visual inspection of each image snapshot, which was carried out in our lab by five lab staff members who were previously carefully instructed. We could examine a subset of R composed of 3209 elements selected at random.

- NORMAL: the landing URL belongs to domain d and the corresponding page does not appear compromised. Even if the operator detected one of the target words, he deemed its usage legitimate.
- FRAUDMODIFIEDPAGE: the landing URL belongs to domain d; the corresponding page appears compromised, yet part of the legitimate content is still present. The operator identified one of the following scenarios: (a) the page textual content includes a target word and its usage is clearly not legitimate; or (b) the page does not include any target word, yet it includes one or more images which are clearly visible and orthogonal to the page legitimate content.
- FRAUDNEWPAGE: the landing URL belongs to domain d; the corresponding page appears compromised, with no legitimate content apparently present (except for a a few graphical elements such as headers, navigation bar and alike).
- FRAUDOTHERSITE: the landing URL is unrelated to domain d, as a result of a redirection; the corresponding page appears compromised, i.e., totally unrelated to the content of d.

For the elements in which all landing URLs are identical ($u_{\text{direct}} = u_{\text{Google}} = u_{\text{Yahoo}} = u_{\text{Bing}}$), we inspected only $\mathcal{I}_{\text{direct}}$. For the other elements we inspected all the 4 image snapshots, assigned a category to each snapshot and take the most severe value as the category of the element (NORMAL being the least severe and FRAUDOTHERSITE the most severe).

Note that this categorization provides a *conservative* estimate of attacks, emphasizing precision over accuracy. In particular, a page containing fraudulent links could be categorized as being NORMAL.

6.1.2 Discussion

Key Insights

The main findings of our study are summarized in Table 6.2. We grouped the elements in R associated with the same target word w_a. The table contains a row for the 10 most frequently occurring words w_a, a row describing elements for which we could not find any w_a (labelled $\emptyset$),

and a row for all other target words w_a. The row indicating the total counts each domain only once, even when the domain appears in multiple rows. There are several elements containing the same domain-target word pair. We counted such elements only once and considered only the one associated with the most severe category, to simplify the analysis. In other words, Table 6.2 counts the number of *different* domains involved in each category.

The key result is that 400 of the 3209 URLs that we could inspect visually (FraudModifiedPage, FraudNewPage, FraudOtherSite) were actually compromised (12.5%). At the domain level, the compromised domains were 31 out of the 312 that we could analyze. Considering that we analyzed 312 on 695 domains and that no search results were returned for the remaining 5270 domains (which we hence conservatively deem as not compromised), this figure corresponds to 1.16%. As discussed in the final part of the introduction—and keeping in mind the corresponding caveats—this value is indeed surprising with respect to the value for [56], which is 1.12%.

An additional analysis on R elements for which $w_a = \emptyset$ suggests that the number of actually compromised domains could be higher than 31. These elements correspond to pages in which we did not find any target word in the corresponding rendered text. We performed a deeper analysis on a subset of these elements and analyzed the textual snippet that the search engine provided along with the URLs. We found that, for about half of the cases, the page appeared to be actually compromised but later restored—possibly partially. The restored (not compromised) version had not yet been indexed by the search engine. For the remaining cases the visual inspection of the snippet did not enable us to tell whether the page had been actually compromised (note that we did not inspect the DOM, hence we did not search for fraudulent links hidden from the visual content).

Redirections and Attack Categories

Table 6.2 also shows that, in our sample, redirection to external sites is less frequent than other forms of illegitimate content: there are 23 compromised domains in categories FraudModifiedPage and FraudNewPage, whereas there are 8 domains in FraudOtherSite.

Moreover, in the analyzed domains illegal drugs appear more frequently than other word categories. On the other side, the ratio between compromised and normal domains which contain a given target word tend to be higher for pharmacy words: e.g., 16 on 44 domains which include the word “viagra” were indeed compromised, whereas only 6 on 93 of domains which include the more frequent word “crack” were com-

w_a	Total		Normal		FraudModified Page	
	URLs	Domains	URLs	Domains	URLs	Domains
crack	404	93	371	87	3	2
marijuana	295	75	286	70	2	2
viagra	237	44	89	28	4	3
cannabis	210	74	210	74	0	0
pills	144	35	106	26	4	2
prednisone	117	30	113	27	1	1
lsd	84	41	84	41	0	0
cialis	72	27	38	17	6	2
morphine	66	13	66	13	0	0
bong	54	26	48	24	0	0
$\emptyset$	1213	242	1166	227	18	7
Other	313	77	232	67	2	2
Total	3209	312	2809	281	40	12

w_a	Total		FraudNew Page		FraudOther Site	
	URLs	Domains	URLs	Domains	URLs	Domains
crack	404	93	30	4	0	0
marijuana	295	75	7	3	0	0
viagra	237	44	25	5	119	8
cannabis	210	74	0	0	0	0
pills	144	35	34	7	0	0
prednisone	117	30	3	2	0	0
lsd	84	41	0	0	0	0
cialis	72	27	14	5	14	3
morphine	66	13	0	0	0	0
bong	54	26	6	2	0	0
$\emptyset$	1213	242	27	6	2	2
Other	313	77	54	7	25	1
Total	3209	312	200	11	160	8

Table 6.2: Full result set.

w_a	Total URLs	Total Domains	Normal URLs	Normal Domains	FraudOtherSite URLs	FraudOtherSite Domains
viagra	119	8	0	0	119	8
propecia	15	1	0	0	15	1
cialis	14	3	0	0	14	3
levitra	10	1	0	0	10	1
$\emptyset$	22	4	20	2	2	2
Total	180	9	20	2	160	8

Table 6.3: Search redirection results: elements for which the returned URL depends on the referrer of the HTTP request.

promised.

Table 6.3 focuses on search redirection, i.e., it considers only R items for which the returned URL depends on the referrer of the HTTP request ($u_{\text{direct}} = u_{\text{Google}} = u_{\text{Yahoo}} = u_{\text{Bing}}$ does not hold). Columns for FraudModifiedPage and FraudNewPage are not shown because we did not find any such values in the considered partition of R.

The main finding here is that most of the domains in this partition have been compromised: the partition is composed of 9 domains and 8 of them have been categorized as FraudOtherSite. In other words, when a redirection is performed basing on the referrer on a web site which contain a target word, the web site has been likely compromised. The 20 web pages categorized as Normal were error pages. We could not clearly tell whether the pages have been actually compromised: indeed, these pages could be the result of an attack that succeeded only in part.

It can be seen that we could find only 4 target words w_a and that all of them belong to W_F, the pharmacy category. In other words, all these compromised pages make the user coming from a search engine visit a pharmacy store.

Another interesting outcome is that, in our sample, search redirection does not affect all search engines equally: Table 6.4 describes which search engines actually trigger the referrer-based redirection. Google and Yahoo trigger all the 9 search redirection cases we found, while Bing triggers only 4 of them. We explain this difference because the formers are, or are perceived to be, more widely used than the latter and hence attackers concentrate their effort on the formers.

Table 6.5 shows the results on the other partition of R, i.e., it considers only R items for which the returned URL does not depend on the referrer of the HTTP request ($u_{\text{direct}} = u_{\text{Google}} = u_{\text{Yahoo}} = u_{\text{Bing}}$ holds). The column for FraudOtherSite is not shown because we did not find any such values in the considered partition of R. This result corroborates the observation made in the previous table, i.e., that referrer-based

Search engines	URLs	Domains
Google	180	9
Bing	74	4
Yahoo	175	9
Google + Yahoo	175	9
Bing + Yahoo	74	4
Google + Bing	74	4
Google + Bing + Yahoo	74	4
no redirection	3029	303

Table 6.4: Number of redirections as a function of search engines.

redirection on an external site is a likely indicator of compromise.

The main finding here is the disparity in the number of compromised domains and URLs in the two categories FraudModifiedPage and FraudNewPage. The two categories exhibit a nearly identical number of compromised domains, but there are much more compromised URLs in the FraudNewPage category than in FraudModifiedPage (this disparity is reflected also in the full set R, Table 6.2). We interpret this result as follows: once an attacker gains sufficient privileges to add a new illegitimate page on a CMS (Content Management System), he will likely exploit these privileges to add further illegitimate pages.

URL structure analysis

We investigated the structure of URLs that identify fraudulent web pages. We have found that, as expected, attackers tend to hide fraudulent web pages by placing them "deeply" into the target site.

In detail, for each URL u of a fraudulent page we determined its *sub-domain level* as follows: (i) we extracted the domain portion from the URL, say d_u; (ii) we removed from d_u the trailing string "www." (if present) and the domain obtained from the list D; (iii) we counted the number of dot characters and defined this value to be the sub-domain level of u. Consider for instance the domain name comune.udine.it; the URL www.comune.udine.it/hacked.html is at level 0 while segreteria.comune.udine.it/hacked.html is at level 1.

We have discovered that only 15.2% of fraudulent URLs are at level 0; 71.5% are at level 1 and 13.5% at level 2. In other words, the vast majority of fraudulent URLs (85%) are not at level 0: this result confirms that attackers indeed attempt to hide fraudulent pages—a fraudulent page at level 1 is harder to detect for the web site administrator than a page at level 0. From another point of view, prevalence of level 1 domains among fraudulent web pages could be due to the fact that the compromised sites

w_a	Total URLs	Total Domains	Normal URLs	Normal Domains
crack	404	93	371	87
marijuana	295	75	286	70
viagra	118	37	89	29
cannabis	210	74	210	74
pills	144	35	106	26
prednisone	117	30	113	27
lsd	84	41	84	41
cialis	58	24	38	17
morphine	66	13	66	13
bong	54	26	48	24
$\emptyset$	1211	241	1166	228
Other	288	76	232	67
Total	3029	311	2809	286

w_a	Total URLs	Total Domains	FraudModified Page URLs	FraudModified Page Domains	FraudNew Page URLs	FraudNew Page Domains
crack	404	93	3	2	30	4
marijuana	295	75	2	2	7	3
viagra	118	37	4	3	25	5
cannabis	210	74	0	0	0	0
pills	144	35	4	2	34	7
prednisone	117	30	1	1	3	2
lsd	84	41	0	0	0	0
cialis	58	24	6	2	14	5
morphine	66	13	0	0	0	0
bong	54	26	0	0	6	2
$\emptyset$	1211	241	18	7	27	6
Other	288	76	2	2	54	7
Total	3029	311	40	13	200	12

Table 6.5: Elements for which the returned URL does not depend on the referrer of the HTTP request.

are actually managed by smaller organizations (related to their parent public administration) which enforce weaker security IT policies.

We also determined the *path depth* of each fraudulent URL u: (i) we removed the two slashes after the protocol name; (ii) we removed the slash character at the end of the domain name; (iii) we counted the number of remaining slashes and defined this value to be the path depth of u. For example, segreteria.comune.udine.it/hacked.html has path depth 0, whereas segreteria.comune.udine.it/people/hacked.html has path depth 1. We counted 1.6 slashes on the average.

Finally, we found that the *URL length* of fraudulent pages is 77.1 characters on the average, while the URL length of the home page of the target sites is, on the average, only 24.45 characters.

Socio-demographic analysis

We performed a few additional analysis to understand whether there is any correlation between compromised sites and some non-technical features of the affected organizations. In particular, we analyzed population and geographical location of municipalities, provinces, and counties. For universities we analyzed number of students and position in a public ranking regarding the quality of the respective IT services. We did not find any significant correlation between compromised sites and these indexes. Of course, the number of compromised sites is too small to draw any statistically relevant conclusion in this respect. However, the lack of any meaningful pattern in this respect seems to be evident: compromised sites tend to be equally distributed in small or large municipalities and universities. Compromised university web sites are scattered more or less randomly across the ranking, going from the 1st to the 51st position. Indeed four of the eleven universities at the top of this ranking have a compromised web site. The geographical position of the organizations owners of the compromised domains are also equally distributed throughout the country.

6.1.3 Remarks

We have described recent attack trends in web applications and illustrated their potential dangers for e-government initiatives. Attackers may hide fraudulent content in widely trusted web sites and let those contents appear only to selected users, perhaps depending on the navigation path they followed. The potential effects of attacks of this form are very dangerous: such hidden content is very difficult to detect by administrators and HTTPS—the ubiquitous main line of defense—does not address this threat, thus it does not defend citizens in any way. In

our opinion, it is only a matter of time before targeted attacks based on the technical means described here will start appearing. Indeed, criminal attacks on government sites aimed at selling fake certifications are already occurring [80].

We have defined a methodology for detecting fraudulent contents and demonstrated that Italians public administrations indeed tend to host fake content. Our study certainly requires further work, in breadth and depth of the analysis, yet we believe it may help the research community in promoting greater awareness of the problem and in developing solutions both effective and practical.

6.2 Hidden fraudulent URL detection

Finally we attempted to define a methodology to detect certain types of fraudulent intrusions that are becoming of practical interest on a large scale. A peculiar form of Internet security threats consists in the fraudulent modification of a web site, both in the form of the alteration of existing content or in the form of addition of new content. Fraudulent modifications might aim at advertising disturbing content which is entirely different from the original one (*defacement*); or, they might aim at facilitate *phishing* campaign; or, they might aim at spreading malware by exploiting vulnerabilities of web browsers. Being *hidden*, that is, reachable at an URL where no page should exist, additive fraudulent content could remain undetected by the web site administrators for a long time.

The strategies for persuading unsuspecting users to visit hidden fraudulent pages can differ, but all the methods require that the user takes some action, usually by clicking on a link which points to the fraudulent URL. Ideally, when a user clicks on an unknown URL, he should assess the risk associated with her action. This risk assessment is indeed a difficult task for common users and is furtherly exhacerbated by the fact that a URL could refer a fraudulent content which is hosted within a site *trusted* by the user, i.e., a site routinely accessed by the user and whose administrators perform their best effort to host content that is indeed genuine and not harmful.

Fraudulent modification to trusted web sites could be broadly divided in two categories: (i) subtle changes to existing pages or (ii) addition of new pages at URLs at which no page is supposed to exist. Both these attacks are hard to detect because site administrators will likely never see any anomaly in the served content. The diffusion of this phenomenon is confirmed by abundant anectodal evicence and quantified in different studies. In [95] researchers found that approximately 1.5% of sites be-

longing to Italian public administrations serve contents that admittedly is not supposed to be there.

It is important to point out that HTTPS—the main and ubiquitous line of defense in sensitive web sites—does *not* provide any defense in this respect. HTTPS ensures secrecy, integrity and authentication by means of cryptographic techniques. The problem is, the server site is authenticated as a whole: thus, any page coming from that site appears as being legitimate, from the HTTPS point of view.

In this paper, we present an approach for the detection of hidden fraudulent URLs before actually fetching the corresponding page. A system with this ability could be deployed in a variety of ways, for instance, within an e-mail client or within a web browser and trigger an alert to the user before actually accessing the fraudulent page itself. It could also be deployed within a web proxy, at the outside border of an organization, as a layer for a defense in depth strategy.

The peculiarity of our proposal consists in not using any feature related to the domain part of the URL, i.e., the URL portion which identifies the host. The rationale for this requirement is our focus on addressing fraudulent URLs inserted into trusted web sites. In this scenario, the domain part of the URL is obviously not a discriminant between fraudulent and legitimate URLs belonging to the same web site. For the same reasons, we purposefully avoided using other domain-related features, e.g., `whois` queries or geographic properties.

We use lexical features extracted from the URL to be classified, excluding the domain part of the URL. These features are then input to a Support Vector Machine. We also propose two variants of this method, which augment the features available for classification based on the responses to HTTP requests directed at the site hosting the URL to be classified, but that do not involve fetching the actual content of that URL.

Our approach effectiveness was assessed on two categories of hidden fraudulent URLs: hidden phishing pages and hidden defacements. The two datasets are composed of about 6500 and 2150 URLs respectively. Our approach achieves an accuracy of about 96% for the phishing category and 99% for the defacement one.

6.2.1 Related work

The problem of detecting fraudulent URLs is long-established and has been studied from several points of view. As far as we know, though, this is the first approach focussed on detecting fraudulent URLs that are hosted on a trusted (in the sense clarified above) web site. With respect to other existing approaches for detecting fraudulent URLs, we (i) ex-

cluded the domain part from the feature used for classifying a URL and (ii) we assessed explicitly our approach ability to discriminate between fraudulent and legitimate URLs belonging to the *same* web site.

Almost all the previous works in this area focused on the detection of phishing URLs and URLs related to spam-advertised web sites. In this paper we also consider hidden *defacements* URLs—a quick look at the on-line defacement archive http://www.zone-h.org shows that exploit of this form occur routinely at organizations of any size. A large scale study showed that the typical reaction time for recovering the defaced page is surprisingly long, in the order of several days [12]. A system for detecting automatically whether a given URL is defaced has been proposed in [13]. The system operates as a cloud-based service and is designed for dynamic web content: it first builds a profile of the web page and then sends an alert whenever the page content deviates from that profile. This system is unable to detect hidden defacements because it must know the URLs to be monitored in advance.

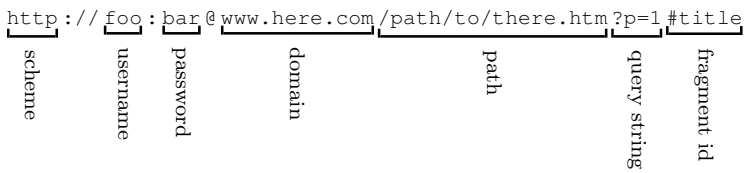
Techniques for detecting phishing attacks can be subdivided in three categories: URL-based, host-based and content-based detection methods. Broadly speaking, URL-based approaches are faster and more scalable than the others since that they can classify a URL based on the URL itself, without collecting any further information.

An approach that rely only on URL-based features is proposed in [54]. The authors of the cited work use structural features of the URLs and some binary features indicating the presence of certain words in the URLs itself. This approach requires a fair amount of domain knowledge (for choosing the words corresponding to features) and, as such, it appears to be difficult to generalize.

Other studies augment URL-based features by features extracted from the host where the corresponding document resides [63, 90, 70, 71]. The approach proposed in [63] uses lexical features, geographic properties and features extracted from `whois` queries: primary domain name, the registrar, the registrant, and the registration date. Authors of [90, 70, 71] use a larger set of host-based features such as `whois` queries, DNS information, geographic properties, connection speed and membership in blacklists; these features are used with online classification algorithms. In [70] the authors experiment with different classifiers (Support Vector Machine (SVM), logistic regression, Bayes), whereas in [90] the features are processed also using the confidence-weighted algorithm, passive-aggressive algorithm and the perceptron.

Approaches that belong to the content-based category [27, 105, 22] are more time consuming than the others, since they involve fetching and analyzing the full page content. The authors of [27] use a number of features coming from the HTML and JavaScript code found in the page

Figure 6.1: The structure of a URL



in addition to URL and host based features: the obtained features set is analyzed with different classifiers. In [22] the features corresponding to a URL are extracted using a bag-of-word approach while the page content is compared to sets of files from previously confirmed phishing websites using MD5 hashes. The classification is performed using a confidence weighted algorithm and tests are conducted on daily batches of URLs simulating a model updated by a daily URL blacklist/whitelist feed. The approach proposed in [105] uses features extracted from the web page content (the presence of password fields and the external links frequency) in addition to URL and host based features with a proprietary machine learning algorithm: due to their nature, these content-based features fit the phishing URLs detection scenario, while they could not be appropriate for the defacement URLs detection scenario—a phishing page is carefully crafted to resemble genuine content, while a defacement page is usually very different from the original page.

6.2.2 Our approach

A URL (Uniform Resource Locator) is a string which identifies a web resource (Figure 6.1). We say that a URL is *hidden* if the corresponding page is hosted within a site without the administrator being aware of it. We say that a URL is *fraudulent* if the corresponding page is a defacement or a phishing attack (pages devoted to disseminating malware are beyond the scope of this work). The goal of the proposed method is to associate an input URL u with a boolean value which indicates if u is an *hidden fraudulent URL*.

We propose three increasingly more complex variants of the method. Each variant makes use of a superset of the information available to the previous variant. The first one, which we call *lexical*, uses only features extracted from the URL u itself. The second one, *lexical+headers*, augments those features with some of the headers obtained as response to an HTTP request for u. Finally, *lexical+headers+age*, uses also some of the headers obtained while fetching the home page of the domain of u—i.e., the web page identified by u without the path and subsequent components. In other words, lexical may be applied for classifying u without

issuing any request for u and may thus be applied *offline*; lexical+headers requires one HTTP HEAD request for u; lexical+headers+age requires one HTTP HEAD request for u and another HEAD request for the home page of the domain of u.

Each variant requires a preliminary parameter calibration to be performed only once based on labelled data collected in a *training set*. The training set is transformed into a matrix F of *features*, with one row for each URL in the training set and one column for each feature analyzed—each variant analyzing a superset of the features analyzed by the previous one. We describe the three variants in the next sections.

Lexical

The lexical variant uses only the URL string itself, as follows (Figure 6.1 shows the structure of an example URL..). Let $U = \{u_1, \ldots, u_n\}$ be the training set and $L = \{l_1, \ldots, l_n\}$ the corresponding set of labels: $l_i = \text{true}$ if and only if u_i is an hidden fraudulent URL.

For the tuning, we first remove from each URL u_i every character up to the domain (included), and obtain a string p_i. Then, we extract the unigrams (i.e., character occurrences) from each p_i and obtain a matrix F of *features*, where $f_{i,j}$ is the number of occurrences of character c_j in p_i. Finally, we train a Support Vector Machine (SVM) on F using the labels of L. We use a third-degree polinomial kernel with cost parameter $C = 10$.

The classification of a unknown URL u is performed with the same steps as above, i.e.: (i) preprocess u to obtain a string p; (ii) compute the occurrences in p of the characters corresponding to the columns of F, obtaining a feature vector; (iii) apply the SVM to the feature vector.

Lexical+headers

The lexical+headers variant uses, in addition to the features of the previous variant, features extracted from the values of some of the HTTP response headers obtained when requesting the url U to be classified:

1. `Server`: name and version of the software running the server;
2. `Content-Type`: MIME type of the content of the named web-resource;
3. `Content-Lenght`: length of the response body in octets;
4. `X-Powered-By`: framework of the web application that produces the content of the web-resource (e.g., ASP.NET, PHP, JBoss);

In order to minimize the traffic, we issue HTTP HEAD requests instead of HTTP GET requests (while a GET asks for a resource, an HEAD asks for a response identical to the one that would correspond to a GET, but without the response body, i.e., without the actual resource).

The tuning of this variant proceeds as follows. For each $u_i \in U$, we perform a HEAD request to u_i and store each of the received header values $h_i^1, \ldots, h_i^4$ (in case the response does not contain the k-th header, we set the corresponding h_i^k is to the empty string). We pre-process the values for the `Server` and `X-Powered-By` headers so as to keep only the framework name and the major and minor version number (e.g., `Apache/2.2.22-12` becomes `Apache/2.2`). We then transform the header values in numerical features as follows. For each k-th header, we build a matrix F^k based on all the $v_1^k, \ldots, v_{n_k}^k$ observed values, as follows. F^k has a row for each URL and a column for each distinct header value: a matrix element $f_{i,j}^k$ of F^k is 1 if and only if $h_i^k = v_j^k$, 0 otherwise. In other words, F^k is a matrix of the n_k binary features corresponding to the observed values for the k-th header. Finally, the new features in F^1, F^2, F^3, F^4 are added to the original feature matrix F by joining all the columns of the five matrices.

The remaining part of the tuning step and the classification step are the same of the lexical variant.

Lexical+headers+age

The lexical+headers+age variant augments the previous features with the difference between the timestamps given by the values of the `Last-Modified` header obtained for the URL u and for the home page corresponding to u. These timestamps correspond to the creation in the tuning of two further columns in F, as follows.

For each $u_i \in U$, we extract the `Last-Modified` value (from the same response to the HEAD request performed for the previous variant) and store it in t_i as a date (in case the response does not contain the `Last-Modified`, we set $t_i = \emptyset$). Next, we remove from u_i the substring which starts from the path (included) and obtain d_i, which is the URL of the home page corresponding to u_i. Then, we perform a HEAD request to d_i, store the value of the `Last-Modified` header in t'_i and set $a_i = t_i - t'_i$ (in seconds). We also set a binary feature a'_i to 0, if both t_i and t'_i were defined (i.e, $t_i \neq \emptyset$ and $t'_i \neq \emptyset$), or 1, if at least one of them was undefined. Finally, we add two new columns to F, given by the a_i and a'_i.

The rationale is that we try to exploit the information given by the *relative age* of the u resource, i.e, its last modification date compared to the home page last modification date. In other words, if the home page of the web site was modified long before the URL u under analysis, this

could be a symptom of an hidden fraudulent URL.

6.2.3 Dataset

We assessed the effectiveness of our approach on two categories of hidden fraudulent URLs: (i) hidden phishing pages and (ii) hidden defacements. Due to the lack of publicly available datasets we collected, for each category, a set of real-world URLs as described below.

Concerning hidden phishing pages, we used the data provided by Phishtank[5]. Phishtank is a public web-based archive of phishing attacks: a user can report Phishtank of a possible attack by providing the phishing page URL. A team of voluntary experts may then verify the user's notification, marking the URLs as "valid phish". Moreover, Phishtank periodically verify that each phishing attack is actually still in place—i.e., if a page is served at the corresponding URL—and mark those URLs as "online". We composed a set U^P of about 7500 valid and online URLs extracted from Phishtank; we verified that each URL in U^P was still marked as valid and online for all the duration of our experimental evaluation.

Concerning hidden defacements, we used data provided by Zone-H[6]. The Zone-H Digital Attacks Archive is a public web-based archive of defacement attacks: users or attackers themselves report a URL of a defaced page to Zone-H; later, a human operator verifies and confirms the attack which is then published on the Zone-H web site. We composed a list U^D of about 2500 URLs extracted from Zone-H.

A key ingredient of our problem is the ability of identifying hidden fraudulent URLs, even when they are hosted within trusted web sites. For this reason, we defined 5 sets of URLs: (i) hidden fraudulent URLs: U^P_+ and U^D_+ (phishing and defacement category respectively); (ii) URLs of legitimate pages belonging to sites that are trusted but also host fraudulent URLs: U^P_- and U^D_- (phishing and defacement category respectively), (iii) URLs of legitimate pages belonging to trusted and (as far as we can tell) uncompromised web sites: U_-. In order to populate these sets we proceeded as follows.

We assumed that both Phishtank and Zone-H are indeed authoritative with respect to URLs being fraudulent, i.e., we assumed that all URLs of U^P and U^D are fraudulent. First, we dropped from these lists: (a) URLs whose domain is an IP address; (b) URLs whose path is empty or equal to `index.html`. The rationale for dropping these items is that they are not intended to be hidden. In the former case, we assume that the whole web site is involved in the attack (since it has no DNS name):

[5] http://www.phishtank.com
[6] http://www.zone-h.org

edition.cnn.com	www.steampunk.dk
www.bbc.com	www.weather.com
www.nytimes.com	www.godaddy.com
www.microsoft.com	www.nbcnews.com
www.whitehouse.gov	www.foxnews.com
www.adobe.com	www.bankofamerica.com
www.huffingtonpost.com	www.spiegel.de
espn.go.com	www.aweber.com
www.mediafire.com	www.chase.com
www.1advice.com	amazonaws.com

Table 6.6: Lists of 20 domains used to select the maximum crawling depth.

in other words, it is not a legitimate web site to which an illegitimate content has been added. In the latter case, the attack is manifestly not hidden, since it resides at the root of the domain.

Second, we dropped from the list all the fraudulent URLs that are not hidden. We should consider that an URL is hidden if it is never reachable while crawling the site. Clearly, crawling an entire site is often not feasible. We hence marked an URL as hidden if it is not found by crawling the corresponding web site within up to the third level of depth. In order to justify this choice, in particular the choice of the third level of depth, we performed the following quantitative analysis. We selected a set W of 20 web sites extracted from the top 500 web sites ranking provided by Alexa[7]. We excluded from this selection: (i) web sites providing different content depending on whether the user is authenticated, (ii) social network web sites, (iii) search engines. Table 6.6 shows the 20 selected web sites. For each web site $w_i \in W$, we crawled the site up to the 10th level of depth and saved all the URLs obtained in the list U_{-}. We also determined, for each level l, the number of URLs $n_{i,l}$ found by crawling up to that level. Then, we computed the crawling coverage $C_{i,l} = \frac{n_{i,l}}{n_{i,10}}$ as the fraction of URLs found by crawling up to level 10 which were also found by crawling up to level l. Figure 6.2 shows crawling coverage, averaged across all 30 web sites, vs. the level l: it can be seen that the curve has a clear slope change at $l = 3$ and tends to be flat for highest values of l. In other words, crawling up to the third level is an acceptable compromise between coverage (which is, on the average, 88% for $l = 3$) and easiness of obtaining the data.

We now describe how we populated the sets necessary for our evaluation. Concerning the sets U_{-}^{P} and U_{-}^{D} of legitimate pages belonging to sites that are trusted but also host fraudulent URLs, we procedeed

[7] http://www.alexa.com/topsites

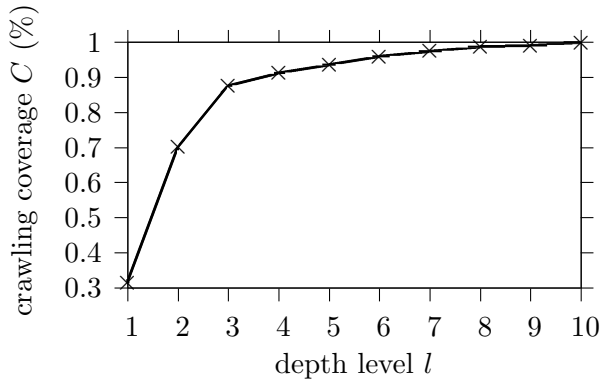


Figure 6.2: Crawling coverage C vs. depth level l.

as follows. For each $u_i \in U^P$, we crawled the corresponding domain d_i up to the third level of depth and added all the obtained URLs to the (initially empty) set U^P_-. We repeated the same procedure on U^D and obtained U^D_-.

Concerning the sets U^P_+ and U^D_+ of hidden fraudulent pages, we procedeed as follows. For each $u_i \in U^P$, we added u_i to the (initially empty) set U^P_+ if, and only if, it was not found in the crawl described above. We repeated the same procedure on U^D and obtained U^D_+.

Finally, the set U_- of legitimate pages belonging to trusted and (as far as we can tell) uncompromised sites, consisted of all the URLs found while crawling the 20 web sites in Table 6.6 up to the 10th depth level.

We collected this data during months July to November, 2012 and obtained 6564, 78388, 2144, 94370, 3713 URLS respectively in U^P_+, U^P_-, U^D_+, U^D_- and U_-. The dataset is publicly available on our lab web site[8].

6.2.4 Experiments

We performed two suites of experiments in order to evaluate our approach effectiveness separately on the two hidden URLs categories, i.e., phishing and defacement.

For the purpose of the experimental evaluation, we built, for each category, a labeled and balanced dataset, as follows. For the phishing category, we set U^P_e to the set containing all U^P_+ URLs, 3282 URLs randomly extracted from U^P_- and 3282 URLs randomly extracted from U_-; we also set the corresponding labels L^P accordingly. For the defacement category, we set U^D_e to the set containing all U^D_+ URLs, 1072 URLs randomly extracted from U^D_- and 1072 URLs randomly extracted from U_-; we also set the corresponding labels L^D accordingly.

[8] http://machinelearning.inginf.units.it/data-and-tools/hidden-fraudulent-urls-dataset

We assessed the our approach effectiveness in terms of the following performance indexes (for the sake of clarity, we here show index names only for the phishing category):

- accuracy, i.e., the ratio between correctly classified URLs and all the processed URLs;
- false negative rate (FNR on U_+^P);
- false positive rate, calculated only on URLs of U_-^P (FPR on U_-^P);
- false positive rate, calculated only on URLs of U_- (FPR on U_-).

For each variant of our method, we repeated 5 times the following procedure, collecting for each execution the four performance indexes explained before (we here consider the phishing category): (i) we randomly split U_e^P in a training set and testing set, 90% of URLs used for training and the remaining 10% for testing (we preserved the proportion of URLs coming from U_+^P, U_-^P and U_-), (ii) we trained the SVM classifier on the training set and, finally, (iii) we applied the trained classifier on each URL of the testing set.

Method	Accuracy (%)		FNR on U_+^P (%)	
	Avg.	Dev. Std.	Avg.	Dev. Std.
Lexical	92.50	0.62	9.80	0.85
Lexical+headers	95.34	0.48	4.93	1.05
Lexical+headers+age	95.57	0.37	4.87	1.06

Method	FPR on U_-^P (%)		FPR on U_- (%)	
	Avg.	Dev. Std.	Avg.	Dev. Std.
Lexical	5.20	0.84	4.93	1.21
Lexical+headers	4.38	0.62	0.79	0.70
Lexical+headers+age	3.98	0.52	0.73	0.73

Table 6.7: Results for the phishing category.

Tables 6.7 and 6.8 report the performance indexes, averaged across the five repetitions, for the three variants of our method applied on the phishing and defacement categories, respectively.

The accuracy of our system, using the lexical+headers+age variant, is greater than 99% and 95% for defacement and phishing categories, respectively.

As expected, the off-line variant (lexical) has a lower accuracy (98% for defacement and 92% for phishing): on the other hand, this variant requires just less than a millisecond to classify an URL, whereas the other

Method	Accuracy (%)		FNR on U_+^D (%)	
	Avg.	Dev. Std.	Avg.	Dev. Std.
Lexical	98.37	0.68	0.93	0.74
Lexical+headers	99.35	0.3	0.37	
Lexical+headers+age	99.26	0.3	0.47	0.66

Method	FPR on U_-^D (%)		FPR on U_- (%)	
	Avg.	Dev. Std.	Avg.	Dev. Std.
Lexical	2.32	0.87	2.41	1.92
Lexical+headers	0.93	0.57	0.93	0.65
Lexical+headers+age	1.02	0.61	0.93	0.65

Table 6.8: Results for the defacements category.

on-line variants require to perform one or two HTTP requests, which can result in several seconds of elapsed time.

Another key finding is that FPR on U_- is lower than 1% for both categories, when using the on-line variants of our method. Only for the phishing category FPR on U_-^P is slightly higher (about 4%): this result is somewhat justified because U_-^P are negative URLs belonging to compromised web sites and could hence resemble fraudulent URLs.

Concerning FNR, the experimental evaluation shows that it is higher for the phishing category (about 5%) than for the defacement category (less than 1%). The reason is because an attacker who puts in place a phishing attack will purposely shape all components of the attack (i.e., including the URL of the fraudulent page) so as to make it as much unnoticed as possible. An attacker which hides a defacement page will likely not care too much of whether the attack URL is easily detectable.

The time needed for the URL classification is smaller than 1 msec, 800 msec and 1600 msec respectively for lexical, lexical+headers and lexical+headers+age, on average. Note, however that the lexical variant can work off-line (i.e., without any access to the Internet). The actual computation time is < 1 msec for classifying an URL; 2 sec and 60 sec are required for the initial tuning, respectively on a training set of about 4300 defacement and about 13000 phishing URLs. We executed all our experiments on a machine powered with a quad-core Intel Xeon X3323 (2.53 GHz) and 2GB of RAM, with an high-speed academic connection to the Internet.

Bibliography

[1] Digital Agenda: Commission outlines action plan to boost Europe's prosperity and well-being. May 2010. [cited at p. 97]

[2] Verizon RISK Team. 2012 Data Breach Investigation Report. Technical report, 2012. [cited at p. 97]

[3] I. Ahmadullin, J. Allebach, N. Damera-Venkata, J. Fan, S. Lee, Q. Lin, J. Liu, and E. O'Brien-Strain. Document visual similarity measure for document search. In *Proceedings of the 11th ACM symposium on Document engineering*, DocEng '11, pages 139–142, New York, NY, USA, 2011. ACM. [cited at p. 23]

[4] M. Aiello, C. Monz, L. Todoran, and M. Worring. Document understanding for a broad class of documents. *International Journal on Document Analysis and Recognition*, 5(1):1–16, Nov. 2002. [cited at p. 22]

[5] C. Alippi, F. Pessina, and M. Roveri. An adaptive system for automatic invoice-documents classification. In *IEEE International Conference on Image Processing, 2005. ICIP 2005*, volume 2, 2005. [cited at p. 23]

[6] A. Amano, N. Asada, M. Mukunoki, and M. Aoyama. Table form document analysis based on the document structure grammar. *International Journal on Document Analysis and Recognition*, 8(2):201–213, June 2006. [cited at p. 22, 49]

[7] R. Babbar. Clustering Based Approach to Learning Regular Expressions over Large Alphabet for Noisy Unstructured. *the 19th ACM conference on Conference on information and knowledge management CIKM 10*, pages 43–50, 2010. [cited at p. 64, 65]

[8] D. Barbosa, L. Mignet, and P. Veltri. Studying the xml web: Gathering statistics from an xml sample. *World Wide Web*, 9:187–212, 2006. 10.1007/s11280-006-8437-6. [cited at p. 83]

[9] L. Barbosa, J. Freire, and A. Silva. Organizing Hidden-Web Databases by Clustering Visible Web Documents. *2007 IEEE 23rd International Conference on Data Engineering*, pages 326–335, 2007. [cited at p. 21]

[10] D. Barrero, D. Camacho, and M. R-Moreno. Automatic Web Data Extraction Based on Genetic Algorithms and Regular Expressions. *Data Mining and Multi-agent Integration*, pages 143–154, 2009. [cited at p. 63, 68]

[11] A. Bartoli, G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference companion*, pages 1477–1478. ACM, 2012. [cited at p. 13, 15, 85, 88]

[12] A. Bartoli, G. Davanzo, and E. Medvet. The Reaction Time to Web Site Defacements. *Internet Computing, IEEE*, 13(4):52–58, July 2009. [cited at p. 97, 111]

[13] A. Bartoli, G. Davanzo, and E. Medvet. A Framework for Large-Scale Detection of Web Site Defacements. *ACM Transactions on Internet Technology*, 10(3), Oct. 2010. [cited at p. 97, 111]

[14] A. Bartoli, G. Davanzo, E. Medvet, and E. Sorio. Improving features extraction for supervised invoice classification. In *Proceedings of the 10th IASTED International Conference*, volume 674, page 401, 2010. [cited at p. 12, 15]

[15] A. Bartoli, G. Davanzo, E. Medvet, and E. Sorio. Semisupervised wrapper choice and generation for print-oriented documents. *IEEE Transactions on Knowledge and Data Engineering*, page 1, 2012. [cited at p. 12, 15]

[16] M. Becchi, C. Wiseman, and P. Crowley. Evaluating regular expression matching engines on network and general purpose processors. In *Proceedings of the 5th ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 30–39. ACM, 2009. [cited at p. 61]

[17] S. M. Beitzel, E. C. Jensen, and D. A. Grossman. Retrieving ocr text: A survey of current approaches. In *Symposium on Document Image Understanding Technologies (SDUIT*, 2003. [cited at p. 49]

[18] Y. Belaid and A. Belaid. Morphological tagging approach in document analysis of invoices. In *Proceedings of the Pattern Recognition, 17th International Conference on (ICPR'04) Volume 1 - Volume 01*, pages 469–472. IEEE Computer Society, 2004. [cited at p. 22, 49]

[19] G. J. Bex, W. Gelade, F. Neven, and S. Vansummeren. Learning Deterministic Regular Expressions for the Inference of Schemas from XML Data. *ACM Transactions on the Web*, 4(4):1–32, Sept. 2010. [cited at p. 61, 65, 84, 90]

[20] G. J. Bex, F. Neven, T. Schwentick, and S. Vansummeren. Inference of concise regular expressions and DTDs. *ACM Transactions on Database Systems*, 35(2):1–47, Apr. 2010. [cited at p. 61, 65]

[21] G. J. Bex, F. Neven, and S. Vansummeren. *Inferring XML schema definitions from XML data*, volume 29, pages 998–1009. VLDB Endowment, 2007. [cited at p. 85]

[22] A. Blum, B. Wardman, T. Solorio, and G. Warner. Lexical feature based phishing URL detection using online learning. *Proceedings of the 3rd ACM workshop on Artificial intelligence and security - AISec '10*, page 54, 2010. [cited at p. 111, 112]

[23] F. Brauer, R. Rieger, A. Mocan, and W. Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1285–1294. ACM, 2011. [cited at p. 61, 64, 65, 69, 70, 72, 74]

[24] A. Bràzma. Efficient identification of regular expressions from representative examples. In *Proceedings of the sixth annual conference on Computational learning theory*, volume 1, pages 236–242. ACM, 1993. [cited at p. 63, 85]

[25] M. Bronzi, V. Crescenzi, P. Merialdo, and P. Papotti. Wrapper generation for overlapping web sources. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2011 IEEE/WIC/ACM International Conference on*, volume 1, pages 32 –35, aug. 2011. [cited at p. 18, 21]

[26] M. J. Cafarella, A. Halevy, and N. Khoussainova. Data integration for the relational web. *Proc. VLDB Endow.*, 2(1):1090–1101, Aug. 2009. [cited at p. 18, 21]

[27] D. Canali, M. Cova, G. Vigna, and C. Kruegel. Prophiler: A Fast Filter for the Large-Scale Detection of Malicious Web Pages. In *World Wide Web Conference*. [cited at p. 111]

[28] F. Cesarini, E. Francesconi, M. Gori, and G. Soda. Analysis and understanding of multi-class invoices. *International Journal on Document Analysis and Recognition*, 6(2):102–114, Oct. 2003. [cited at p. 22, 24, 46, 49]

[29] A. Cetinkaya. Regular expression generation through grammatical evolution. In *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*, GECCO '07, pages 2643–2646, New York, NY, USA, 2007. ACM. [cited at p. 63, 68, 69, 70, 78, 79]

[30] C. H. Chang, M. Kayed, R. Girgis, and K. F. Shaalan. A Survey of Web Information Extraction Systems. *IEEE Transactions on Knowledge and Data Engineering*, 18(10):1411–1428, 2006. [cited at p. 18, 22]

[31] C.-C. Chen, K.-H. Yang, C.-L. Chen, and J.-M. Ho. BibPro: A Citation Parser Based on Sequence Alignment. *IEEE Transactions on Knowledge and Data Engineering*, 24(2):236–250, Feb. 2012. [cited at p. 61]

[32] K. Chen, G. Gu, J. Zhuge, J. Nazario, and X. Han. Webpatrol: automated collection and replay of web-based malware scenarios. In *Proceedings of the 6th ACM Symposium on Information, Computer and Communications Security*, pages 186–195. ACM, 2011. [cited at p. 61]

[33] N. Chen and D. Blostein. A survey of document image classification: problem statement, classifier architecture and performance evaluation. *International Journal on Document Analysis and Recognition*, 10(1):1–16, June 2007. [cited at p. 23]

[34] B. Chidlovskii. Schema extraction from xml data: A grammatical inference approach. In *KRDB'01 Workshop (Knowledge Representation and Databases*, 2001. [cited at p. 85]

[35] S. L. Chuang, K. C. C. Chang, and C. X. Zhai. Context-aware wrapping: synchronized data extraction. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 699–710. VLDB Endowment, 2007. [cited at p. 18, 21]

[36] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *Evolutionary Computation, IEEE Transactions on*, 6(2):182 –197, apr 2002. [cited at p. 68]

[37] B. Dunay, F. Petry, and B. Buckles. Regular language induction with genetic programming. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on*, volume 1, pages 396–400. IEEE, 1994. [cited at p. 63]

[38] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *Proceedings of the VLDB Endowment*, 2(1):209–226, 2009. [cited at p. 22]

[39] H. Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521–541, Apr. 2009. [cited at p. 64, 85]

[40] E. Ferrara, G. Fiumara, and R. Baumgartner. Web Data Extraction , Applications and Techniques : A Survey. *ACM Computing Surveys*, V(June):1–20, 2010. [cited at p. 18, 22]

[41] S. Flesca, E. Masciari, and A. Tagarelli. A fuzzy logic approach to wrapping pdf documents. *Knowledge and Data Engineering, IEEE Transactions on*, 23(12):1826–1841, Dec. 2011. [cited at p. 24]

[42] D. Florescu. Managing semi-structured data. *Queue*, 3(8):18–24, Oct. 2005. [cited at p. 84]

[43] J. Friedl. *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006. [cited at p. 65]

[44] A. Fujiyoshi, M. Suzuki, and S. Uchida. Syntactic detection and correction of misrecognitions in mathematical ocr. In *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, ICDAR '09, pages 1360–1364, Washington, DC, USA, 2009. IEEE Computer Society. [cited at p. 50]

[45] M. Garofalakis, A. Gionis, R. Rastogi, S. Seshadri, and K. Shim. XTRACT: A System for Extracting Document Type Descriptors from XML Documents. pages 165–176, 2000. [cited at p. 84]

[46] A. González-Pardo, D. Barrero, D. Camacho, and M. R-Moreno. A case study on grammatical-based representation for regular expression evolution. In Y. Demazeau, F. Dignum, J. Corchado, J. Bajo, R. Corchuelo, E. Corchado, F. Fernández-Riverola, V. Julián,

P. Pawlewski, and A. Campbell, editors, *Trends in Practical Applications of Agents and Multiagent Systems*, volume 71 of *Advances in Intelligent and Soft Computing*, pages 379–386. Springer Berlin / Heidelberg, 2010. [cited at p. 63, 68]

[47] Z. Gyongyi and H. Garcia-Molina. Web spam taxonomy. In *First International Workshop on Adversarial Information Retrieval on the Web (AIRWeb 2005)*, April 2005. [cited at p. 98]

[48] H. Hamza, Y. Belaid, A. Belaid, and B. Chaudhuri. Incremental classification of invoice documents. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4, 8-11 2008. [cited at p. 23]

[49] H. Hamza, Y. Belaïd, and A. Belaïd. Case-Based reasoning for invoice analysis and recognition. In *Case-Based Reasoning Research and Development*, pages 404–418. 2007. [cited at p. 46, 47]

[50] T. Hassan. User-guided wrapping of pdf documents using graph matching techniques. In *Document Analysis and Recognition, 2009. 10th International Conference on*, pages 631–635, 2009. [cited at p. 24]

[51] H. He, W. Meng, C. Yu, and Z. Wu. Automatic integration of Web search interfaces with WISE-Integrator. *The VLDB Journal*, 13(3):1–29, 2004. [cited at p. 18, 21]

[52] J. Hegewald, F. Naumann, and M. Weis. Xstruct: Efficient schema extraction from multiple and large xml documents. *22nd International Conference on Data Engineering Workshops ICDEW06*, pages 81–81, 2006. [cited at p. 85]

[53] T. Hruby, K. van Reeuwijk, and H. Bos. Ruler: high-speed packet matching and rewriting on npus. In *Proceedings of the 3rd ACM/IEEE Symposium on Architecture for networking and communications systems*, pages 1–10. ACM, 2007. [cited at p. 61]

[54] H. Huang, L. Qian, and Y. Wang. A SVM-based Technique to Detect Phishing URLs. *Information Technology Journal*, 2012. [cited at p. 111]

[55] J. J. Hull and S. N. Srihari. Experiments in text recognition with binary n-gram and viterbi algorithms. *IEEE Trans. Pattern Anal. Mach. Intell.*, 4:520–530, May 1982. [cited at p. 49]

[56] L. Invernizzi and P. M. Comparetti. Evilseed: A guided approach to finding malicious web pages. *Security and Privacy, IEEE Symposium on*, 0:428–442, 2012. [cited at p. 99, 103]

[57] R. Khare, Y. An, and I.-Y. Song. Understanding Deep Web Search Interfaces : A Survey. *ACM SIGMOD Record*, 39(1):33–40, 2010. [cited at p. 19, 20]

[58] E. Kinber. Learning regular expressions from representative examples and membership queries. *Grammatical Inference: Theoretical Results and Applications*, pages 94–108, 2010. [cited at p. 64]

[59] B. Klein, S. Agne, and A. Dengel. Results of a study on InvoiceReading systems in Germany. In *Document Analysis Systems VI*, pages 451–462. 2004. [cited at p. 45]

[60] N. Kroes. A European Strategy for Internet Security. Mar. 2012. [cited at p. 97]

[61] K. Kukich. Techniques for automatically correcting words in text. *ACM Comput. Surv.*, 24:377–439, December 1992. [cited at p. 49]

[62] W. B. Langdon, J. Rowsell, and A. P. Harrison. Creating regular expressions as mrna motifs with gp to predict human exon splitting. In *Proceedings of the 11th Annual conference on Genetic and evolutionary computation*, GECCO '09, pages 1789–1790, New York, NY, USA, 2009. ACM. [cited at p. 65, 68]

[63] A. Le and A. Markopoulou. PhishDef: URL Names Say It All. In *Proceedings IEEE INFOCOM*, 2010. [cited at p. 111]

[64] E. Lee and T.-h. Kim. Automatic generation of XForms code using DTD. *Fourth Annual ACIS International Conference on Computer and Information Science (ICIS'05)*, pages 210–214, 2005. [cited at p. 85]

[65] N. Leontiadis and T. Moore. Measuring and analyzing search-redirection attacks in the illicit online prescription drug trade. *Proc. USENIX Security*, 2011. [cited at p. 98]

[66] D. Lewis, G. Agam, S. Argamon, O. Frieder, D. Grossman, and J. Heard. Building a test collection for complex document information processing. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 665–666, Seattle, Washington, USA, 2006. ACM. [cited at p. 43]

[67] Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and A. Arbor. Regular Expression Learning for Information Extraction. *Computational Linguistics*, (October):21–30, 2008. [cited at p. 64, 65, 69, 70, 72, 74]

[68] Y. Li, R. Krishnamurthy, S. Vaithyanathan, and H. V. Jagadish. H.v.jagadish. getting work done on the web: Supporting transactional queries. In *In SIGIR*, 2006. [cited at p. 70]

[69] W. Liu, X. Meng, and W. Meng. Vide: A vision-based approach for deep web data extraction. *IEEE Transactions on Knowledge and Data Engineering*, 22(3):447 –460, march 2010. [cited at p. 18, 20, 21]

[70] J. Ma, L. Saul, S. Savage, and G. Voelker. Beyond blacklists: learning to detect malicious web sites from suspicious URLs. *Proceedings of the 15th ACM . . .*, 2009. [cited at p. 111]

[71] J. Ma, L. K. Saul, S. Savage, and G. M. Voelker. Learning to detect malicious URLs. *ACM Transactions on Intelligent Systems and Technology*, 2(3), 2011. [cited at p. 111]

[72] E. Medvet and A. Bartoli. Brand-related events detection, classification and summarization on twitter. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology - Volume 01*, WI-IAT '12. IEEE Computer Society, 2012, to appear. [cited at p. 70]

[73] E. Medvet, A. Bartoli, and G. Davanzo. A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition IJDAR*, 2010. [cited at p. 18, 25, 32, 40, 49]

[74] E. Medvet, A. Bartoli, and G. Davanzo. A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition*, 14:335–347, 2011. 10.1007/s10032-010-0137-1. [cited at p. 70]

[75] L. Mignet, D. Barbosa, and P. Veltri. The xml web: a first study. In *Proceedings of the 12th international conference on World Wide Web*, WWW '03, pages 500–510, New York, NY, USA, 2003. ACM. [cited at p. 83]

[76] D. Miller, S. Boisen, R. Schwartz, R. Stone, and R. Weischedel. Named entity extraction from noisy input: Speech and ocr. In *In Proceedings of the 6th Applied Natural Language Processing Conference*, pages 316–324, 2000. [cited at p. 49]

[77] J.-K. Min, J.-Y. Ahn, and C.-W. Chung. Efficient extraction of schemas for XML documents. *Information Processing Letters*, 85(1):7–12, 2003. [cited at p. 85]

[78] E. Minkov, R. C. Wang, and W. W. Cohen. Extracting personal names from email: applying named entity recognition to informal text. In *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, HLT '05, pages 443–450, Stroudsburg, PA, USA, 2005. Association for Computational Linguistics. [cited at p. 70]

[79] C.-h. Moh, E.-p. Lim, and W.-k. Ng. DTD-Miner: a tool for mining DTD from XML documents. *Proceedings Second International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems. WECWIS 2000*, (Xml):144–151, 2000. [cited at p. 85]

[80] P. Muncaster. Cyber gang made £30 million from fake gov certs. Technical report, 2012 http://www.theregister.co.uk/2012/07/26/fake_qualifications_scam_busted. [cited at p. 109]

[81] Y. Navon, E. Barkan, and B. Ophir. A generic form processing approach for large variant templates. In *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition*, ICDAR '09, pages 311–315, Washington, DC, USA, 2009. IEEE Computer Society. [cited at p. 22, 25, 49]

[82] E. Oro and M. Ruffolo. XONTO: An Ontology-Based System for Semantic Information Extraction from PDF Documents. *2008 20th IEEE International Conference on Tools with Artificial Intelligence*, (i):118–125, Nov. 2008. [cited at p. 24]

[83] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. Giffin, and S. Jha. Automatic generation of remediation procedures for malware infections. In *Proc. of the 19th Usenix Security Symposium, Washington, DC*, 2010. [cited at p. 61]

[84] H. Peng, F. Long, and Z. Chi. Document image recognition based on template matching of component block projections. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 25(9):1188–1192, 2003. [cited at p. 23]

[85] P. Prakash, M. Kumar, R. Kompella, and M. Gupta. Phishnet: Predictive blacklisting to detect phishing attacks. In *INFOCOM, 2010 Proceedings IEEE*, pages 1 –5, march 2010. [cited at p. 61]

[86] P. Prasse, C. Sawade, N. Landwehr, and T. Scheffer. Learning to Identify Regular Expressions that Describe Email Campaigns. In *Proceedings International Conference on Machine Learning (ICML)*, 2012. [cited at p. 65]

[87] N. Provos, P. Mavrommatis, M. A. Rajab, and F. Monrose. All your iframes point to us. In *Proceedings of the 17th conference on Security symposium*, SS'08, pages 1–15, Berkeley, CA, USA, 2008. USENIX Association. [cited at p. 97]

[88] B. Ross. Probabilistic pattern matching and the evolution of stochastic regular expressions. *Applied Intelligence*, pages 285–300, 2000. [cited at p. 63]

[89] H. Sako, M. Seki, N. Furukawa, H. Ikeda, and A. Imaizumi. Form reading based on form-type identification and form-data recognition. In *Proceedings of the Seventh International Conference on Document Analysis and Recognition - Volume 2*, page 926. IEEE Computer Society, 2003. [cited at p. 22, 45]

[90] L. K. Saul, S. Savage, G. M. Voelker, and L. Jolla. Identifying Suspicious URLs: An Application of Large-Scale Online Learning. In *Proceedings of the 26th International Conference on Machine Learning*, 2009. [cited at p. 111]

[91] F. Schulz, M. Ebbecke, M. Gillmann, B. Adrian, S. Agne, and A. Dengel. Seizing the treasure: Transferring knowledge in invoice analysis. In *Proceedings of the 2009 10th International Conference on Document Analysis and Recognition - Volume 00*, pages 848–852. IEEE Computer Society, 2009. [cited at p. 45]

[92] H. Shiu, J. Fong, and R. Biuk-Aghai. Recovering data semantics from XML documents into DTD graph with SAX. In *Proceedings of the 5th WSEAS . . .*, volume 2006, pages 491–496, 2006. [cited at p. 85]

[93] E. Sorio, A. Bartoli, G. Davanzo, and E. Medvet. Open world classification of printed invoices. In *Proceedings of the 10th ACM symposium on Document engineering*, DocEng '10, pages 187–190, New York, NY, USA, 2010. ACM. [cited at p. 12, 15, 18, 23, 25, 27]

[94] E. Sorio, A. Bartoli, G. Davanzo, and E. Medvet. A domain knowledge-based approach for automatic correction of printed invoices. In *Information Society (i-Society), 2012 International Conference on*, pages 151–155. IEEE, 2012. [cited at p. 13, 15]

[95] E. Sorio, A. Bartoli, and E. Medvet. A look at hidden web pages in italian public administrations. In *Computational Aspects of Social Networks (CASoN), 2012 Fourth International Conference on*, pages 291–296. IEEE, 2012. [cited at p. 14, 15, 109]

[96] I. Sourdis, J. a. Bispo, J. a. M. P. Cardoso, and S. Vassiliadis. Regular Expression Matching in Reconfigurable Hardware. *Journal of Signal Processing Systems*, 51(1):99–121, 2007. [cited at p. 61]

[97] B. Svingen. Learning Regular Languages Using Genetic Programming. In J. R. Koza, W. Banzhaf, K. Chellapilla, K. Deb, M. Dorigo, D. B. Fogel, M. H. Garzon, D. E. Goldberg, H. Iba, and R. Riolo, editors, *Genetic Programming 1998 Proceedings of the Third Annual Conference*, pages 374–376. Morgan Kaufmann, 1998. [cited at p. 63, 85]

[98] K. Taghva, R. Beckley, and J. Coombs. The effects of ocr error on the extraction of private information. In H. Bunke and A. Spitz, editors, *Document Analysis Systems VII*, volume 3872 of *Lecture Notes in Computer Science*, pages 348–357. Springer Berlin / Heidelberg, 2006. 10.1007/11669487_31. [cited at p. 49]

[99] K. Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11:419–422, June 1968. [cited at p. 61]

[100] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. *Proceedings of the fourth annual cognitive science conference*, pages 105–108, 1982. [cited at p. 63]

[101] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. Markatos, and S. Ioannidis. Regular expression matching on graphics hardware for intrusion detection. In E. Kirda, S. Jha, and D. Balzarotti, editors, *Recent Advances in Intrusion Detection*, volume 5758 of *Lecture Notes in Computer Science*, pages 265–283. Springer Berlin / Heidelberg, 2009. 10.1007/978-3-642-04342-0_14. [cited at p. 61]

[102] M. Villegas and N. Bel. From DTD to relational dB. An automatic generation of a lexicographical station out off ISLE guidelines. In *Language Resources and Evaluation 2002*, pages 694–700, 2002. [cited at p. 85]

[103] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition).* [cited at p. 83]

[104] W3C. *W3C XML Schema Definition Language (XSD) 1.1.* [cited at p. 83]

[105] C. Whittaker, B. Ryner, and M. Nazif. Large-Scale Automatic Classification of Phishing Pages. In *Network and IT Security Conference: NDSS 2010*, 2008. [cited at p. 111, 112]

[106] M. L. Wick, M. G. Ross, and E. G. Learned-Miller. Context-sensitive error correction: Using topic models to improve ocr. In *ICDAR'07*, pages 1168–1172, 2007. [cited at p. 49]

[107] Wikipedia. Codice fiscale — Wikipedia, the free encyclopedia, 2011. [Online; accessed 22-december-2011]. [cited at p. 51]

[108] T. Wu and W. Pottenger. A semi-supervised active learning algorithm for information extraction from textual data. *Journal of the American Society for Information Science and Technology*, 56(3):258–271, 2005. [cited at p. 64]

[109] T.-F. Wu, C.-J. Lin, and R. C. Weng. Probability estimates for multi-class classification by pairwise coupling. *J. Mach. Learn. Res.*, 5:975–1005, December 2004. [cited at p. 29]

[110] H. Zhao, W. Meng, Z. Wu, V. Raghavan, and C. Yu. Fully automatic wrapper generation for search engines. *Proceedings of the 14th international conference on World Wide Web WWW 05*, WWW '05: P:66, 2005. [cited at p. 18, 20, 21]