**UNIVERSITÀ DEGLI STUDI DI TRIESTE**
**Sede Amministrativa del Dottorato di Ricerca**

XXVI Ciclo - Dottorato in
Ingegneria dell'Informazione

# Genetic Programming Techniques in Engineering Applications

(Settore scientifico-disciplinare ING-INF/05)

DOTTORANDO
**Andrea De Lorenzo**

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 2012/2013

# Contents

# Abstract

Machine learning is a suite of techniques that allow developing algorithms for performing tasks by generalizing from examples. Machine learning systems, thus, may automatically synthesize programs from data. This approach is often feasible and cost-effective where manual programming or manual algorithm design is not. In the last decade techniques based on machine learning have spread in a broad range of application domains.

In this thesis, we will present several novel applications of a specific machine Learning technique, called Genetic Programming, to a wide set of engineering applications grounded in real world problems. The problems treated in this work range from the automatic synthesis of regular expressions, to the generation of electricity price forecast, to the synthesis of a model for the tracheal pressure in mechanical ventilation. The results demonstrate that Genetic Programming is indeed a suitable tool for solving complex problems of practical interest. Furthermore, several results constitute a significant improvement over the existing state-of-the-art.

The main contribution of this thesis is the design and implementation of a framework for the automatic inference of regular expressions from examples based on Genetic Programming. First, we will show the ability of such a framework to cope with the generation of regular expressions for solving text-extraction tasks from examples. We will experimentally assess our proposal comparing our results with previous proposals on a collection of real-world datasets. The results demonstrate a clear superiority of our approach. We have implemented the approach in a web application that has gained considerable interest and has reached peaks of more 10 000 daily accesses.

Then, we will apply the framework to a popular "regex golf" challenge, a competition for human players that are required to generate the shortest regular expression solving a given set of problems. Our results rank in the top 10 list of human players worldwide and outperform those generated by the only existing algorithm specialized to this purpose.

Hence, we will perform an extensive experimental evaluation in order to compare our proposal to the state-of-the-art proposal in a very close and long-established research field: the generation of a Deterministic Finite Automata (DFA) from a labelled set of examples. Our results demonstrate that the existing state-of-the-art in DFA learning is not suitable for text extraction tasks.

We will also show a variant of our framework designed for solving text processing tasks of the search-and-replace form. A common way to automate search-and-replace is to describe the region to be modified and the desired changes through a regular expression and a replacement expression. We will propose a solution to automatically produce both those expressions based only on examples provided by user. We will experimentally assess our proposal on real-word search-and-replace tasks. The results indicate that our proposal is indeed feasible.

Finally, we will study the applicability of our framework to the generation of schema based on a sample of the eXtensible Markup Language documents. The eXtensible Markup Language documents are largely used in machine-to-machine interactions and such interactions often require that some constraints are applied to the contents of the documents. These constraints are usually specified in a separate document which is often unavailable or missing. In order to generate a missing schema, we will apply and will evaluate experimentally our framework to solve this problem.

In the final part of this thesis we will describe two significant applications from different domains. We will describe a forecasting system for producing estimates of the next day electricity price. The system is based on a combination of a predictor based on Genetic Programming and a classifier based on Neural Networks. Key feature of this system is the ability of handling outliers—i.e., values rarely seen during the learning phase. We will compare our results with a challenging baseline representative of the state-of-the-art. We will show that our proposal exhibits smaller prediction error than the baseline.

Finally, we will move to a biomedical problem: estimating tracheal pressure in a patient treated with high-frequency percussive ventilation. High-frequency percussive ventilation is a new and promising non-conventional mechanical ventilatory strategy. In order to avoid barotrauma and volutrauma in patience, the pressure of air insufflated must be monitored carefully. Since measuring the tracheal pressure is difficult, a model for accurately estimating the tracheal pressure is required. We will propose a synthesis of such model by means of Genetic Programming and we will compare our results with the state-of-the-art.

# Riassunto

Il Machine Learning è una serie di tecniche che permettono di sviluppare algoritmi per svolgere dei compiti generalizzandoli da degli esempi. I sistemi di Machine Learning, quindi, possono sintetizzare automaticamente dei programmi da dei dati. Questo approccio è spesso fattibile e conveniente dove invece non lo sono la programmazione manuale o la progettazione manuale di algoritmi. Nell'ultima decade le tecniche basate sul Machine Learning si sono diffuse in una vasta gamma di domini applicativi.

In questa tesi verranno presentate diverse nuove applicazioni di una specifica tecnica di Machine Learning, chiamata Genetic Programming, ad una vasta gamma di applicazioni di ingegneria collegate con il mondo reale. I problemi trattati in questo lavoro spaziano dalla sintesi automatica di espressioni regolari, alla generazione di previsioni per il prezzo dell'energia elettrica, alla sintesi di un modello per la pressione tracheale nella ventilazione meccanica. I risultati dimostrano che il Genetic Programming è effettivamente uno strumento adatto per risolvere problemi complessi di interesse pratico. Inoltre, diversi risultati costituiscono un significativo miglioramento rispetto allo stato dell'arte esistente.

Il principale contributo di questa tesi è la progettazione e l'implementazione di uno strumento per la deduzione automatica di espressioni regolari da esempi basato sul Genetic Programming. Inizialmente verrà mostrata l'abilità di questo framework di far fronte alla generazione automatica di espressioni regolari per risolvere il compito di estrarre testo a partire da esempi. La proposta sarà valutata sperimentalmente confrontando i risultati su una collezione di dati reali con quelli delle proposte precedenti. I risultati mostreranno una chiara superiorità dell'approccio. L'approccio è stato anche implementato in una applicazione web che ha suscitato un notevole interesse e ha raggiunto picchi di oltre 10 000 accessi giornalieri.

Dopo, lo strumento verrà applicato alla popolare sfida "regex golf", una competizione per giocatori umani ai quali è richiesto di generare l'espressione regolare più corta che risolve un dato insieme di problemi.

Quindi, sarà effettuata una valutazione sperimentale estensiva al fine di confrontare lo strumento proposto in questa tesi con la proposta allo stato dell'arte in un campo di ricerca molto vicino e ormai consolidato: la generazione di Automi Finiti Deterministici (DFA) da un insieme etichettato di esempi. I risultati mostrano che lo stato dell'arte

esistente nell'apprendimento di DFA non è adatto per i compiti di estrazione di testo.

Sarà anche mostrata una variante dello strumento progettata per risolvere il compito di elaborazione del testo nella forma della ricerca e sostituzione. Un modo comune per automatizzare la ricerca e sostituzione è di descrivere la regione da modificare e il cambiamento desiderato per mezzo di una espressione regolare e di una espressione di sostituzione. Verrà proposta una soluzione per produrre automaticamente entrambe queste espressioni basandosi solo su esempi forniti dall'utente. La proposta verrà valutata sperimentalmente su dei compiti reali di ricerca e sostituzione.

Infine, verrà studiata l'applicabilità del nostro strumento alla generazione di schemi basandosi su campioni di documenti eXtensible Markup Language. I documenti eXtensible Markup Language sono largamente utilizzati nelle interazioni macchina-macchina e queste interazioni spesso richiedono che alcuni vincoli siano applicati al contenuto dei documenti. Questi vincoli sono solitamente specificati in un documento separato che spesso è non disponibile o mancante. Al fine di generare uno schema quando questo è mancante, verrà applicato e valutato sperimentalmente lo strumento per risolvere questo problema.

Nella parte finale di questa tesi verranno descritte due applicazioni significative da due domini differenti. Verrà descritto un sistema di previsione per produrre stime del prezzo del giorno dopo dell'energia elettrica. Il sistema si fonda su una combinazione di un predittore basato sul Genetic Programming e un classificatore basato su reti neurali. Caratteristica principale di questo sistema è la sua abilità di gestire i valori anomali—valori raramente visti nella fase di apprendimento. I risultati saranno confrontati con un riferimento molto competitivo e rappresentativo dello stato dell'arte. Verrà mostrato che la proposta esibisce errori di predizioni minore del riferimento.

Infine, verrà spostata l'attenzione su un problema biomedicale: stimare la pressione tracheale nei pazienti trattati con la ventilazione percussiva ad alta frequenza. La ventilazione percussiva ad alta frequenza è una nuova e promettente strategia non convenzionale di ventilazione meccanica. Per evitare barotrauma e volutrauma nei pazienti, la pressione tracheale dell'aria insufflata deve essere controllata attentamente. Siccome misurare la pressione tracheale è difficile, è necessario un modello per stimare accuratamente la pressione tracheale. Verrà proposta una sintesi di questo modello per mezzo del Genetic Programming e i risultati verranno confrontati con lo stato dell'arte.

Chapter 1

# Introduction

Machine learning encloses a variety of techniques and strategies able to discover general conjectures and knowledge from specific data. This ability allows developing algorithms to perform complex tasks by generalizing from examples and observations. In other words, machine learning may automatically generate programs from data. Machine learning provides a very attractive alternative to the manual design of algorithms, specially when this is expensive in terms human effort and human time. The growth in avaiable computational power and the large amount of data available today have made the machine learning approach feasible and cost-effective. Nowadays machine learning has spread in a wide range of application domains. Its techniques are broadly employed in applications such as web search, spam filters, recommender systems, advertising, natural language processing, fraud detection, forecasting, drug design, medical diagnosis and many other.

One of the most promising techniques in field of machine learning is the Genetic Programming (GP). GP is a computational paradigm inspired by biological evolution [48] in which a solution for a given problem is encoded as a computer program. Initially, in GP, a *population* of computer programs is generated at random starting from a predefined set of building blocks. Each such program is called an *individual*. The ability of any individual to solve the problem of interest is measured by the *fitness function* and called fitness. Individuals that exhibit highest fitness are selected for producing a new population. The new population is obtained by recombining the selected individuals through certain genetic operators, such as "crossover" and "mutation". These steps constitute a *generation*. This process is iterated until either a solution with perfect fitness is found or some termination criterion is satisfied, e.g., a predefined maximum number of generations have evolved.

In this thesis we will present applications of GP to a collection of practical problems, problems which range from the inference of regular expressions to forecasting of time series and estimation of physical models. The results demonstrate that Genetic Programming is indeed a suitable tool for solving complex problems of practical interest. Furthermore, several results constitute a significant improvement over the existing state-of-the-art. The applications presented in this thesis have been published in international journals

and conferences.

## 1.1 Thesis outline

In chapter 2 we propose a system for the automatic generation of regular expressions for text-extraction tasks. The user describes the desired task only by means of a set of labeled examples. The generated regular expressions may be used with common engines such as those which are part of Java, PHP, Perl and so on. Usage of the system does not require any familiarity with regular expressions syntax. 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, even in comparison to earlier state-of-the-art proposals. The results are highly promising toward the achievement of 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 modern IT technology. Moreover, we have implemented the approach in a public available web application which has been very well received and has gathered peak of more than 10 000 visits per day. These results have been published in the international journal IEEE Computer [10] and the international conference ACM Genetic and Evolutionary Computation Conference (GECCO) [8].

Chapter 3 describes the application of system presented in chapter 2 to the so-called "regex golf" problem. Regex golf has recently emerged as a specific kind of code golf, i.e., unstructured and informal programming competitions aimed at writing the shortest code solving a particular problem. A problem in regex golf consists in writing the shortest regular expression which matches all the strings in a given list and does not match any of the strings in another given list. The regular expression is expected to follow the syntax of a specified programming language, e.g., Javascript or PHP. In this chapter, we propose a *regex golf player* internally based on GP. We generate a population of candidate regular expressions represented as trees and evolve such population based on a multi-objective fitness which minimizes the errors and the length of the regular expression. We assess experimentally our player on a popular regex golf challenge consisting of 16 problems and compare our results against those of a recently proposed algorithm—the only one we are aware of. Our player obtains scores which improve over the baseline and are highly competitive also with respect to human players. The time for generating a solution is usually in the order of tens minutes, which is arguably comparable to the time required by human players.

The problem of synthesizing a Deterministic Finite Automata (DFA) automatically, based on a set of labelled examples, is long-established and the literature abounds of proposals in this area. The state of the art is focussed on a class of so-called Abbadingo-style problems, which are not inspired by any real application. Recent proposals in the area of automatic synthesis of regular expressions from examples have obtained results of potentially practical relevance on text extraction problems, by following a radically different line of research. Since a DFA may be converted to a regular expression, two key questions arise: are research results in the area of DFA learning practically

useful for text extraction problems? how do they compare to the state of the art in text extraction? In chapter 4 of this thesis we address these questions by performing an extensive experimental evaluation of two state-of-the-art proposals of the respective approaches on real datasets. Our analysis suggests that the state of the art in DFA learning is not competitive for solving text extraction problems. We believe this result is insightful for assessing the actual relevance of Abbadingo-style problems and, perhaps, for explaining their lack of practical impact.

Search-and-replace is a text processing task which may be largely automated with regular expressions: the user must describe with a specific formal language the regions to be modified (search pattern) and the corresponding desired changes (replacement expression). Writing and tuning the required expressions requires high familiarity with the corresponding formalism and is typically a lengthy, error-prone process. In chapter 5 we propose a tool based on GP for generating automatically both the search pattern and the replacement expression based only on examples. The user merely provides examples of the input text along with the desired output text and does not need any knowledge about the regular expression formalism nor about GP. We are not aware of any similar proposal. We experimentally evaluated our proposal on 4 different search-and-replace tasks operating on real-world datasets and found good results, which suggests that the approach may indeed be practically viable. These results have been published in the international conference ACM Genetic and Evolutionary Computation Conference (GECCO) [29]

The eXtensible Markup Language (XML) is an essential ingredient of modern web technology and is widely used for describing structured documents to be exchanged in machine-to-machine interactions. The specific constraints to be enforced by a specific application are described in a separate schema document. Although availability of a schema for a specific application is very important, for example, for automating input validation processing, in practice many applications either do not have any schema or the corresponding schema is incomplete. For this reason, several proposals have been made for synthesizing a schema based on a sample of the XML documents used by the specific application. In chapter 6 we describe the design, implementation and experimental evaluation of a tool that solves this problem based on GP. 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.

Chapter 7 focuses on the time series forecasting. The electric power market is increasingly relying on competitive mechanisms taking the form of day-ahead auctions, in which buyers and sellers submit their bids in terms of prices and quantities for each hour of the next day. Methods for electricity price forecasting suitable for these contexts are crucial to the success of any bidding strategy. Such methods have thus become very important in practice, due to the economic relevance of electric power auctions. In this

thesis we propose a novel forecasting method based on GP. Key feature of our proposal is the handling of outliers, i.e., regions of the input space rarely seen during the learning. Since a predictor generated with GP can hardly provide acceptable performance in these regions, we use a classifier that attempts to determine whether the system is shifting toward a difficult-to-learn region. In those cases, we replace the prediction made by GP by a constant value determined during learning and tailored to the specific subregion expected. We evaluate the performance of our proposal against a challenging baseline representative of the state-of-the-art. The baseline analyzes a real-world dataset by means of a number of different methods, each calibrated separately for each hour of the day and recalibrated every day on a progressively growing learning set. Our proposal exhibits smaller prediction error, even though we construct one single model, valid for each hour of the day and used unmodified across the entire testing set. We believe that our results are highly promising and may open a broad range of novel solutions. These results have been published in the international conference EuroGP [9]

Finally in chapter 8 we move our attention to a biomedical problem: the estimation of tracheal pressure during the HFPV mechanical ventilation. High-frequency percussive ventilation (HFPV) is a non-conventional mechanical ventilatory strategy which has proven useful in the treatment of a number of pathological conditions. HFPV usually involves the usage of endotracheal tubes (EET) connecting the ventilator circuit to the airway of the patient. The pressure of the air flow insufflated by HFPV must be controlled very accurately in order to avoid barotrauma and volutrauma. Since the actual tracheal pressure cannot be measured, a model for estimating such a pressure based on the EET properties and on the air flow properties whom can actually be measured in clinical practice is necessary. In this chapter we propose a novel methodology, based on GP, for synthesizing such a model. We experimentally evaluated our models against the state-of-the-art baseline models, crafted by human experts, and found that our models for estimating tracheal pressure are significantly more accurate. These results have been published in the international conference Image and Signal Processing and Analysis (ISPA) [1]

## 1.2 Publication list

- [9] A. Bartoli, G. Davanzo, A. De Lorenzo, and E. Medvet. Gp-based electricity price forecasting. In *Genetic Programming*, pages 37–48. Springer Berlin Heidelberg, 2011

- [63] E. Medvet, A. Bartoli, G. Davanzo, and A. De Lorenzo. Automatic face annotation in news images by mining the web. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 47–54. IEEE Computer Society, 2011

- [8] 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

- [29] A. De Lorenzo, E. Medvet, and A. Bartoli. Automatic string replace by examples. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1253–1260. ACM, 2013

- [10] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio. Automatic synthesis of regular expressions from examples. 2013

- [1] M. Ajcevic, A. De Lorenzo, A. Accardo, A. Bartoli, and E. Medvet. A novel estimation methodology for tracheal pressure in mechanical ventilation control. In *Image and Signal Processing and Analysis (ISPA), 2013 8th International Symposium on*, pages 695–699. IEEE, 2013

Chapter 2

# Regular expressions generation for text extraction

## 2.1 Overview

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 [94] and continue to be a routinely used tool due to their expressiveness and flexibility [19]. Regular expressions have become an essential device in broadly different application domains, including construction of XML schemas [14, 15], extraction of bibliographic citations [23], network packets rewriting [44], network traffic classification [96, 11], signal processing hardware design [91], malware [77, 24] and phishing detection [81] 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 6.2), 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.

## 2.2 Related work

The problem of synthesizing a regular expression or deterministic finite automata (DFAs) [60] from a set of examples is long-established (e.g., [20]) 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 [22] 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 [22]. Interestingly, our results improve those of the cited work even in terms of flagging precision and recall.

Problem and fitness definition in [7] and [39] are more similar to ours. The proposal in [7] 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 [39], 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 [32], whereas the generation of regular expressions was proposed in [93] and applied to the Tomita benchmark languages [95]. Stochastic regular expressions, also applied to the Tomita languages, were considered in [87]. 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 [106]. 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 [33]. This limitation is not present in the active learning algorithm proposed in [46], 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 [53]. 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 [5]. 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 [19], 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 [53] (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, [53, 5, 19]). 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 [50]. 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 [14, 15]. 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 [82]. 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.

## 2.3 Problem statement

We define the problem of text extraction by examples. Let $E_G$ be a *problem generator entity* which poses the problem to a *problem solver entity* $E_S$, whose aim is to find a solution. The problem is described by: (i) a tuple $\mathcal{I}$ which univocally identifies a *problem instance*; (ii) a procedure followed by $E_G$ to generate from $\mathcal{I}$ the *learning information* $\mathcal{L}$ available to $E_S$; (iii) the *metrics* to be maximized by $E_S$.

Text extraction involves input strings constructed over a large alphabet $\alpha = \text{UTF-8}$. A problem instance is defined by $\mathcal{I} = (T, h)$. The *dataset* $T$ is composed of pairs of strings $\langle t, t' \rangle$, where $t'$ is a substring which has to be extracted from $t$—$t'$ may be the empty string $\emptyset$. We call *positive* example a pair in which $t' \neq \emptyset$, *negative* example otherwise. The number $h < |T|$ is a positive integer.

$E_G$ generates the learning information $\mathcal{L} = (T^{\mathcal{L}})$ by uniformly sampling $T$ such that $|T^{\mathcal{L}}| = h$. The sampling is done so as to ensure that positive and negative examples in $T^{\mathcal{L}}$ are balanced.

The objective of $E_S$ consists in proposing a solution $r^*$ which maximizes the following metrics on $T^E = T \setminus T^{\mathcal{L}}$ (note that $T^E$ is *not* available to $E_S$):

$$\text{precision} := \frac{\left|\left\{t \in T^E : E(t; r^*) = t' \neq \emptyset\right\}\right|}{\left|\left\{t \in T^E : E(t; r^*) \neq \emptyset\right\}\right|}$$

$$\text{recall} := \frac{\left|\left\{t \in T^E : E(t; r^*) = t' \neq \emptyset\right\}\right|}{\left|\left\{t \in T^E : t' \neq \emptyset\right\}\right|}$$

$$\text{F-measure} := 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

where $E(t; r^*)$ is the leftmost substring of $t$ extracted by $r^*$.

In practical settings, $E_G$ is a user and $E_S$ is a regular expression generation tool. $E_G$ will typically use the generated expression not only on $T$, but also on a larger dataset. In other words, despite being $T^{\mathcal{L}}$ a uniform sampling of $T$, no guarantees exist about whether $T$ is indeed a "good model" of the regular expression actually needed by the user. Consider, for example, a user who wants to generate a regular expression for extracting dates. The dataset $T$ could include many "recent" dates (e.g., `2013-11-10`, `2011-02-07`, ...), which could not be an uniform sampling of all possible dates. Of course, this *generalization issue* is intrinsic to any realistic machine learning application.

## 2.4 Our approach

### 2.4.1 User experience

The user provides a set of examples, each composed by a pair of strings $\langle t, t' \rangle$ where $t$ is a text line and $t'$ is the substring of $t$ that must be extracted by the regular expression. A pair where $t'$ 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 [35] 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

| 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 2.1:** Function set

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.

### 2.4.2 Implementation

Every individual of the genetic programming (GP) search process is a tree $\tau$. 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 2.1, 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 2.1: (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 2.2.

A tree $\tau$ is transformed into a string $R_\tau$ which represents a regular expression by means of a depth-first post order visit. In detail, $R_\tau := \textsc{Node2Regex}(\textsc{root}(\tau))$, where the function Node2Regex is defined in Algorithm 1: $\textsc{child}(N, i)$ denotes the $i$-th child of node $N$ and $\textsc{replace}(t_1, t_2, t_3)$ is a function that substitutes string $t_2$ with string $t_3$ in string $t_1$. A simple example is shown in Fig. 2.1.

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

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

**Table 2.2:** Terminal set

**Algorithm 1** 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 ≤ 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
```

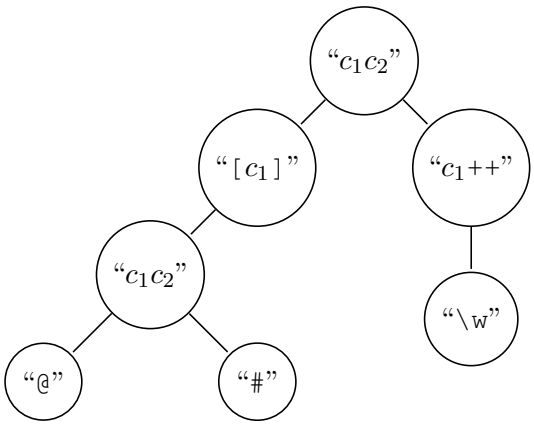



**Figure 2.1:** Tree representation of the “[@#]\w++” regular expression.

| 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 2.3:** Sample quantifiers

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 corresponding 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:

$$f_d(r) = \sum_{i=1}^{n} d(t'_i, E_r(t_i)) \tag{2.1}$$

$$f_l(r) = l(r) \tag{2.2}$$

where: (i) $t_i$ is the $i$-th example text line in a set of $n$ given examples, (ii) $t'_i$ is the substring to be found in $t_i$, (iii) $E_r(t_i)$ is the leftmost 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) [31].

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 [50, 22]. 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 [7, 39]) and obtained very good results. A more systematic comparison between different fitness definitions is given in the experimental evaluation.

### 2.4.3 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 2.3): *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 **??**). 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 2.4, 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.

## 2.5 Experiments

### 2.5.1 Extraction tasks and datasets

We considered 12 different datasets, 2 of which are taken from [53], and [19] and other 2 are taken from [22]. 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 [53] 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 [22] 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 2.5. 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 [54] (used by [53]).

**ReLIE Phone Number** Extract phone numbers from a collection of 10,000 emails obtained from the publicly available Enron email collection [73] (used by [53, 19]).

**Cetinkaya HREF** Extract the HTML attribute HREF from the HTML source of a set of 3 web pages (used by [22]).

**Cetinkaya URL** Extract URLs from a set of 3 web pages (used by [22]).

**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 [61].

**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 [62].

**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.

**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.

### 2.5.2 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 2.4; (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

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

| 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 2.4:** GP parameters

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.

### 2.5.3 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. 2.2). 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 [53, 19]. 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 the training set size. The results are in Fig. 2.3(a)

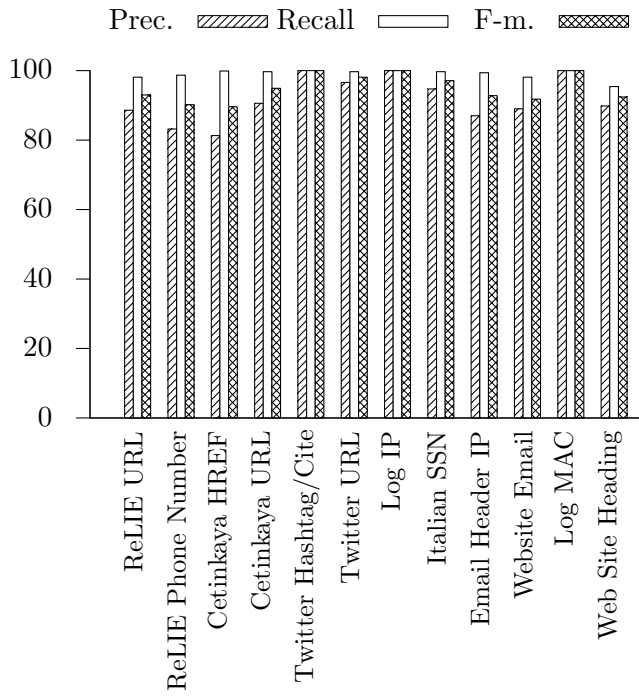
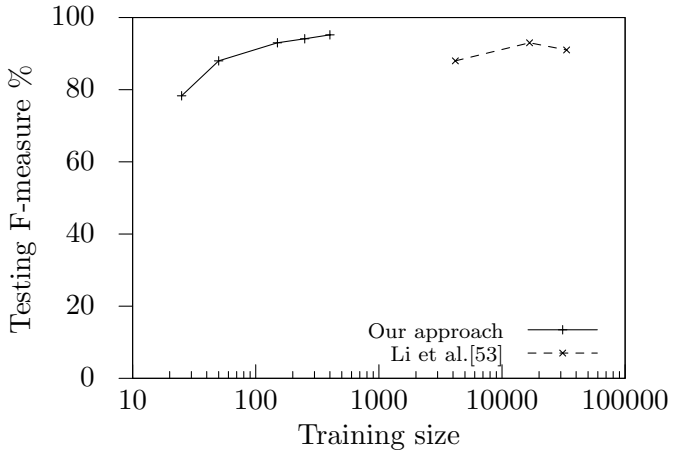


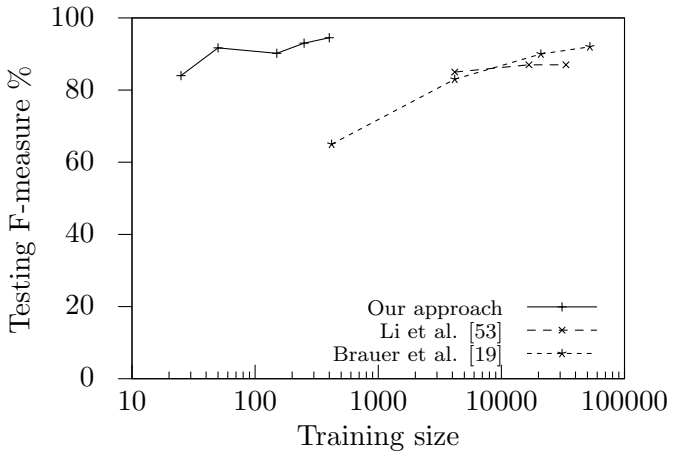

**Figure 2.2:** Experiment results, in terms of Precision, Recall and F-Measure, of each task

| 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 2.5:** Dataset compositions




(a) ReLIE URL task




(b) ReLIE Phone Number task

**Figure 2.3:** Analysis of tasks ReLIE URL and ReLIE Phone Number. Performance comparison between our approach and earlier state-of-the-art proposals.

for ReLIE URL and in Fig. **??** 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 $5 \times 128$ individuals that compose the final populations of the five folds and reported the corresponding performance distributions in Fig. 2.4 and Fig. 2.5 (learning set with 100 examples and $J = 128$, i.e., the experiment in Fig. 2.2). 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 2.6:** Experiment results with different learning size

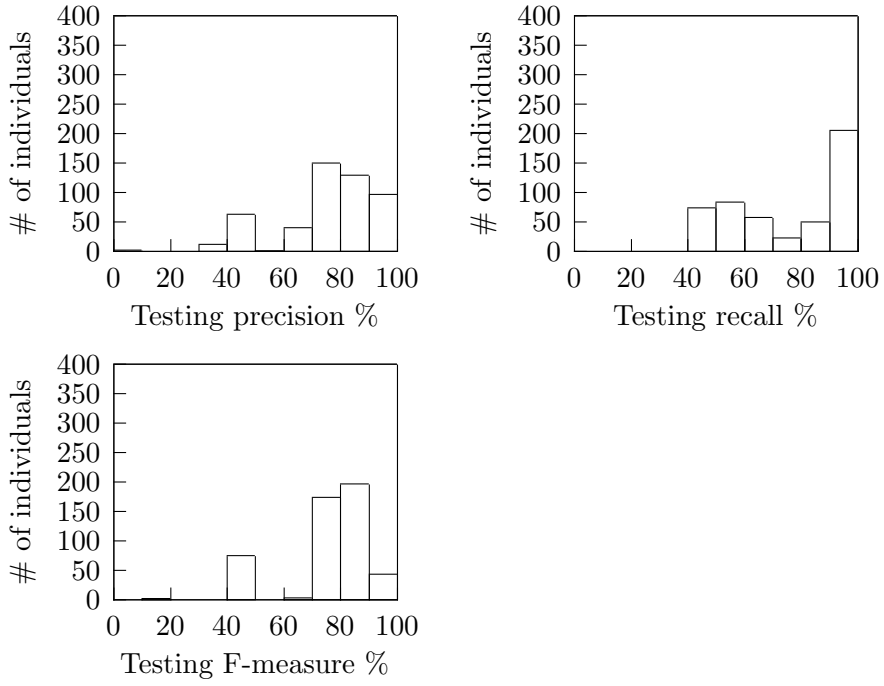



**Figure 2.4:** Distributions of precision, recall and F-measure on the testing set of ReLIE URL task.

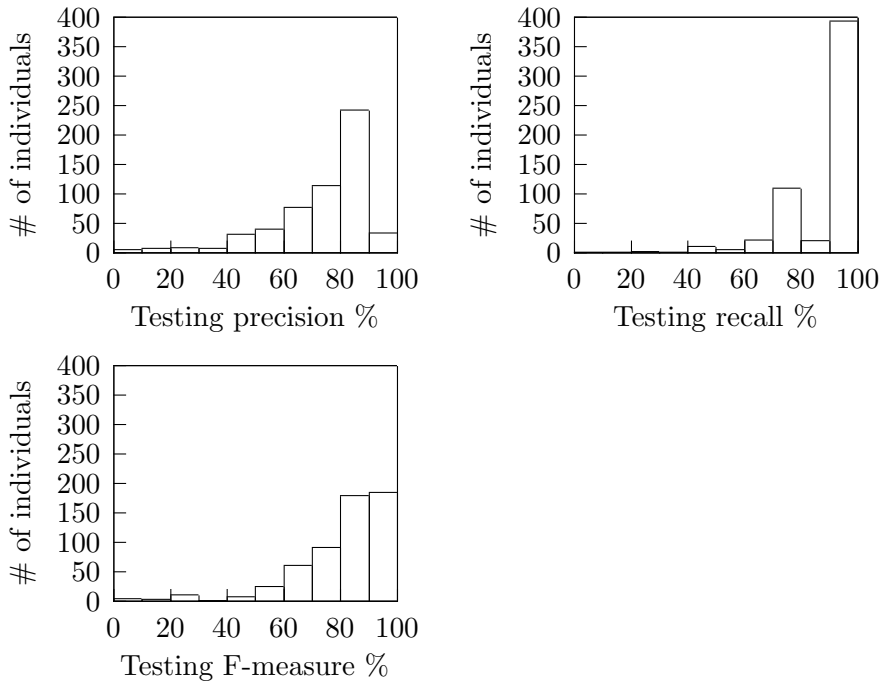



**Figure 2.5:** Distributions of precision, recall and F-measure on the testing set of ReLIE Phone Number task.

The datasets of tasks Cetinkaya HREF and Cetinkaya URL were also used in earlier relevant works in the context of a flagging problem [22]: 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 [22] 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 2.7. Our results exhibit better performance, which is interesting because: (i) the regular expressions in [22] 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 2.6. 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 2.6 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 2.6 by four. In this perspective, we decided to set $J = 32$ in the prototype of our system available at http://regex.inginf.units.it.

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. (6.1) and (8.5):

$$f(r) = \sum_{i=1}^{n} d(t'_i, E_r(t_i)) + \alpha l(r) \tag{2.3}$$

| Approach | Cetinkaya HREF | Cetinkaya URL |
|---|---|---|
| Cetinkaya [22] | 99.97 | 76.07 |
| Our approach | 100.00 | 99.64 |

**Table 2.7:** Comparison between our approach and the one presented in [22] (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 2.8:** 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.

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 2.4.2).

*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.

$Edit + \alpha Length$ a linear combination of the objectives, with varying values for the $\alpha$ parameter (Eqn. 2.3);

$Edit + \alpha Depth$ 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 2.9. 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. 2.3(a) and Fig. **??** 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 $\alpha$ provide expressions that are shorter and of comparable quality.

Finally, we show a sample of the expressions generated by our system in Table 2.8. The table has one row for each of the previous experiments 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.

## 2.6 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

| Fitness | $\alpha$ | 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 + $\alpha$Length | 0.01 | 97.40 | 99.50 | 98.44 | 30 |
| Edit + $\alpha$Length | 0.10 | 97.36 | 99.50 | 98.42 | 28 |
| Edit + $\alpha$Length | 1.00 | 97.36 | 99.50 | 98.42 | 28 |
| Edit + $\alpha$Depth | 0.01 | 97.41 | 99.50 | 98.44 | 51 |
| Edit + $\alpha$Depth | 0.10 | 97.67 | 99.50 | 98.58 | 55 |
| Edit + $\alpha$Depth | 1.00 | 97.41 | 99.50 | 98.44 | 47 |

**Table 2.9:** Performance indexes and average length $l$ of the resulting regular expressions for different fitness functions and values for the $\alpha$ parameter used for weighing the two objectives. MO indicates a Multi-Objective approach.

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 3

# Regex golf

## 3.1 Overview

Regex golf has recently emerged as a specific kind of code golf, i.e., unstructured and informal programming competitions aimed at writing the shortest code solving a particular problem. A problem in regex golf usually consists in writing the shortest regular expression which matches all the strings in a given list and does not match any of the strings in another given list. Examples of such lists could be the names of all winners of an US presidential election and of the names of all losers (the specific constraints on the contents of these lists will be clarified later, e.g., their intersection must be empty). A trivial way for generating systematically a regular expression with these requirements consists in building a disjunction of all the desired matches—i.e., all the matches glued together by the | character which, in common regex syntax, means "or". To reward non trivial solutions, the *score* assigned to a given solution is higher for more compact expressions.

There has recently been a growing interest toward regex golf in the programmers' communities, motivated more by the challenge itself than by the actual utility of any given problem. Such interest has been fueled further by a blog post of a famous researcher—Peter Norvig—in which a simple yet powerful algorithm for solving regex golf problems systematically is proposed [76]. Norvig points out that problems of this sort are related to set cover problems, which are known to be NP-hard, and describes a greedy algorithm which is very efficient and works well in a number of cases, while at the same time identifying the fundamental trade-offs made in his proposal.

In this thesis work, we propose a methodology based on Genetic Programming (GP) for generating solutions to regex golf problems—a *regex golf player*. We generate a population of candidate regular expressions represented as trees and constructed with carefully selected regular expression operators and constants. We evolve such population based on a multi-objective fitness which maximizes the correct handling of the provided matches and unmatches while minimizing the length of the regular expression represented by the individual.

We implemented our proposal and assessed its performance on a recently proposed

suite of 16 regex golf problems which is very popular. We used as baseline the algorithm proposed by Norvig—the only one we are aware of—and an existing GP-based system for generating regular expressions for text extraction tasks by examples [10]. Our proposal compares very favorably to the baseline and obtains the highest score on the full suite. We also attempted the construct a baseline based on scores obtained by human players, which is difficult because no structured collections of human players results are available: however, we collected several results by crawling the web and found that our proposal is ranked in the top positions.

A prototype of our regex golf player is available at http://regex.inginf.units.it/golf.

## 3.2 Related Work

The only algorithm explicitly designed for solving regex golf-related problems which we are aware of is the one by Peter Norvig mentioned in the introduction. We used this algorithm as baseline for our proposal.

Several proposals for learning regular expressions from examples exist for *text extraction* problems [106, 53, 7, 46, 19, 8, 10]. Text extraction from examples is radically different from regex golf in several crucial points. First, regex golf assumes an input stream segmented so that the input strings listed in the problem specification are processed by the solution one segment at a time. Text extraction requires instead the ability to identify and extract specific portions from a longer stream. In other words, regex golf consists in binary classifying input strings whereas text extraction requires the identification in the input string of the boundaries of the substring(s) to extract, if any. Second, a regex golf problem places no requirements on how strings not listed in the problem specification will be classified. Text extraction requires instead a form of generalization, i.e., the ability of inducing a general pattern from the provided examples. Third, text extraction requires the ability to identify a context for the desired extraction, that is, a given sequence of characters may or may not constitute a match depending on its surroundings. A requirement of this form is not meaningful in regex golf.

For example, a regex golf problem requiring the match of all winners of US presidential elections and no loser may be solved with a disjunction of `1s` and several short regexes [76]. Such a regular expression is not useful for the text extraction problem, because applying it to a superstring of a winner would provide no information about the substring which actually identifies the winner. Furthermore, any string containing the substring `1s` will thus be matched by the regex. On the other hand, a regex generated for text extraction might be applied to regex golf but it would be largely suboptimal: the solution generation process must induce a general pattern and there is clearly no syntactical pattern capable of predicting the names of future US presidents. In other words, learning approaches tailored to text extraction purposefully attempt to prevent any overfitting of the examples which is instead a necessity in regex golf.

Our proposal builds on the text extraction method in [10], which we modified and specialized by taking the specific requirements of regex golf into account. We included the cited method in the baseline because, although it was designed for a different problem, it

is available as a webapp[1] and its inclusion demonstrates that solving regex golf effectively calls for a specialized solution.

Another proposal for learning regular expressions from examples is [22], but this work considers a problem whose requirements are a mix of regex golf and text extraction. On the one hand, the problem consists in merely classifying input strings without the need of identifying the boundaries of the matching substrings. On the other hand, the problem assumes input streams not necessarily segmented in advance at the granularity of the desired matches and unmatches. Moreover, and most importantly, the cited work aims at inferring a general pattern capable of solving the desired task beyond the provided examples.

Since a regular expression may be obtained from a deterministic finite automata (DFA), approaches for learning a DFA from labelled examples and counterexamples could be used (e.g., [60, 18]; see [26] for a survey). On the other hand, such proposals assume the number of states of the target DFA is known and, most importantly, they are not concerned with minimizing the length of the regular expression corresponding to the generated DFA. While approaches of this form may deserve further investigation, they do not appear to match the specific requirements of regex golf. Similar remarks may be applied also to proposals for induction of non-deterministic finite automata (NFA) from labelled examples [36, 104].

Finally, regex golf might be seen as a problem in the broader category of *programming by examples* (PBE), where a program in a given programming language is to be synthesized based on a set of input-output pairs. Notable results in this area have been obtained recently for problems of string manipulation [40, 66] and some of the corresponding algorithms have been included in the latest release of Microsoft Excel (Flash-Fill functionality). While such approaches are able to deal with *context-free grammars* and are thus potentially able to solve classification problems of the form encountered in regex golf, they use an underlying language which is much richer than regular expressions and thus may not generate solutions useful for regex golf.

## 3.3 The Problem

While the term "regex golf" may indicate any challenge requiring the generation of a regular expression, its usual meaning is the one described in the introduction and formalized as follows.

We consider strings constructed over a large alphabet $\alpha$ = UTF-8. Strings may potentially include arbitrary characters in the alphabet, including spaces, newline and so on. A problem instance is defined by $\mathcal{I} = (M, U)$, where $M$ and $U$ are sets of strings whose intersection is empty.

The problem consists in generating a regular expression which:

1. matches all strings in $M$;
2. does not match any string in $U$; and,

[1] http://regex.inginf.units.it

3. is shorter than the regular expression constructed as a disjunction[2] of all strings in $M$.

Note that, for a given problem instance, it might not be known whether a regular expression satisfying the above requirements actually exists. Furthermore, given a solution $r'$ satisfying the three requirements, it might not be known whether there exists a shorter solution $r''$ satisfying requirements 1 and 2.

Solutions may satisfy requirements 1 and 2 in part. That is, a solution might fail to match one or more strings in $M$ and/or match one or more strings in $U$. Solutions are thus given a *score* quantifying their behavior in terms of the desired matches and unmatches, as well as their compactness.

We use the score definition in http://regex.alf.nu, from which we have also collected the suite of problem instances for our experimental evaluation. The definition is as follows. Let $r$ be a candidate solution, let $n_M$ and $n_U$ denote the number of elements in $M$ and $U$, respectively, which are matched by $r$. The score of $r$ on instance $\mathcal{I} = (M, U)$ is:

$$w_{\mathcal{I}}(n_M - n_U) - \text{length}(r)$$

where $w_{\mathcal{I}}$ is a statically defined value which is meant to weigh the "difficulty" of the problem instance $\mathcal{I}$. Note that the numerical value of the score, as well as the range of possible values, is problem instance-dependent and that a solution may obtain a negative score.

## 3.4 Our Approach

The system requires a description of the problem instance $\mathcal{I} = (M, U)$ and generates a Javascript-compatible regular expression. A prototype is available at http://regex.inginf.units.it/golf.

Our proposal builds on the text extraction method in [10], which we modified and specialized by taking the specific requirements of regex golf into account. We will summarize the differences at the end of this section.

Every individual of the GP search process is a tree $\tau$, where labels of leaf nodes are taken from a specified *terminal set* and labels of internal nodes from a specified *function set* as follows.

The function set consists of the following regular expressions operators (the central dot $\cdot$ represents a placeholder for a child node): *possessive quantifiers* ($\cdot*+$, $\cdot++$, $\cdot?+$ and $\cdot\{\cdot,\cdot\}+$), *group* `(·)`, *character class* `[·]` and *negated character class* `[^·]`, *concatenator* $\cdot\cdot$—a binary node which concatenates its children—and *disjunction* $\cdot|\cdot$. We did not include greedy or lazy quantifiers [35] because, as indicated in [10], these operators lead to execution times which are not practically acceptable.

The terminal set consists of a set of terminals which do not depend on the problem instance $\mathcal{I}$ and other terminals which depend on $\mathcal{I}$. Instance independent terminals are:

[2] More precisely, the disjunction of all strings in $M$, where each string is prefixed by the "start of string" anchor `^` and postfixed by the "end of string" anchor `$`.

**Table 3.1:** Salient information for the 16 problems.

| | Problem name | $\|M\|$ | $\|U\|$ | $w_{\mathcal{I}}$ | Ideal score |
|---|---|---|---|---|---|
| 1 | Plain strings | 21 | 21 | 10 | 210 |
| 2 | Anchors | 21 | 21 | 10 | 210 |
| 3 | Ranges | 21 | 21 | 10 | 210 |
| 4 | Backrefs | 21 | 21 | 10 | 210 |
| 5 | Abba | 21 | 22 | 10 | 210 |
| 6 | A man, a plan | 19 | 21 | 10 | 190 |
| 7 | Prime | 20 | 20 | 15 | 300 |
| 8 | Four | 21 | 21 | 10 | 210 |
| 9 | Order | 21 | 21 | 10 | 210 |
| 10 | Triples | 21 | 21 | 30 | 630 |
| 11 | Glob | 21 | 21 | 20 | 420 |
| 12 | Balance | 32 | 32 | 10 | 320 |
| 13 | Powers | 11 | 11 | 10 | 110 |
| 14 | Long count | 1 | 20 | 270 | 270 |
| 15 | Long count v2 | 1 | 21 | 270 | 270 |
| 16 | Alphabetical | 17 | 17 | 20 | 340 |
| | Total | | | | 4320 |

the alphabetical *ranges* `a-z` and `A-Z`, the start of string *anchor* `^` and the end of string anchor `$`, and the *wildcard character* `.`. Instance dependent terminals are: all characters appearing in $M$, *partial ranges* appearing in $M$, and *n-grams*.

Partial ranges are obtained as follows. We (i) build the sequence $C$ of all characters appearing in $M$ (without repetitions), sorted according to natural order; (ii) for each maximal subsequence of $C$ which includes *all* characters between subsequence head $c_h$ and tail $c_t$, build a partial range $c_h$-$c_t$. For example, if $M = \{\texttt{bar}, \texttt{den}, \texttt{foo}, \texttt{can}\}$, then the partial ranges are `a-e` and `n-o`.

n-grams are obtained as follows. We (i) build the set $N$ of all n-grams occurring in $M$ and $U$ strings, with $2 \leq n \leq 4$; (ii) give a score to each n-gram as follows: $+1$ for each string in $M$ which contains the n-gram and $-1$ for each string $U$ which contains the n-gram; (iii) sort $N$ according to descending score and (iv) select the smallest subset $N'$ of all top-scoring n-grams such that the sum of their scores is at least $|M|$ and each individual score is positive. For example, if $M = \{\texttt{can}, \texttt{banana}, \texttt{and}, \texttt{ball}\}$ and $U = \{\texttt{indy}, \texttt{call}, \texttt{name}, \texttt{man}\}$, then the n-grams are `an` and `ba`, as they are the two top-scoring n-grams and the sum of their scores is $2 + 2$.

A tree $\tau$ is transformed into a string $r_\tau$ which represents a regular expression by means of a depth-first post order visit—Figure 3.1 shows an example of a tree and the corresponding regular expression (in the caption). In our implementation, each regular expression is evaluated by the Java regular expression engine, which works with possessive quantifiers. However, the regex golf competition being considered accepts only Javascript-compatible regular expressions and Javascript regular expression engine does not work

**Table 3.2:** Best human score and solutions for the 16 problems.

| | Problem name | Best human score | Best human solution |
|---|---|---|---|
| 1 | Plain strings | 207 | `foo` |
| 2 | Anchors | 208 | `k$` |
| 3 | Ranges | 202 | `^[a-f]*$` |
| 4 | Backrefs | 201 | `(...).*\1` |
| 5 | Abba | 193 | `^(?!.*(.)(.)\2\1)` |
| 6 | A man, a plan | 177 | `^(.)[^p].*$` |
| 7 | Prime | 286 | `^(?!(..+)\1$` |
| 8 | Four | 199 | `(.)(.\1){3}` |
| 9 | Order | 199 | `^.5[^e]?$` |
| 10 | Triples | 596 | `00($|3|6|9|12|15)|4.2|.1.+4|55|.17` |
| 11 | Glob | 397 | `ai|c$|^p|[bcnrw][bnopr]` |
| 12 | Balance | 289 | `^(<(<(<(<<?>?>|.9)>)>)>)$` |
| 13 | Powers | 93 | `^(?!(.(..)+)\1*$)` |
| 14 | Long count | 254 | `((.+)0 \2?1 ){7}` |
| 15 | Long count v2 | 254 | `((.+)0 \2?1 ){7}` |
| 16 | Alphabetical | 317 | `.r.{32}r|a.{10}te|n.n..` |
| | Total | 4072 | |

with possessive quantifiers. Hence, we further transform $r_\tau$ into a Javascript-compatible regular expression by means of a mechanical transformation [35].

The initial population is generated as follows. Let $n_P = 500$ be the size of the population to be generated. For each string $s$ in $M$, we generate an individual corresponding to $s$, built using only the concatenator node and single characters of $s$ as terminals. We generate the remaining $n_P - |M|$ individuals randomly, with the ramped half-and-half method and depth of 1–5 levels.

We drive the evolutionary search based on two fitness indexes associated with each individual. Let $r$ be an individual and let $n_M$ and $n_U$ be the number of elements in $M$ and $U$, respectively, which are matched by $r$. The two fitness indexes are: $n_M - n_U$, which has to be maximized (the upper bound being $|M|$), and the length of $r$ (in the Javascript-compatible version), which has to be minimized. We use the *Non-Dominated Sorting Genetic Algorithm II* (NSGA-II) [31] to rank individuals according to their fitness values.

We evolve the population for a number of generations $n_g = 1000$, according to the following iterative procedure. Let $P$ be the current population. We generate an evolved population $P'$ as follows: 10% of the individuals are generated at random, 10% of the individuals are generated by applying the genetic operator "mutation" to individuals of $P$, and 80% of the individuals are generated by applying the genetic operator "crossover" to a pair of individuals of $P$. We select individuals for mutation and crossover with a *tournament* of size 7, i.e., we pick 7 individuals at random and then select the best individual in this set, according to NSGA-II. Finally, we generate the next population by

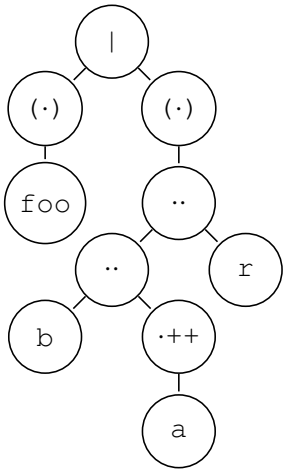



**Figure 3.1:** Tree representation of the regular expression `(foo)|(ba++r)`.

choosing the individuals with highest fitness among those in $P$ and $P'$. The size of the population is kept constant during the evolution. Upon generation of a new individual, we check the syntactic correctness of the corresponding expression: if the check fails, we discard the individual and generate a new one.

In order to generate a solution for a problem instance $\mathcal{I}$, we evolve $n_e = 32$ independent populations, with different random seeds, obtaining 32 candidate regular expressions. Finally, we choose the regular expression with the highest score.

We remark the key features of our proposal (w.r.t. [10]):

1. a method for constructing the terminal set based on the problem instance $\mathcal{I}$, rather than being defined once and for all;

2. a method for initializing the population based on the problem instance $\mathcal{I}$, rather than being completely random;

3. a different functions set which includes, in particular, a disjunction operator—which is difficult to use in text extraction because it tends to promote overfitting;

4. fitness definitions based on the number of examples handled correctly—definitions proven to be inadequate for text extraction [10];

5. usage of all learning information for synthesizing candidate solutions, that is, without reserving any partition as validation set for assessing the generalization capabilities of those solutions.

## 3.5 Experimental Evaluation

We considered the 16 problem instances along with the accompanying scores proposed in http://regex.alf.nu. Salient properties of these instances are summarized in Table 3.1

and 3.2. The Table 3.1 shows, for each problem instance, the ideal score—i.e., the score equal to $w_{\mathcal{I}}|M|$ which could be obtained with a zero-length regular expression matching all strings in $M$ and no strings in $U$. The Table 3.2 shows, for each problem instance, the highest score obtained by (different) human players[3] and the corresponding regular expression.

### 3.5.1 Baseline

We used as baseline the algorithm by Peter Norvig, which we call Norvig-RegexGolf, and the system for generating regular expressions for text extraction presented in [10], which we call GP-RegexExtract.

We provide a brief outline of Norvig-RegexGolf below. Full details, including the (partially for fun) motivations and design trade-offs can be found in [76]. The solution for a given problem instance $\mathcal{I} = (M, U)$ is obtained as a disjunction of a set of *components*, a component being a short regular expression which matches at least one string in $M$ and does not match any string in $U$. Initially, a pool of components is built with several heuristics, including the generation of a component for each n-gram of each string in $M$ (up to $n$=4) and, for each such component, the generation of a component for every possible substitution of a single character with the dot character (meaning "match any" in regular expression syntax). Next a set of components from that pool is built, such that each string in $M$ is matched by at least one component in the set. Components in the resulting set are then glued together by the or regular expression operator `|`.

Concerning GP-RegexExtract, we reimplemented the algorithm according to the details presented in [10] (see also Section 3.2). We set those parameters which determine the computational weight of GP to the same values for GP-RegexExtract and GP-RegexGolf, in order to allow a fair comparison of the results w.r.t. computational weight: $n_e = 32$, $n_g = 500$ and $n_P = 500$. Note that GP-RegexExtract requires that examples are partitioned in a training set and a validation set: while using it as a regex golf player, we chose to use half of $M$ and half of $U$ strings as training set, and the remaining string as validation set.

### 3.5.2 Results

We executed each of the algorithms on each problem instance and computed the score of the corresponding solution. Tables 3.3, 3.4 and 3.5 summarize the resulting scores, which are presented as absolute value and as the percentage of the ideal and the best human score associated with each problem instance. Table 3.6 shows the regular expressions generated by GP-RegexGolf for each problem.

It can be seen that GP-RegexGolf outperforms both Norvig-RegexGolf and GP-RegexExtract: 3090 vs. −665 and 249, respectively. In particular, considering individual problem instances, GP-RegexGolf performs better than Norvig-RegexGolf in 6 problems, worse in 8 problems and obtains the same score in 2 problems. Despite obtaining a better

[3]The information is obtained from https://gist.github.com/jonathanmorley/8058871.

**Table 3.3:** Results of the Norvig-RegexGolf as score, score %, score % w.r.t. to best human score and competitive ratio (C.R., see text). For each problem, the score of the best algorithm is shown in bold.

| | Norvig-RegexGolf | | | |
|---|---|---|---|---|
| Problem | Score | Score % | Hum. % | C.R. |
| 1 | **207** | 98.6 | 100.0 | 66.3 |
| 2 | **208** | 99.1 | 100.0 | 105.5 |
| 3 | 191 | 91.0 | 94.6 | 8.5 |
| 4 | **175** | 83.3 | 87.0 | 8.0 |
| 5 | **186** | 88.6 | 96.4 | 11.5 |
| 6 | **157** | 82.6 | 88.7 | 5.1 |
| 7 | −398 | $< 0$ | $< 0$ | 1.0 |
| 8 | **192** | 91.4 | 96.5 | 17.5 |
| 9 | **190** | 90.5 | 95.5 | 8.7 |
| 10 | **589** | 93.5 | 98.8 | 6.1 |
| 11 | **392** | 93.3 | 98.7 | 25.2 |
| 12 | −1457 | $< 0$ | $< 0$ | 1.0 |
| 13 | −1969 | $< 0$ | $< 0$ | 1.0 |
| 14 | 189 | 70.0 | 74.4 | 1.0 |
| 15 | 189 | 70.0 | 74.4 | 1.0 |
| 16 | **294** | 86.5 | 92.7 | 18.8 |
| Total | −665 | - | - | - |

score in 8 problems, Norvig-RegexGolf obtains a negative score on the full suite because on three problems (7, 12 and 13) it is not able to generate a non trivial solution: in these problems, the regular expression generated by Norvig-RegexGolf is the disjunction of all the $M$ strings. Our algorithm, on the contrary, generates non trivial solutions for these problems. Concerning GP-RegexExtract, both its score on the full suite and its score on individual instances make it clear that this approach does not the requirements of regex golf.

Tables 3.3, 3.4 and 3.5 list also, for each algorithm, the *competitive ratio* of the solutions [76], defined as the ratio between the length of a trivial solution (disjoining all the strings in $M$) and the length of the corresponding solution. Note that this index does not take matches or unmatches into account. It can be seen that both Norvig-RegexGolf and GP-RegexGolf generate solutions which are much shorter than the trivial solution for several problems: regular expressions generated by GP-RegexGolf are shorter than those of Norvig-RegexGolf in 9 problems, longer in 5 problems and with the same length in 2 problems.

Table 3.7 shows the time required by GP-RegexGolf and GP-RegexExtract for generating a solution. It is similar for all the problems (around 50 min) with the exception of 13, 14 and 15. For the latter problems, in which $M$ is composed by very long strings, GP-RegexExtract attempts to generate a regular expression which extracts (rather than

**Table 3.4:** Results of the GP-RegexGolf algorithm as score, score %, score % w.r.t. to best human score and competitive ratio (C.R., see text). For each problem, the score of the best algorithm is shown in bold.

| | GP-RegexGolf | | | |
|---|---|---|---|---|
| Problem | Score | Score % | Hum. % | C.R. |
| 1 | **207** | 98.6 | 100.0 | 66.3 |
| 2 | **208** | 99.1 | 100.0 | 105.5 |
| 3 | **195** | 92.9 | 96.5 | 10.7 |
| 4 | 138 | 65.7 | 68.7 | 6.7 |
| 5 | 184 | 87.6 | 95.3 | 17.2 |
| 6 | 136 | 71.6 | 76.8 | 7.0 |
| 7 | **188** | 35.3 | 37.0 | 24.1 |
| 8 | 183 | 87.1 | 92.0 | 11.7 |
| 9 | 186 | 88.6 | 93.5 | 7.3 |
| 10 | 430 | 68.3 | 72.2 | 12.6 |
| 11 | 340 | 81.0 | 85.6 | 17.7 |
| 12 | **130** | 40.6 | 45.0 | 11.1 |
| 13 | **51** | 46.4 | 54.8 | 109.4 |
| 14 | **191** | 70.7 | 75.2 | 1.0 |
| 15 | **191** | 70.7 | 75.2 | 1.0 |
| 16 | 132 | 38.8 | 41.6 | 8.0 |
| Total | **3090** | - | - | - |

**Table 3.5:** Results of the Gp-RegexExtract algorithm as score, score %, score % w.r.t. to best human score and competitive ratio (C.R., see text). For each problem, the score of the best algorithm is shown in bold.

| | GP-RegexExtract | | | |
|---|---|---|---|---|
| Problem | Score | Score % | Hum. % | C.R. |
| 1 | 170 | 81.0 | 82.1 | 5.0 |
| 2 | 185 | 88.1 | 88.9 | 8.4 |
| 3 | 107 | 51.0 | 53.0 | 7.0 |
| 4 | −70 | $< 0$ | $< 0$ | 4.0 |
| 5 | 77 | 36.7 | 39.9 | 4.4 |
| 6 | −246 | $< 0$ | $< 0$ | 0.7 |
| 7 | −52 | $< 0$ | $< 0$ | 13.4 |
| 8 | −45 | $< 0$ | $< 0$ | 7.0 |
| 9 | −39 | $< 0$ | $< 0$ | 4.5 |
| 10 | −106 | $< 0$ | $< 0$ | 2.4 |
| 11 | −163 | $< 0$ | $< 0$ | 4.3 |
| 12 | −85 | $< 0$ | $< 0$ | 20.9 |
| 13 | −47 | $< 0$ | $< 0$ | 44.2 |
| 14 | **191** | 70.7 | 75.2 | 1.0 |
| 15 | **191** | 70.7 | 75.2 | 1.0 |
| 16 | 181 | 53.2 | 57.1 | 11.0 |
| Total | 249 | - | - | - |

**Table 3.6:** Regular expressions generated by GP-RegexGolf.

| Problem | Regular expression |
|---|---|
| 1 | `foo` |
| 2 | `k$` |
| 3 | `(^..[a-f][a-f])` |
| 4 | `v\|[^b][^o][^p]t\|ngo\|lo\|[n]o\|rp\|rb\|ro\|ro\|rf` |
| 5 | `z\|.u\|nv\|st\|ca\|it` |
| 6 | `oo\|x\|^k\|ed\|^m\|ah\|^r\|v\|^t` |
| 7 | `^(?=((?:x[A-Zx])+))\1x` |
| 8 | `ell\|j\|W\|ele\|o.o\|Ma\|si\|de\|do` |
| 9 | `ch\|[l-p]o\|ad\|fi\|ac\|ty\|os` |
| 10 | `24\|55\|02\|54\|00\|95\|17` |
| 11 | `lo\|ro\|^p\|(?=((c)+))\1r\|en\|^w\|y.\|le\|^p\|rr` |
| 12 | `((?=((?:<<\>\>\>)*))\2(?=((?:<<<(?=(<*))\4\>\><<<<)*))\3 (?=((?:<<<<<\>\>\>(?=(<*))\6\>\>\>)*))\5(?=((?:<<<<<<)*)) \7^(?=((?:<<\><<)*))\8(?=((?:<<<\>\>\>)*))\9<<)` |
| 13 | `^(?=(((x\|^)x)+))\1$` |
| 14 | `0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111` |
| 15 | `0000 0001 0010 0011 0100 0101 0110 0111 1000 1001 1010 1011 1100 1101 1110 1111` |
| 16 | `tena\|[^et][^etren](?=((?:(?:ren\|eren.(?=((?:(?:ren\|[^ren] ))+))\2\|eren.(?=((?:(?:ren\|[^ren]))+))\3))+))\1\|eas` |

**Table 3.7:** Execution times of GP-RegexGolf and GP-RegexExtract algorithms in minutes.

| Problem | GP-RegexGolf | GP-RegexExtract |
|---|---|---|
| 1 | 53 | 51 |
| 2 | 52 | 52 |
| 3 | 53 | 65 |
| 4 | 38 | 25 |
| 5 | 34 | 18 |
| 6 | 20 | 23 |
| 7 | 33 | 43 |
| 8 | 19 | 27 |
| 9 | 46 | 21 |
| 10 | 45 | 25 |
| 11 | 44 | 47 |
| 12 | 56 | 71 |
| 13 | 71 | 269 |
| 14 | 94 | 289 |
| 15 | 95 | 173 |
| 16 | 66 | 64 |
| Total | 820 | 1262 |

just matching) the each $M$ string entirely: this leads to a population composed by very long regular expressions which require long times to be evaluated. The time required by Norvig-RegexGolf is practically negligible (less than a second per problem). All the experiments have been performed on a quad core Intel Xeon E5-2440 (2.40GHz) with 4 GB of RAM.

We wanted to investigate whether our approach can achieve better scores at the expense of increased computational weight. To this end, we repeated the experiments by setting $n_P = 1000$ and $n_P = 1500$, i.e., with an enlarged population: Table 3.8 shows the results in terms of score and competitive ratio. It can be seen that the full score does improve for larger values of $n_P$. Moreover, with $n_P = 1500$, the number of problems for which GP-RegexGolf score is not worse than Norvig-RegexGolf score is 11 vs. 8 with $n_P = 500$. The computation time for the full suite goes from 820 min for $n_P = 500$ to 1551 min and 2611 min for $n_p = 1000$ and $n_p = 1500$, respectively. As expected, with higher values for $n_P$ GP-RegexGolf takes longer to generate a solution: yet, it is fair to claim that even such longer times may be acceptable for playing to a game of this kind.

Finally, we attempted to assess the performance of our proposal with respect to scores of highly skilled human players. We remark that there are several caveat concerning the assessment of the results obtained by human players. The web site hosting the challenge does not, at the time of this writing, provide a score ranking computed on the full suite of problems—on the other hand, there exists a collection of "Best possible answers collected so far for regex golf" (see Table 3.1) which shows, for each problem, the best solution. Results by human players are advertised on web forums by players themselves, often

**Table 3.8:** Scores and competitive ratio (C.R.) of GP-RegexGolf with different values for $n_P$.

| | $n_P = 1000$ | | $n_P = 1500$ | |
|---|---|---|---|---|
| Problem | Score | C.R. | Score | C.R. |
| 1 | 207 | 66.3 | 207 | 66.3 |
| 2 | 208 | 105.5 | 208 | 105.5 |
| 3 | 196 | 11.5 | 197 | 12.4 |
| 4 | 146 | 5.2 | 147 | 6.5 |
| 5 | 188 | 12.5 | 186 | 11.5 |
| 6 | 142 | 6.0 | 151 | 4.3 |
| 7 | 188 | 24.1 | 188 | 24.1 |
| 8 | 183 | 11.7 | 180 | 10.5 |
| 9 | 190 | 8.7 | 190 | 8.7 |
| 10 | 456 | 10.5 | 354 | 27.2 |
| 11 | 355 | 28.2 | 522 | 3.2 |
| 12 | 36 | 7.3 | 223 | 26.5 |
| 13 | 65 | 46.2 | 40 | 29.7 |
| 14 | 191 | 1.0 | 191 | 1.0 |
| 15 | 191 | 1.0 | 191 | 1.0 |
| 16 | 259 | 21.1 | 237 | 10.4 |
| Total | 3201 | - | 3412 | - |

without providing any actual evidence of their results. On the other hand, there are players which do make some very good solutions publicly available, thereby simplifying the job of other players, which may either attempt to improve those solutions further or may use them for the corresponding problem instance while focusing their efforts on the remaining instances. In other words, the score obtained by a given player may actually result from efforts by multiple players. Finally, human players generally do not care to indicate the time they spent for generating a solution.

We collected several human players scores on the full suite from different web locations (including Reddit, Hacker News, Github) which we obtained by querying Google and Twitter with the search string "regex golf". Table 3.9 shows the 10 best scores we found, along with the total ideal score (i.e., the sum of ideal scores on the 16 problems), the best human score (i.e., the sum of the highest human player scores on the 16 problems) and the score of the three considered algorithms.

It can be seen that GP-RegexGolf would rank from 6th to 8th among human players (with $n_P = 1500$ and $n_P = 500$, respectively), whereas the scores of the other two algorithms are significantly lower than those of human players. In other words, leaving aside any caveat about how we gathered human scores, GP-RegexGolf is in the top ten of worldwide regex golf players.

**Table 3.9:** Total scores obtained by the 10 best humans and by the three algorithms.

| | Player | Score |
|---|---|---|
| | *Total ideal score* | 4320 |
| | *Best human score* | 4072 |
| 1 | geniusleonid | 4006 |
| 2 | k_hanazuki | 3785 |
| 3 | bisqwit | 3753 |
| 4 | AlanDeSmet | 3736 |
| 5 | adamhiker | 3693 |
| | *GP-RegexGolf* ($n_P = 1500$) | 3412 |
| | *GP-RegexGolf* ($n_P = 1000$) | 3201 |
| 6 | adamschwartz | 3181 |
| 7 | flyingmeteor | 3171 |
| | *GP-RegexGolf* ($n_P = 500$) | 3090 |
| 8 | jpsim | 3060 |
| 9 | ItsIllak | 2939 |
| 10 | bg666 | 2683 |
| | *GP-RegexExtract* | 249 |
| | *Norvig-RegexGolf* | −665 |

## 3.6 Remarks

We have proposed and assessed experimentally an approach based on Genetic Programming for playing regex golf automatically, i.e., for generating automatically solutions to challenges which have recently become popular in the programmers' communities. The challenges consist in writing the shortest regular expression that matches all strings in a given list and does not match any string in another given list.

Our approach collects a score that is highly competitive against human players and improve significantly over a challenging baseline including a recently proposed algorithm tailored to this specific problem class and the proposal for automatic generation of regular expressions tailored to text extraction tasks presented in chapter 2. The time for generating a solution is in the order of tens of minutes and a prototype is available at http://regex.inginf.units.it/golf.

We think that our work shows how a GP-based approach running on modern IT machinery may deliver results, at least for this task, which are practically useful and can compete with humans.

Chapter 4

# Evolutionary learning of text extraction patterns: Abbadingo-style vs. GP-regex

## 4.1 Overview

The problem of learning Deterministic Finite Automata (DFA) from examples is long established [26]. Research in this area has mostly been driven by synthetic benchmarks and *Abbadingo-style* problems have played a key role in this respect [49, 26]. Abbadingo-style problems involve the learning of a DFA acting as binary classifier of input strings constructed over a binary alphabet. The target DFA is generated randomly and the learning information is sampled uniformly from the input space. The learner knows in advance the number of states of the target DFA as well as the maximum length of input strings.

A problem related to the learning of DFAs from examples is the learning of *regular expressions*. Recent research in this area has produced results of practical relevance in text processing applications [22, 7, 39, 106, 46, 53, 5, 19, 8] demonstrating, for example, high precision and recall on *text extraction* problems such as extraction of HTML headings from web pages and of phone numbers from email messages [10]. This field of research proceeded independently from the proposals for DFA learning and, in particular, was not based on synthetic benchmarks.

Since DFAs and regular expressions are intimately related, two key questions arise: are research results in the area of DFA learning practically useful for text extraction problems? how do they compare to the state of the art in text extraction?

These questions are relevant because Abbadingo-style problems are not inspired by any real world application [26] and the applicability of learning algorithms proposed for Abbadingo-style problems to other application domains is still largely unexplored [18]. Indeed, similar questions have been recently addressed in the software engineering field,

where automated inference of state machines from examples may be of great help for constructing models of software behavior automatically. In this context, Abbadingo-style problems have not been considered to be an adequate framework [101].

In this chapter we compare experimentally two approaches representative of the state of the art in the respective fields. We consider the DFA learning algorithm in [60] which was developed several years after the Abbadingo One competition and outperformed (optimized versions of) the winners of the competition, on the same class of Abbadingo-style problems. We also consider the recently developed *GP-Regex* approach specifically designed for text extraction [10], whose performance improves over early proposals for regular expression learning on the same datasets. We implemented both the approaches and carefully assessed them on different and challenging text extraction problems from real data: URLs, phone numbers, `href` HTML attributes, Twitter hashtags and citations.

Our analysis and experimental evaluation suggest that the GP-Regex is much more suitable for solving text extraction problems by examples than the Abbadingo-style approach. While this outcome might not appear surprising—the former was tailored to text extraction while the latter was not—we do believe the result is important for gaining further insights on the actual relevance of Abbadingo-style problems for a specific application domain.

## 4.2 Problem statements

We define the Abbadingo-style problem. Let $E_G$ be a *problem generator entity* which poses the problem to a *problem solver entity* $E_S$, whose aim is to find a solution. The problem is described by: (i) a tuple $\mathcal{I}$ which univocally identifies a *problem instance*; (ii) a procedure followed by $E_G$ to generate from $\mathcal{I}$ the *learning information* $\mathcal{L}$ available to $E_S$; (iii) the *metrics* to be maximized by $E_S$.

### 4.2.1 Abbadingo-style by examples

Abbadingo-style problems consist in binary classifying input strings constructed over a binary alphabet $\alpha = \{0, 1\}$. A problem instance is defined by $\mathcal{I} = (n, \rho)$, where $n$ is a positive integer representing the number of states of the target DFA and $\rho \in [0, 1]$ is a number called *density* that quantifies the relative amount of learning examples available.

$E_G$ generates the learning information $\mathcal{L} = (n, S^{\mathcal{L}}, \alpha)$ as follows: (i) randomly generate a target DFA $d$ with $n$ states; $d$ either accepts or rejects each input string; (ii) construct the set of *all* binary strings $s$ with length $\ell(s) \leq 2\log_2 n + 3$; (iii) determine the output of $d$ for each such string $s$, which may be $\{\mathsf{accept}, \mathsf{reject}\}$ and is denoted by $M(s; d)$; (iv) select a maximum length for the input strings $2\log_2 n + 3$; (v) build the set $S$ of pairs $\langle s, M(s; d)\rangle$ including all the inputs along with the corresponding labels; (vi) build a subset $S^{\mathcal{L}} \subset S$ by uniformly sampling $S$ and such that $|S^{\mathcal{L}}| = \rho|S|$; the sampling is done so as to ensure that $S^{\mathcal{L}}$ is balanced, i.e., the number of elements with $M(s; d) = \mathsf{accept}$ is equal to the number of elements with $M(s; d) = \mathsf{reject}$.

The objective of $E_S$ consists in proposing a DFA $d^*$ which maximizes the accuracy[1]:

$$\text{accuracy} := \frac{\left|\left\{s \in S^E : M(s; d^*) = M(s; d)\right\}\right|}{|S^E|}$$

We remark three crucial properties of the learning information. First, the number of states $n$ of the target DFA is known to $E_S$—in text extraction problems $E_S$ has no clue about the structure of the pattern to be generated. Second, the symbols of the input alphabet are known to occur all with the same probability. Third, the learning examples $S^{\mathcal{L}}$ are generated by uniformly sampling the set of all possible inputs, thus they are by construction a "good model" of the possible inputs themselves—in practical applications of the text extraction problem, the set of all possible inputs may be much bigger than the set of examples $T$ available to $E_G$.

### 4.2.2 Abbadingo-style for text extraction

In order to learn a DFA suitable for text extraction, we need to define procedures for: (i) obtaining an Abbadingo-style learning information $(n, S^{\mathcal{L}}, \alpha)$ from a text extraction problem instance $(T, h)$; (ii) using the Abbadingo-style solution $d^*$—i.e., a DFA that acts as a binary classifier—as a text extraction tool.

We obtain the learning information $(n, S^{\mathcal{L}}, \alpha)$ from $(T, h)$ as follows: (i) assign $S^{\mathcal{L}}_+$ and $S^{\mathcal{L}}_-$ to the empty set; (ii) for each element $\langle t, t' \rangle \in T^{\mathcal{L}}$, add all the substrings of $t$ different from $t'$ to $S^{\mathcal{L}}_-$ and, if $t'$ is non empty, add $t'$ to $S^{\mathcal{L}}_+$; (iii) assign $S^{\mathcal{L}} = S^{\mathcal{L}}_+ \cup S^{\mathcal{L}}_-$; (iv) assign $n = \max_{\langle t,t' \rangle \in T} \ell(t')$; (v) include in $\alpha$ all the symbols occurring in $T$.

This procedure for obtaining the learning information generates a learning set $S^{\mathcal{L}}$ that may be not balanced, with a number of negative examples typically much larger than the number of positive examples. We executed several experiments with a balanced learning set, as required by the original Abbadingo-style formulation, by selecting one random substring of $t$ for each $\langle t, t' \rangle \in T^{\mathcal{L}}$, rather than all substrings of $t$. We chose to not use a balanced learning set because this strategy exhibited significantly worse performance.

We define $E(t; d^*)$, i.e., the leftmost substring of $t$ extracted by $d^*$, as follows: $E(t; d^*)$ is the leftmost substring $t'$ of $t$ for which $M(t', d^*) = \mathsf{accept}$ (if there are multiple substrings with the same left alignment for which $M(t', d^*) = \mathsf{accept}$, then $E(t; d^*)$ is the longest substring); in case no such substring $t'$ exists, then $E(t; d^*)$ is the empty string.

### 4.2.3 Hardness indicators

The hardness of an Abbadingo-style problem instance $\mathcal{I} = (n, \rho)$ grows with the number of states of the target DFA $n$ and when the density of the examples $\rho$ decreases. This consideration is supported by the results of the Abbadingo competition itself, which consisted of 12 problem instances resulting from the combinations of 4 values for $n$ and

[1] This is the objective used in [60]. The learning competition used a different testing protocol [49] on $S^E = S \setminus S^{\mathcal{L}}$ ($S^E$ is *not* available to $E_S$) but such a difference is irrelevant to this discussion.

3 values for $\rho$. The problems with smallest density $\rho$ remained unsolved, with the only exception of the one with the smallest number of states $n$ [49].

We remark that, when applying an Abbadingo-style algorithm to a text extraction problem instance, $n$ and $\rho$ may *not* be very meaningful hardness indicators. An Abbadingo-style problem involves the learning of a randomly-generated DFA, i.e., one for which the occurrence of a symbol in a given input string provides no information about the next symbol. In a text extraction problem, in contrast, input strings are not composed of symbols drawn uniformly from the alphabet. It follows that the difficulty in constructing a pattern describing the desired extractions does not depend only on, say, the length of the extractions and the number of provided examples with respect to the full size of the dataset. The nature of the examples and of the dataset clearly play a key role in this respect. The problem of developing meaningful hardness indicators for a specific text extraction problem instance is beyond the scope of this technical note.

## 4.3 Approaches

### 4.3.1 GP-Regex

We consider the evolutionary approach based on Genetic Programming proposed in [10], that we call *GP-Regex* and outline below.

A candidate solution (i.e., a regular expression) is represented by an abstract syntax tree. Each leaf node is an element from a predefined terminal set composed of constants (`a`, ..., `z`, `A`, ..., `Z`, `0`, ..., `9`, `@`, `#`, ...) and character classes (`\w`, `0-9`, `a-z` and `A-Z`). Each branch node is an element from a predefined function set which includes: the concatenator $\cdot\cdot$, which concatenates its two children, the character class operators `[`$\cdot$`]` and `[^`$\cdot$`]`, the non-capturing group `(?:`$\cdot$`)` operator and the possessive quantifiers ($\cdot$`*+`, $\cdot$`++`, $\cdot$`?+`, $\cdot${$\cdot$`,`$\cdot$`}+`). A regular expression $r$ is produced from a tree by concatenating node labels encountered in a depth-first post order visit of the tree (the dot character $\cdot$ in the function set represents a placeholder for the children nodes).

The evolutionary search is based on the NSGA-II [31] multi-objective minimization. Each candidate solution $r$ has two *fitness* indexes to be minimized ($t'$ is the substring to be extracted from $t$): $f_E(r) = \sum_{\langle t,t' \rangle \in T^T} L(t', E(t;r))$ and $f_\ell(r) = \ell(r)$, where $L(x,y)$ is the Levenshtein distance between strings $x$ and $y$, $T^T$ is defined below, and $\ell(r)$ is the length of the regular expression $r$.

A partition of $T^{\mathcal{L}}$ into a training set $T^T$ and a validation set $T^V$ is generated, such that: (i) $|T^T| = |T^V|$ and (ii) positive examples are distributed equally between $T^T$ and $T^V$. Then, a set of 128 different and independent GP searches are performed. Each GP search has available only $T^T$ and consists of 500 candidate solutions that evolve for 1000 generations (see the cited paper for full details about the GP search procedure). Among the $128 \times 500$ trees resulting at the end, the generated solution $r^*$ is the one with the highest F-measure on $T^V$:

$$\text{F-measure} := 2\frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

### 4.3.2 SSL-DFA

We consider the evolutionary algorithm for Abbadingo-style problems proposed in [60], which we call *SSL-DFA* after the name adopted in the cited paper (Smart State Labelling Evolutionary Algorithm). We implemented this algorithm for text extraction according to the procedures in the previous section.

A candidate solution (i.e., a DFA) is represented by a pair composed of an output vector of size $n$ and a transition matrix of size $n \times |\alpha|$: the former has one element for each DFA state and each element contains the label $\{\mathsf{accept}, \mathsf{reject}\}$ for the corresponding state; the latter contains, for each state and transition, the corresponding destination state index. The fitness of a candidate solution $d$ is the rate of examples classified correctly: $f(d) = \frac{1}{|S^T|} \left|\left\{\langle s, l\rangle \in S^T : M(s; d) = l\right\}\right|$.

The search algorithm is as follows (with parameter values set as in [60]): (i) a DFA $d$ is built by randomly filling the above representation; (ii) an intermediate DFA $d'$ is built by mutating the current DFA $d$: the one with the highest fitness becomes the new current solution. Step ii is repeated until no fitness increments have shown for a given number of consecutive iterations; then, the procedure restarts from step i. The search terminates when either a DFA with perfect fitness is found or a predefined number of iterations have been executed. The solution $d^*$ is the DFA represented at the end of the procedure.

The size of the search space in SSL-DFA is $n^{n|\alpha|}2^n$, hence it grows exponentially with the alphabet size $|\alpha|$. Text extraction problems involve a very large value for $|\alpha|$. For this reason, we experimented also with a variant of this approach in which solutions are represented in a more compact form. We grouped several alphabet symbols in three classes, similarly to the regular expression formalism: `[A-Z]` containing the uppercase letters, `[a-z]` containing the lowercase letters and `[0-9]` containing the digits. This way, the transition matrix of a solution has a smaller number of rows hence the search space is smaller. We call this variant *SSL-DFA-classes*.

## 4.4 Experiments

We consider the 5 datasets $T$ listed below and 3 values for $h \in \{25, 50, 100\}$, resulting in 15 problem instances $(T, h)$. These datasets have been previously used (with much larger values for $h$) to assess earlier proposals [53, 19, 22, 10].

**ReLIE URL:** $T$ contains the HTML code—split by lines—of several web-pages obtained from the publicly available University of Michigan Web page collection, where URLs have been annotated. The annotation consists of pairing each positive example $t$ with another string $t'$ equal to the substring to be extracted from $t$.

**ReLIE Phone Number:** $T$ contains the body—split by lines—of several emails obtained from the publicly available Enron email collection, where phone numbers have been annotated.

**Table 4.1:** Datasets.

| Dataset $T$ | $\lvert T\rvert$ | $\lvert T_+\rvert$ | $\lvert\alpha\rvert$ | $n$ | $\rho$ |
|---|---|---|---|---|---|
| ReLIE URL | 3877 | 2820 | 92 | 153 | 0.026 |
| ReLIE Ph. N. | 41 832 | 4097 | 93 | 14 | 0.002 |
| Cetinkaya HREF | 3416 | 211 | 101 | 513 | 0.074 |
| Cetinkaya URL | 1233 | 767 | 49 | 49 | 0.063 |
| Twitter | 50 000 | 34 879 | 92 | 42 | 0.002 |

**Cetinkaya HREF:** $T$ contains the HTML code—split by lines—of 3 web-pages, where the HTML attributes `href` have been annotated.

**Cetinkaya URL:** $T$ is the same as above, but URLs have been annotated.

**Twitter:** $T$ contains thousands of Twitter posts (a superset of these posts has been used in [61]), where hashtags and citations have been annotated.

Table 4.1 summarizes the composition of the datasets. The table includes values for the Abbadingo-style learning information: $n$ and $|\alpha|$, computed as in Section 4.2.2, and $\rho$, computed as $|S^{\mathcal{L}}|$ divided by the number of all possible substrings in the dataset $T$. We provide these values for completeness of analysis and remark that their meaning in text extraction problems and in Abbadingo-style problems is very different (Section 4.2.3). The experimental evaluation in [60], concerned with Abbadingo-style problems, considered a binary alphabet, $n \leq 32$ and $\rho$ values in the range $[0.01, 0.2]$.

We applied the three approaches (GP-Regex, SSL-DFA, SSL-DFA-classes) 5 times to each problem instance, varying only the seed for the random generator. The corresponding performance indexes, averaged across the 5 repetitions, are in Table 4.2. It is evident that GP-Regex outperforms the two other approaches in all the problem instances. Furthermore, both SSL-DFA and SSL-DFA-classes exhibit values for precision and recall that are not practically useful.

We wondered if SSL-DFA performed sharply worse than GP-Regex because it had fewer iterations for finding a good solution during the evolutionary search. We verified that 65 on 75 searches terminated before reaching the maximum number of iterations (which we doubled with respect to [60]): hence, in those 65 searches the approach provided its best possible performance and it would not have benefited from a further increase of this bound. Table 4.2 also shows that SSL-DFA-classes does not exhibit better performance than SSL-DFA, with the only exception of Twitter-related tasks.

For completeness of analysis, we assessed performance of the generated solutions on $T^E$ also with requirements weaker than extraction. We considered a positive example $t$ handled correctly when some non-empty string is extracted, *irrespective* of the desired substring $t'$, and measured the accuracy as:

$$\text{accuracy} := \frac{\left|\left\{t \in T^E : (E(t; r^*) \neq \emptyset \wedge t' \neq \emptyset) \vee (E(t; r^*) = t' = \emptyset)\right\}\right|}{|T^E|}$$

**Table 4.2:** Precision and recall, in percentage.

| | | GP-Regex | | SSL-DFA | | SSL-DFA-classes | |
|---|---|---|---|---|---|---|---|
| Dataset $T$ | $h$ | Prec. | Recall | Prec. | Recall | Prec. | Recall |
| ReLIE URL | 25 | 77.3 | 82.5 | 9.8 | 23.5 | 6.1 | 40.7 |
| | 50 | 79.9 | 98.1 | 14.8 | 22.7 | 13.0 | 30.7 |
| | 100 | 88.6 | 98.1 | 15.0 | 25.5 | 15.1 | 27.2 |
| ReLIE Ph. N. | 25 | 80.9 | 90.9 | 30.2 | 10.6 | 17.5 | 9.0 |
| | 50 | 85.4 | 99.2 | 12.1 | 18.3 | 6.8 | 17.8 |
| | 100 | 83.2 | 98.7 | 14.0 | 22.5 | 15.5 | 14.4 |
| Cetinkaya HREF | 25 | 34.5 | 94.8 | 20.1 | 8.5 | 16.0 | 6.5 |
| | 50 | 72.2 | 94.4 | 20.4 | 8.5 | 19.3 | 6.3 |
| | 100 | 81.3 | 99.9 | 10.1 | 30.7 | 8.8 | 27.3 |
| Cetinkaya URL | 25 | 79.4 | 89.6 | 13.2 | 15.2 | 7.3 | 16.2 |
| | 50 | 79.4 | 89.6 | 14.2 | 20.5 | 13.7 | 23.1 |
| | 100 | 90.6 | 99.7 | 15.4 | 17.2 | 24.0 | 11.0 |
| Twitter | 25 | 98.7 | 91.2 | 12.5 | 32.8 | 27.6 | 33.1 |
| | 50 | 99.1 | 95.6 | 16.0 | 34.0 | 28.3 | 32.2 |
| | 100 | 100.0 | 100.0 | 24.1 | 34.6 | 39.9 | 42.6 |

We report for brevity only the accuracy with $h = 100$ in Table 4.3 (results for $h = 25, 50$ are qualitatively similar). Even from this point of view, the results still indicate a clear superiority of GP-Regex.

Finally we report the execution time for each experiment with $h = 100$ averaged across the 5 repetitions. The execution times for the GP-Regex method are: 365 s for ReLIE URL, 421 s for ReLIE Phone, 1017 s for Cetinkaya HREF, 727 s for Cetinkaya URL and 229 s for Twitter. The execution times for the SSL-DFA are (those for SSL-DFA-classes are nearly identical): 183 s, 251 s, 286 s, 225 s and 48 s. Experiments have been executed in parallel on 4 machines with a quad-core Intel Xeon X3323 (2.53 GHz) and 2GB of RAM.

**Table 4.3:** Accuracy, in percentage, assessed when some non-empty substring is extracted, irrespective of the desired substring ($h = 100$).

| Dataset $T$ | GP-Regex | SSL-DFA | SSL-DFA-classes |
|---|---|---|---|
| ReLIE URL | 98.5 | 32.2 | 26.7 |
| ReLIE Ph. N. | 91.0 | 89.5 | 90.3 |
| Cetinkaya HREF | 99.3 | 94.3 | 94.9 |
| Cetinkaya URL | 99.2 | 65.8 | 63.2 |
| Twitter | 100.0 | 30.5 | 30.2 |

## 4.5 Remarks

We have empirically investigated the relevance of research results in the area of DFA learning for solving text extraction problems. We have considered an approach that is representative of the state-of-the-art in DFA learning and exhibits excellent results in so called Abbadingo-style problems (SSL-DFA [60]). We have compared this approach to a recent proposal representative of the state-of-the-art in the learning of regular expressions for text extraction (GP-Regex [10]).

Our analysis indicates that GP-Regex is much more suitable for solving text extraction problems by examples than SSL-DFA. We have currently no evidence of the reason for such sharp difference. Text extraction problems involve values for $\alpha, \rho$ (i.e., alphabet size and relative amount of learning examples) which are much larger than those typically used in Abbadingo-style problems (Section 4.4, discussion of Table 4.1). On the other hand, text patterns are not made up of symbols drawn uniformly from the alphabet thus such parameters are not meaningful indicators of the problem difficulty (Section 4.2.3). We speculate that GP-Regex exhibits much better performance because it represents candidate solutions in a way that is much more compact and much more suitable for describing patterns of interest in text extraction, thereby greatly simplifying the evolutionary exploration of the solution space.

We believe this contribution allows gaining further insights on the actual relevance of Abbadingo-style problems for specific application domains, an issue that has just begun to be addressed in depth [18, 101].

Chapter 5

# Regular expressions generation for search and replace

## 5.1 Overview

Techniques for automated text processing are becoming increasingly important due to the uninterrupted growth and diffusion of text sources that are unstructured or loosely structured, e.g., logs—which exist in many different forms and application domains, including server administration, web access, phone calls, intrusion detection—web catalogs, email messages, social networking sites and so on. A specific text processing task potentially suitable to being automated is *search-and-replace*, where all text regions matching a given pattern should be replaced according to a given scheme. Many tools offer powerful support for this task based on the usage of *regular expressions*, which are a way of specifying a pattern using a formal language. The user must specify a *search pattern* for identifying the text regions to be modified and a separate *replacement expression* for describing the changes to be applied. The replacement expression usually include references to specific subregions of the region to be modified and the description of these subregions must be encoded in the search pattern appropriately. This framework requires the involvement of technically savvy users because defining the expressions required for solving a specific search-and-replace task requires high familiarity with the corresponding formalism. Furthermore, tuning the expressions is usually a time-consuming, error-prone process.

In this thesis, we propose a tool based on Genetic Programming (GP) that is capable of generating both the search pattern and the replacement expression *automatically*, only by means of *examples*. The user merely provides a set of examples of the search-and-replace task, each example consisting of the text before and after the desired replacement without any further annotation. The tool then generates automatically both the search pattern and the replacement expression for fulfilling the task. The output can be used with popular processing engines, e.g., Java, PHP and so on. We emphasize that the user

does not need to have any knowledge about genetic programming nor is she required to provide any hints about the structure of the search pattern or replacement expressions to be obtained. We are not aware of any other method capable of automatically defining the required expressions, based solely on examples.

Our tool internally works in three phases. In the first phase, it executes a GP search for generating a regular expression able to localize the text regions to be processed. This regular expression defines a single pattern across all the provided examples, usually including the text to be modified and some surrounding text, thereby defining a sort of *context* for characterizing the scope of the desired replacements. For instance, consider the anonimization of Twitter usernames (e.g., `@GECCO2013` → `@xxxxxxxxx`): a letter should be replaced by `x`, but only when it is part of a Twitter username—which is the context. This phase is essential for making the user experience as simple as possible: the user merely specifies the input text $t$ and the desired output text $t'$; she does not need to annotate $t$ for indicating which of its portions have to be modified. The regular expression produced in the first phase is required internally in the next phases and is not visible to the user. In the second phase, the tool builds the replacement expression by identifying the subregions to be modified and using references to those subregions appropriately, i.e., according to the provided examples. In the third and final phase, the tool executes a further GP search for generating the search pattern to be used along with the replacement expression generated at the previous step.

We evaluated our proposal on 4 search-and-replace tasks executed on different real-world datasets, each including several hundreds of manually-labelled examples. The tasks consist of anonymization of usernames in tweets, partial anonymization of IP addresses, format change of dates and phone numbers. The experimental evaluation shows that our tool is indeed able to define an effective search-and-replace task with only few tens of examples. Even leaving aside the potential of our tool for non technically savvy users, this result also suggests that evolutionary computing, coupled with the power of modern computing resources, may increasingly become a surrogate for some specific technical expertise of human specialists.

## 5.2 Related works

Automatic generation of regular expressions from examples of the desired behavior may be useful in a variety of problems. We categorize these problems in increasing order of difficulty, as follows. (1) The *flagging* problem consists in assigning a binary label to a text: true, if some region of the text matches the regular expression, false otherwise. The examples consist of text regions, each accompanied by the respective desired label. (2) The *text extraction* problem consists in extracting from a text each region which matches the regular expression. The examples consist of texts, each accompanied by the annotation of the region to be extracted or the indication that nothing has to be extracted. (3) The *search-and-replace* problem is a generalization of the text extraction problem: the extracted regions have to be modified according to the specification encoded into an additional replacement expression. The examples consist of texts, each accompanied by

the corresponding modified text or the indication that nothing has to be modified.

To the best of our knowledge, no method for automatic generation of expressions suitable for search-and-replace has been proposed before. Accordingly, we briefly review in this section: (i) approaches suitable for the flagging problem or the text extraction problem; (ii) proposals not based on regular expressions for facilitating users while performing search-and-replace tasks.

The authors of [22] propose an evolutionary approach for generating regular expressions for the flagging problem. Their approach is based on grammatical evolution and individuals are specified in the Backus-Naur Form. The method effectiveness is assessed on the task of identifying those lines of HTML documents which contain hyperlinks.

Several earlier works addressed the text extraction problem with evolutionary approaches, such as genetic algorithms (GA) [7, 39] and genetic programming (GP) [95, 32, 93, 8]. In [7], after an initial evolution, individuals are recombined and then selected to obtain the final regular expression. In [39], the alphabet for regular expressions is chosen after a preliminary frequency analysis on the set of examples. Both proposals are evaluated on the task of URL extraction (composition and size of training and testing sets are not provided in [39]). Methods based on GP encode regular expressions as GP trees and evolve the corresponding individuals—each individual being a candidate expression—to maximize effectiveness according to some metric [95, 32, 93].

The large computing power widely available today revamped GP-based solutions, allowing them to outperform earlier proposals and solve practically relevant problems [8]. In this work we solve a more general problem than in [8]—search-and-replace, rather than mere text extraction. Our proposal generates automatically, in the first and third phase, regular expressions for text extraction. The corresponding procedures are built according to the proposal in [8] but extend that proposal in several key aspects. First, part of the initial population contains individuals generated directly from the examples, rather than randomly. Second, we generate regular expressions including capturing groups, a feature which is needed for solving the search-and-replace problem (we describe this feature in the next section). Third, we use fitness definitions tailored to the specific requirements of search-and-replace. In the first phase, the evolutionary search promotes individuals describing a suitable context across all the examples, which is usually larger than the region to be extracted and modified. In the third phase, the fitness of an individual depends on the behavior of that individual when coupled with the replacement expression found in the second phase.

The scenario considered in [106] concerns criminal justice information systems and the goal consists in minimizing human effort for data mining. The proposed approach starts from a single example and produces a reduced form of regular expression exploiting the operator interventions during the learning process, which is hence not fully automatic. A similar scheme for regular expression generation which involves the human operator is presented in [46]: here an active learning algorithm is proposed which starts from a single example and then requires an external operator to respond to membership queries about candidate expressions. In our work, instead, we just require a set of examples and never require human involvement during the search.

Other promising proposals for the text extraction problem which are not based on evolutionary approaches have been presented in [53, 5, 19]. In [53] the user is required to provide a set of examples and an initial regular expression: the algorithm then applies successive transformations until it reaches a local optimum in terms of precision and recall. The system proposed in [5] works similarly but the authors focus on noisy data. The method proposed in [19] does not rely on an initial regular expression: instead, it identifies relevant patterns in the set of examples and then combines the most promising pattern into a single regular expression. The proposal is evaluated on several business-related text extraction tasks, e.g., phone numbers and invoice numbers.

Concerning proposals for facilitating users in automated text editing that are not based on regular expressions, the authors of [69] propose a system (LAPIS) based on a pattern language. This pattern language, previously proposed by the same authors [71], can replace regular expressions in many common tasks, including simple forms of search-and-replace. Since some skill is still required to use the language, LAPIS offers an assisted mode in which an initial pattern is inferred from a set of positive and negative examples. Differently from our work, the assisted mode addresses only the search portion of the search-and-replace task. Furthermore, the results produced by our tool are not bound to a specific text processing system but can be used in a wealth of different environments. A similar scheme for inferring a pattern from examples is used in [70]: the goal here is to guess multiple selections for simultaneous editing.

A method for assisting the user in executing a search-and-replace tasks is proposed in [68]. The authors consider a scenario where, in order to mitigate the difficulty of defining a search pattern—which is the point we address in this work—users often work with less precise patterns and then manual check each suggested match. They propose to cluster the suggested matches so as to reduce the number of manual checks, since the user approve or reject the whole cluster instead of inspecting each single match.

## 5.3 Our approach

The user provides a set of examples $T$. Each example is composed by a pair of strings $\langle t, t' \rangle$, where $t$ is a string to be modified in $t'$. An example in which $t' = t$ is called a *negative* example.

The output of our system is a pair of strings $\langle s, r \rangle$, where $s$ is the regular expression which defines the search pattern and $r$ is the replacement expression.

The regular expression $s$ may contain zero or more *capturing groups*. A capturing group is a substring of $s$ enclosed between round parenthesis. A capturing group is itself a regular expression: when a string $t$ matches a regular expression $s$ containing a capturing group, a substring of $t$ matches the capturing group. The substring matched by a capturing group can be referenced in the replacement expression. The corresponding syntax is `$`$n$[1], where $n$ is the index of occurrence of the capturing group in the regular expression—e.g., `$1` indicates the substring matched by the first capturing group. For

[1]Some regular expression engines (e.g., Python, .NET, Ruby) use the $\backslash n$ notation, instead of the `$`$n$ notation used, e.g., in Java, JavaScript, PHP, ...)

example, suppose the user needs to change the date format from month-date-year to day-month-year: a suitable search pattern, which includes three capturing groups, is the regular expression `(\d+)-(\d+)-(\d+)` and the corresponding replacement expression is `$2-$1-$3`.

Solving the search-and-replace problem by means of regular expressions requires a notion of *context*. In general, a single pattern identifying only the substrings to be replaced might not exist. That is, it might not exist a single regular expression which, for each example $\langle t, t' \rangle$, exactly matches the shortest substring of $t$ including the characters which have to be modified in order to obtain $t'$. In practice, however, a pattern may often be found for superstrings of the strings to be replaced: we call these superstrings the *context* and this pattern the *context pattern*.

To clarify, the first two columns of Table 5.1 show a few examples related to three different search-and-replace tasks (the other columns show intermediate results discussed later). The third column illustrates the substring of $t$ that is to be replaced: it is apparent that, for each of the three tasks, there is not any pattern identifying these substrings. On the other hand, a context pattern as defined above does exist for each of the three tasks: column $c_k$ contains the contexts—i.e., superstrings of the substring to be replaced—extracted by the context pattern in column $s_c$.

The context thus describes portions of input that do have to be replaced and where replacements have to be confined. An essential component of our approach is that we do *not* require the user to specify the context. On the contrary, the context is discovered automatically by the system. For instance, in the third row in Table 5.1, we do not require that the user specifies that `ic` has to be replaced by `xx` only when `ic` is part of the Twitter username `@Toxic`, but not when is part of `Sick`. It is up to the system to extract that information from the examples. The user only describes the input text and the desired output.

Note that, for a given set of examples, the context pattern may not be determined univocally—e.g., in Table 5.1, each context could also start with the space character, or with multiple arbitrary characters followed by a space. A given set of examples might thus be associated with 0, 1 or more context patterns and finding those patterns is not straightforward.

Our proposal consists of three phases, described in the next sections in full detail: (i) generate a context pattern $s_c$, (ii) build the replacement expression $r$, (iii) generate the search pattern $s$ which works with $r$. We remark that phase 1 is necessary because we decided to not rely on the user for specifying the context of the desired changes. The result $s_c$ of this phase is not exposed to the user and is only an input for the next phase.

## 5.4 Implementation

### 5.4.1 Generating the context pattern

In this phase we aim at generating the context pattern $s_c$. To this end, we build from $T$ a learning set $T_c$ suitable for a text extraction problem: for each example $\langle t, t' \rangle$ in $T$ we

**Table 5.1:** Three sets of synthetic examples and corresponding intermediate and final results: ∅ indicates the empty string.

| $t_k$ | $t'_k$ | $D(t_k, t'_k)$ | $s_c$ | $c_k$ | $c'_k$ | $r_k$ | $s$ | $r$ |
|---|---|---|---|---|---|---|---|---|
| I like @GECCO13 conf | I like @GExxxx conf | CCO13 | @\w\w(\w+) | @GECCO13 | @GExxxx | $1xxxx | (@\w\w)\w+ | $1xxxx |
| RT @MaleLabTs New paper | RT @Maxxxx New paper | leLabTs | | @MaleLabTs | @Maxxxx | $1xxxx | | |
| Sick of @Toxic chatter | Sick of @Toxxxx chatter | ic | | @Toxic | @Toxxxx | $1xxx | | |
| nothing new here | nothing new here | ∅ | | ∅ | ∅ | ∅ | | |
| today is 1-23-13 | today is 23/01/13 | 1-23- | \d+-\d+-\d+ | 1-23-13 | 23/1/13 | $2/$1/$3 | (\d+)-(\d+)-(\d+) | $2/$1/$3 |
| he left on 3-13-12 | he left on 13/3/12 | 3-13- | | 3-13-12 | 13/3/12 | $2/$1/$3 | | |
| great 1-1-13 party! | great 1/1/13 party! | -1- | | 1-1-13 | 1/1/13 | $1/$2/$3 | | |
| he is Nick | he is *Nick* | Nick | [A-Z]\w+ | Nick | *Nick* | *$1* | ([A-Z]\w+) | *$1* |
| John was here | *John* was here | John | | John | *John* | *$1* | | |

construct exactly one example $\langle t, D(t,t')\rangle$ for $T_c$, where $D(t,t')$ is the substring of $t$ to be replaced and is determined as described below. Then, we run a GP search for generating a regular expression that attempts to satisfy the examples in $T_c$.

The string $D(t,t')$ is the shortest substring of $t$ which includes all characters that have to be modified in order to obtain $t'$ (see also the third column in Table 5.1). Formally, let $t^i$ be the $i$-th character in $t$ and let $\mathcal{L}(t)$ be the length of $t$. Then $D(t,t') = t^i t^{i+1} \dots t^{\mathcal{L}(t)-j}$, where: $i \geq 1$ is the lowest integer for which $t^i \neq t'^i$, and $j \geq 0$ is the lowest integer for which $t^{\mathcal{L}(t)-j} \neq t'^{\mathcal{L}(t')-j}$; if $t = t'$, $D(t,t') = \emptyset$.

For the purpose of the GP search, we partition the learning set $T_c$ in a training set $T_c^t$ and a validation set $T_c^v$. We form these two sets so as to distribute the negative examples evenly.

We then run a GP search on $T_c^t$, as follows. Every individual is a tree which represents a regular expression. The *function set* consists of (we assume the reader has some familiarity with regular expressions [35]): (i) the *concatenator*, that is a binary node that concatenates other nodes or leaves, (ii) the *character class* operators `[·]` and `[^·]`, (iii) the *capturing group* `(·)` and the *non-capturing group*[2] `(?:·)` operators, (iv) the *possessive quantifiers* (`·*+`, `·++`, `·?+`, `{·,·}+`). The *terminal set* consists of: (i) *constants*—i.e., a single character, a number or a string, (ii) *ranges*—i.e., `a-z` or `A-Z` and (iii) *character classes*, i.e., `\w` or `\d`.

We build the initial population of $P$ individuals so that half of them are generated directly from the examples, rather than randomly. If the number of examples is smaller than $\frac{P}{2}$, then we use them all and generate the remaining individuals randomly. In detail, let $P_E = \min\left(\frac{P}{2}, 2|T_c^t|\right)$, where $|T_c^t|$ is the size of $T_c^t$, and let $\langle t_k, D(t_k,t'_k)\rangle$ denote the $k$-th example in $T_c^t$. Then, the initial population consists of: $\frac{P_E}{2}$ individuals, each representing one of the strings $D(t_k,t'_k)$; $\frac{P_E}{2}$ individuals, each representing one of the strings $D(t_k,t'_k)$ where each digit is replaced with a `\d` and each alphabetical character with a `\w`; the remaining $P - P_E$ individuals are generated at random with a Ramped half-and-half method. We chose this strategy because we found, after preliminary experimentation, that it greatly speeds up the convergence toward good solutions.

The *fitness* function to be minimized during the search is the sum of the Levenshtein distances, across all the examples in the training set, between the string to be extracted and the string actually extracted by the first capturing group:

$$f_E(s_c) = \sum_{k=1}^{|T_c^t|} L(D(t_k,t'_k), E_1(t_k,s_c)) \tag{5.1}$$

where: (i) $s_c$ is the evaluated individual, (ii) $L(x,y)$ is the Levenshtein distance between strings $x$ and $y$, (iii) $E_1(t_k,s_c)$ is the substring of $t_k$ matched by the first capturing group of $s_c$—if $s_c$ does not contain a capturing group, we set $f_E(s_c) = +\infty$. We designed the fitness based on the key requirement of this phase, that is, automatic generation of a single pattern capable of extracting a superstring of the substring to be modified. For this reason, we promote individuals with at least one capturing group and such that this

[2] A non-capturing group is a group which cannot be referenced in the replacement expression.

group matches $D(t_k, t'_k)$. We did not include in the fitness any component depending on what is extracted beyond $D(t_k, t'_k)$ because, as discussed in the previous section, we have no explicit information about what the whole expression should extract, i.e., about the context.

We run $N_1$ independent GP searches, differing only for the random seed. We evaluate on the validation set $T_c^v$ each of the $N_1$ resulting regular expressions, and select as output $s_c$ of this phase the one which minimizes $f_E$.

### 5.4.2 Building the replacement expression

In this phase we aim at generating the replacement expression $r$. To this end, we generate a candidate replacement expression $r_k$ for each positive example $\langle t_k, t'_k \rangle$ in $T$, as follows.

Let $c_k$ be the substring of $t_k$ extracted by the context pattern $s_c$; let $b, e$ be the integers such that $c_k = t_k^b t_k^{b+1} \dots t_k^{\mathcal{L}(t_k)-e}$; let $c'_k$ be the substring of $t'_k$ delimited in the same way by $b$ and $e$, i.e., $c'_k = t'^b_k t'^{b+1}_k \dots t'^{\mathcal{L}(t'_k)-e}_k$ (if $b \geq \mathcal{L}(t'_k) - e$, then $c'_k = \emptyset$). We set the candidate replacement expression $r_k$ for the example $\langle t_k, t'_k \rangle$ by executing Algorithm 2, which takes as input $c'_k$ and the list of tuples $C_k$ constructed with the following steps. Intuitively, $C_k$ describes the boundaries of all the (maximal) substrings of $c_k$ which appear in $c'_k$. In detail, we (i) construct the list $C_k$ containing all tuples $\langle i, j, i', j' \rangle$ such that $c_k^i \dots c_k^j = c'^{i'}_k \dots c'^{j'}_k$; (ii) remove from $C_k$ each tuple $\langle i, j, i', j' \rangle$ for which there exists another tuple $\langle i^*, j^*, i'^*, j'^* \rangle$ such that $i \geq i^*$ and $j \leq j^*$; (iii) sort $C_k$ according to index $i$, in ascending order, and insert into each tuple an increasing integer $n$ that represents the position of the tuple in $C_k$; (iv) sort $C_k$ according to index $i'$, in ascending order. At this point, we set $r_k = \mathcal{R}(c'_k, C_k)$ where $\mathcal{R}$ is defined in Algorithm 2. As pointed out above, $C_k$ describes the boundaries of the substrings of $c_k$ which appear in $c'_k$—in brief, of the common substrings. Algorithm 2 builds $r_k$ by concatenating the substrings of $c'_k$ between two common substrings, replacing common substrings by tokens $\$n$, after an appropriate sorting—see Figure 5.1 for an example.

**Algorithm 2** Algorithm for building a candidate replacement expression $r_k$.

**function** $\mathcal{R}(c', C = \{\langle i_1, j_1, i'_1, j'_1, n_1 \rangle, \dots\})$
  $r \leftarrow \emptyset$
  $b \leftarrow 1$
  $e \leftarrow i'_1 - 1$
  **for** $h \leftarrow 1, |C| - 1$ **do**
    $r \leftarrow r c'^b \dots c'^e \$n_h$ ▷ concatenation
    $b \leftarrow j'_h + 1$
    $e \leftarrow i'_{h+1} - 1$
  **end for**
  $r \leftarrow r c'^b \dots c'^e \$n_{|C|} c'^{j'_{|C|}+1} \dots c'^{\mathcal{L}(c')}$ ▷ concatenation
  **return** $r$
**end function**

**Figure 5.1:** Example of execution of $\mathcal{R}(c'_k, C_k)$ where $C_k$ has been constructed from $c_k, c'_k$

$$
\begin{aligned}
c_k &= \texttt{07-14-1789} \\
c'_k &= \texttt{<b>14/07/1789</b>} \\
C_k = & \begin{array}{rrrrrl} i, & j, & i', & j', & n & \\ \langle 4, & 5, & 4, & 5, & 2\rangle & \texttt{14} \rightarrow \texttt{\$2} \\ \langle 1, & 2, & 7, & 8, & 1\rangle & \texttt{07} \rightarrow \texttt{\$1} \\ \langle 7, & 10, & 10, & 13, & 3\rangle & \texttt{1789} \rightarrow \texttt{\$3} \end{array} \\
r_k &= \texttt{<b>\$2/\$1/\$3</b>}
\end{aligned}
$$

Different examples might generate different replacement expressions, owing to conflicting or ambiguous examples. For instance, in Table 5.1, the third example generates a replacement expression `$1xxx` different from the one corresponding to the first and second example `$1xxxx`. The reason is the ambiguity associated with the `x` character, which is both part of the input text to be modified (`Toxic`) and of the text to be obtained (`Toxxxx`).

We select as output of this phase the replacement expression $r$ which occurs most frequently among all the examples. In case two or more candidates occur the same number of times, we choose one of them at random.

### 5.4.3 Generating the search pattern

In this phase, we aim at generating the search pattern $s$. To this end, we partition the set of examples $T$ in a training set $T^t$ and a validation set $T^v$, by distributing negative examples evenly, and execute a GP search similarly to Section 5.4.1 with a crucial difference in the fitness definition. In this case we associate with each individual, i.e., regular expression, $s$ two objective functions to be minimized:

$$f_R(s) = \sum_{k=1}^{|T^t|} L(t'_k, R(t_k, s, r)) \tag{5.2}$$

$$f_G(s) = |G_s(s) - G_r(r)| \tag{5.3}$$

where: (i) $R(t_k, s, r)$ is the string obtained by performing on $t_k$ the search-and-replace task defined by the regular expression $s$ and the replacement expression $r$ (found in the previous phase), (ii) $G_s(s)$ is the number of capturing groups defined in $s$ and (iii) $G_r(r)$ is the number of capturing groups defined in $r$. We minimize this multi-objective fitness by means of NSGA-II [31].

We run $N_2$ independent GP searches, differing only for the random seed, thereby obtaining $N_2$ search patterns $s_h$. Finally, we evaluate on the validation set $T^v$ each of the $N_2$ candidate solutions $\langle s_h, r\rangle$ for the search-and-replace task, and select as final result the one which minimizes $f_R$ and $f_S$ by means of NSGA-II.

## 5.5 Experiments

We experimentally evaluated our proposal on real-world datasets that we manually annotated for 4 search-and-replace tasks:

**Twitter anonymization** Replace each username found in a tweet corpus with `@xxxxxx`—e.g., `@GECCO2013` becomes `@xxxxxx`. The tweet corpus has been taken from [61];

**IP partial anonymization** Replace the second two digit groups of each IP address (expressed in dot-decimal notation) found in a web server log with `xxx.xxx`—e.g., `127.0.0.1` becomes `127.0.xxx.xxx`.

**Date format change** Change each date found in the web server log of the previous task from the Gregorian little-endian slash separated format to the Gregorian big-endian dash separated format—e.g., `31/Dec/2012` becomes `2012-Dec-31`.

**Phone number format change** Change each phone number found in an email collection by removing the parenthesis around the area code and adding a dash—i.e, `(555) 555-5555` becomes `555-555-5555`. The email corpus has been taken from [73] and was used by [53, 19].

The dataset consists of 1000 examples for each task, i.e., 1000 pairs $\langle t, t' \rangle$, of which 500 are negative. We executed three experiments for each task, varying the size of the set of examples.

We executed each experiment as follows: (i) we randomly split the dataset in two subsets $T$ and $T^e$; each subset is balanced, i.e., contains the same number of positive and negative examples; (ii) we generated a solution $\langle s, r \rangle$ using $T$ and evaluated the solution on $T^e$—i.e., $T$ is the learning set and $T^e$ is the testing set (during phases 1 and 3 the learning set is further split in training and validation, as discussed in the previous sections). We report results obtained with 5-fold cross-validation, i.e., we repeated the two steps above 5 times and averaged the performance indexes exhibited by the 5 solutions on the testing set of the corresponding experiment. We set the parameters for the GP searches as in Table 5.2. We chose the number of generations and the population size, which are different between phase 1 and phase 3, after some preliminary experimentation. We chose an equal number of independent searches in phase 1 and phase 3: $N_1 = N_2 = 32$.

We evaluated each generated solution $\langle s, r \rangle$ by means of two metrics. The *distance error rate* $\epsilon_d$ quantifies the percentage of characters that have not been processed correctly, i.e., it averages on $T^e$ pairs the distance between the expected string and the string actually obtained, divided by the length of the former. The *count error rate* $\epsilon_c$ quantifies the percentage of $T^e$ pairs that have not been processed correctly. In detail:

$$\epsilon_d = \frac{1}{|T^e|} \sum_{i=1}^{|T^e|} \frac{L(t'_k, R(t_k, s, r))}{\mathcal{L}(t'_i)} \tag{5.4}$$

$$\epsilon_c = \frac{1}{|T^e|} \sum_{i=1}^{|T^e|} \mathbb{1}(t'_k, R(t_k, s, r)) \tag{5.5}$$

**Table 5.2:** GP parameters

| Parameter | Settings |
|---|---|
| Population size (phase 1) | 500 |
| Population size (phase 3) | 3000 |
| Number of generations (phase 1) | 1000 |
| Number of generations (phase 3) | 200 |
| Selection | Tournament of size 7 |
| Initialization depths | 1–5 |
| Max. depth after crossover | 15 |
| Reproduction rate | 10% |
| Crossover rate | 80% |
| Mutation rate | 10% |

where $\mathbb{1}(x, y)$ is a function whose value is 1 if $x$ and $y$ are equal, 0 otherwise.

### 5.5.1 Results

The salient results are summarized in Table 5.3 and Table 5.4. Table 5.4 shows the values of $\epsilon_d$ for each repetition. In Table 5.4 each row corresponds to one experiment and reports the results in terms of $\epsilon_d$ and $\epsilon_c$, with average $\mu$ and standard deviation $\sigma$ across the 5 repetitions. We remark that the learning set $T = T^t \cup T^v$ is always a small portion of the dataset, less than 10%.

It seems fair to claim that the approach does provide very good performance. A set of 50 examples suffices to execute the "IP partial anonymization" and "Date format change" tasks without any mistake, which seems to be a remarkable result. Furthermore, 50 examples for the "Twitter anonymization" task suffice to achieve correct processing of 97%–98% of the testing set instances. Concerning the "Phone number format change" task, the percentage of testing instances processed correctly ranges between 92% and 95%, again as long as 50 or more examples are available for the learning procedure. To place this result in perspective we observe that the dataset for this task has been used in earlier works addressing automatic generation of solutions for the text extraction problem from examples [53, 19]. The cited works used training sets much bigger than ours. The results were provided in terms of F-measure and range in 85%–87% with 4100–33400 learning examples for [53] and 65%–92% with 400–52000 learning examples for [19]. Although our indexes cannot be compared directly to F-measure, which is not meaningful in our context, our ability of processing more than 92% of the testing instances correctly, even with a learning set smaller by one order of magnitude or more, seems to be a good result.

The previous results are the average performance across 5 repetitions of each experiment, where the result of each experiment is the best solution (on the learning set) across the 32 independent GP searches in phase 3. Further insights can be obtained by analyzing the performance of all the $5 \times 32 = 160$ solutions found for each task. To this end, Figure 5.3 plots the number of solutions with $\epsilon_d < 10\%$ and $\epsilon_c < 10\%$ (on the

**Table 5.3:** Experiment results for each fold

| Task | Dataset | | | Repetition ($\epsilon_d$ %) | | | | |
|---|---|---|---|---|---|---|---|---|
| | $\|T^t\|$ | $\|T^v\|$ | $\|T^e\|$ | 1 | 2 | 3 | 4 | 5 |
| Twitter anonymization | 10 | 10 | 980 | 1.7 | 0.0 | 7.7 | 10.7 | 0.0 |
| | 25 | 25 | 950 | 0.0 | 53.9 | 0.0 | 14.0 | 0.0 |
| | 50 | 50 | 900 | 0.0 | 0.0 | 14.0 | 9.5 | 0.0 |
| IP partial anonymization | 10 | 10 | 980 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | 25 | 25 | 950 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | 50 | 50 | 900 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Date format change | 10 | 10 | 980 | 32.7 | 58.5 | 58.1 | 0.0 | 0.0 |
| | 25 | 25 | 950 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| | 50 | 50 | 980 | 0.0 | 0.0 | 0.0 | 0.0 | 0.0 |
| Phone number format change | 10 | 10 | 980 | 7.2 | 4.7 | 6.7 | 6.2 | 7.1 |
| | 25 | 25 | 950 | 1.9 | 0.9 | 0.0 | 0.0 | 12.9 |
| | 50 | 50 | 900 | 12.3 | 0.0 | 0.0 | 2.8 | 0.0 |

**Table 5.4:** Experiment results

| Task | Dataset | | | Overall ($\epsilon_d$ %) | | Overall ($\epsilon_c$ %) | |
|---|---|---|---|---|---|---|---|
| | $\|T^t\|$ | $\|T^v\|$ | $\|T^e\|$ | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| Twitter anonymization | 10 | 10 | 980 | 4.0 | 4.9 | 5.5 | 10.5 |
| | 25 | 25 | 950 | 13.6 | 23.3 | 3.1 | 3.7 |
| | 50 | 50 | 900 | 4.7 | 6.6 | 2.0 | 1.6 |
| IP partial anonymization | 10 | 10 | 980 | 0.0 | 0.0 | 0.5 | 0.7 |
| | 25 | 25 | 950 | 0.0 | 0.0 | 0.0 | 0.0 |
| | 50 | 50 | 900 | 0.0 | 0.0 | 0.0 | 0.0 |
| Date format change | 10 | 10 | 980 | 29.9 | 29.2 | 60.0 | 54.8 |
| | 25 | 25 | 950 | 0.0 | 0.0 | 0.0 | 0.0 |
| | 50 | 50 | 980 | 0.0 | 0.0 | 0.0 | 0.0 |
| Phone number format change | 10 | 10 | 980 | 6.4 | 1.0 | 52.4 | 4.13 |
| | 25 | 25 | 950 | 3.2 | 5.5 | 8.2 | 10.8 |
| | 50 | 50 | 900 | 3.0 | 5.3 | 6.6 | 11.2 |

**Table 5.5:** Experiment execution times

| Task | $\lvert T^t\rvert + \lvert T^v\rvert$ | Time (min) | | | |
|---|---|---|---|---|---|
| | | 1 | 2 | 3 | Overall |
| Twitter anonymization | 20 | 0 | 0 | 1 | 1 |
| | 25 | 0 | 0 | 1 | 1 |
| | 50 | 0 | 0 | 2 | 2 |
| IP partial anonymization | 20 | 2 | 0 | 10 | 12 |
| | 25 | 7 | 0 | 26 | 33 |
| | 50 | 4 | 0 | 39 | 43 |
| Date format change | 20 | 3 | 0 | 15 | 18 |
| | 25 | 7 | 0 | 29 | 36 |
| | 50 | 13 | 0 | 65 | 78 |
| Phone number format change | 20 | 7 | 0 | 28 | 35 |
| | 25 | 17 | 0 | 63 | 80 |
| | 50 | 27 | 0 | 105 | 132 |

testing set). Each bar corresponds to an experiment repetition. It can be seen that our approach does generate a number of good solutions systematically, that is, the good performance is not the result of a single lucky individual.

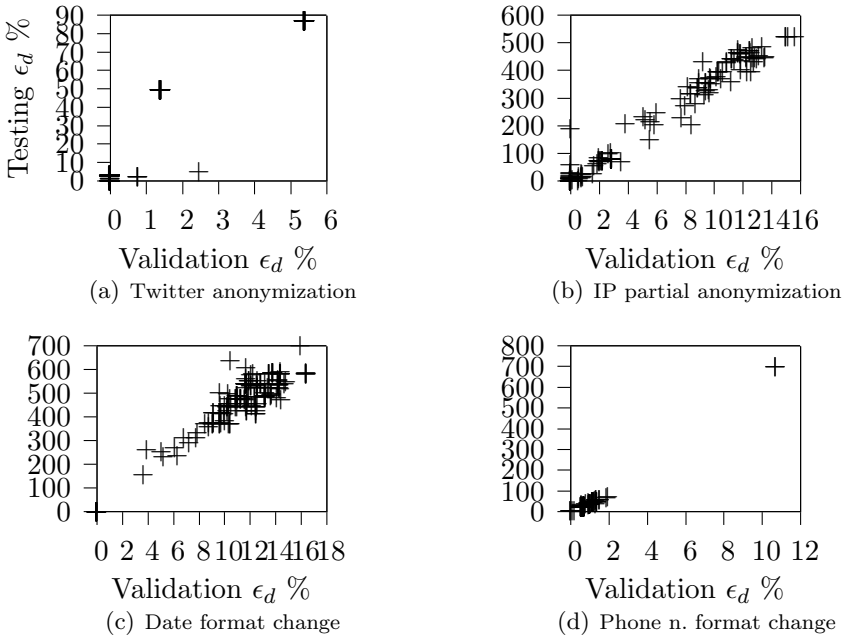
It is also interesting to point out that there is a clear correlation between performance of an individual on the validation set $T^v$ and its performance on the testing set $T^e$: Figure 5.2 shows $\epsilon_d$ on the two sets, for each of the $5 \times 32 = 160$ solutions found in our experiments (one plot per task). This outcome demonstrates that the relative performance on individuals on the validation set is a good predictor of their relative performance on the testing set.

Table 5.5 reports the average execution time for each experiment repetition, with the indication of the time taken by each of the three phases. Each experiment has been executed in parallel on 4 identical machines powered with a quad-core Intel Xeon X3323 (2.53 GHz) and 2GB of RAM. The execution times are, in most cases, too high to devise a possible interactive use of our approach. On the other hand, they seem to be sufficiently low to be practical, especially in a not far away future. Besides, the corresponding computing effort might be leased at 1 or 2 USD per hour[3]—a less accurate but much cheaper surrogate for the specific skills of a specialist.

## 5.6 Concluding remarks

We have considered the feasibility of solving a search-and-replace task described solely through examples by means of regular expressions. The motivation for this problem follows from the ever increasing wealth of unstructured or loosely structured text sources,

[3] http://aws.amazon.com/ec2/pricing/




(a) Twitter anonymization (b) IP partial anonymization

(c) Date format change (d) Phone n. format change

**Figure 5.2:** Validation $\epsilon_d$ vs. testing $\epsilon_d$.

along with the need of automated techniques for their processing.

We have presented the implementation of a tool able to generate the required search pattern and replacement expression automatically. The user merely provides examples of the input text coupled with the desired output text, without any further annotation or hints about the expected results. We are not aware of any other proposal with these features.

We assessed the performance of our tool on challenging search-and-replace tasks executed on real-world datasets. The experimental evaluation provided very good results and suggests that the approach may indeed be practically viable.

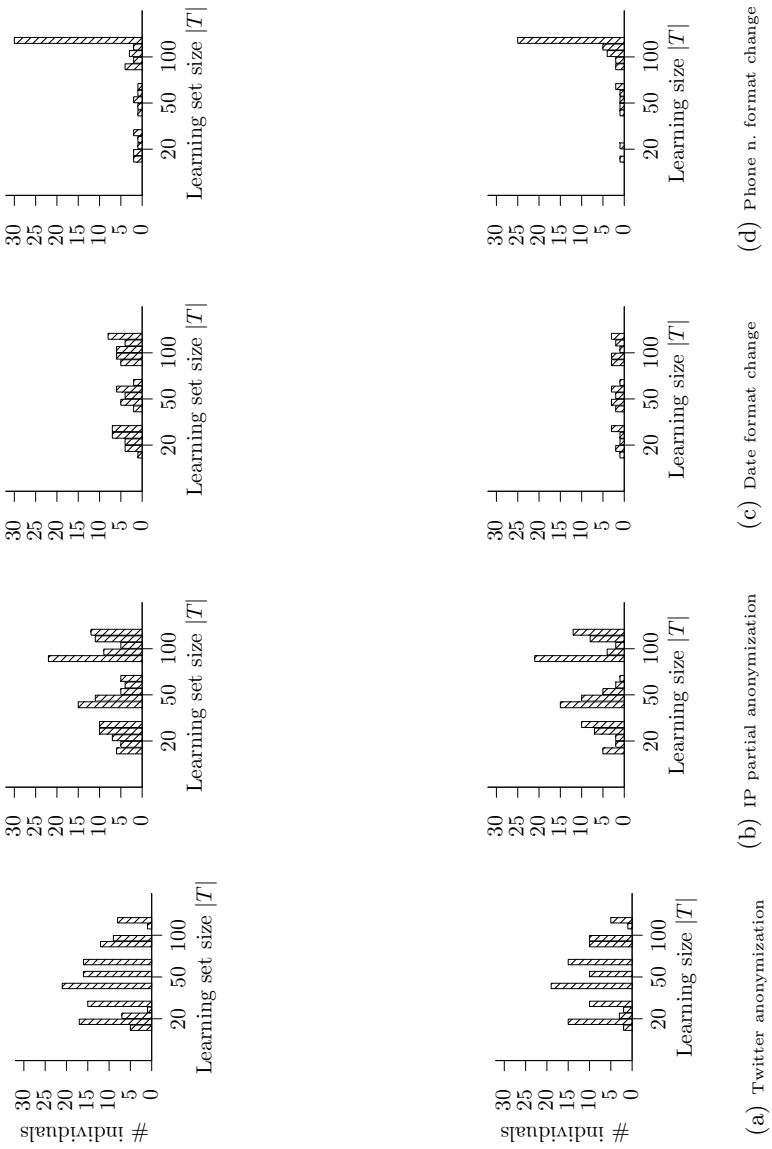



**Figure 5.3:** Number of generated solutions with $\epsilon_d \leq 10\%$ (above) and $\epsilon_c \leq 10\%$ (below). Each bar corresponds to a repetition.

Chapter 6

# Schema Generation for XML validation

## 6.1 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) [99] and XML Schema Definitions (XSD) [100]. 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 [67, 6]. 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,

[1] http://wiki.openstreetmap.org/wiki/API_v0.6/DTD, visited in November 2012

processing and consumptions of XML encoded data [34]. In this thesis work, 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.

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 [15]. 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.

## 6.2 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 [37] 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 [15] 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 automatically infers the latter based on a set of examples.

Use of spanning graphs is proposed in [74]. 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 [72], 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 [90] 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 [98] 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 [51].

Since a DTD expression is a form of regular expression, as will be discussed in section 6.3, 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., [20]) and has been studied from several points of view. For example, [33] 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 [8, 93].

Finally, the inference of XML Schemas (i.e., XSD, which are not addressed in this work) from examples is addressed in [25, 43, 72, 16].

## 6.3 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 6.1-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`, $e := a$,

```
<dblp>
  <inproceedings doi="10.1145/2330784.2331000">
    <author>Alberto Bartoli</author>
    <author>Giorgio Davanzo</author>
    <author>Andrea De Lorenzo</author>
    <author>Marco Mauri</author>
    <author>Eric Medvet</author>
    <author>Enrico Sorio</author>
    <title>
      Automatic generation of regular expressions from
      examples with genetic programming
    </title>
    <year>2012</year>
  </inproceedings>
  <article doi="10.1016/0304-3975(86)90088-5">
    <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 6.1:** An XML document (left) and a DTD (right) which validates the XML document.

$e := (e_1, \ldots, e_n)$, $e := (e_1 \mid \ldots \mid e_n)$, $e := e?$, $e := e+$, $e := e\star$, 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 $\star$ 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 6.1-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 6.1-right contains 2 attribute type declarations.

## 6.4 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

**Table 6.1:** Part of the result of pre-processing step, including the set $\tau_E$ (third column) built from the document in Figure 6.1. Character `C` is the character associated with `#text` during pre-processing.

| 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`} |

generation of the schema, (iii) post-processing of the schema which produces a syntactically correct DTD.

### 6.4.1 Pre-processing

In this step, GP-DEI parses the input XML documents and produces two sets of *training corpora*: (i) a set $\tau_E$ which will be used for generating the *element type* declarations, and (ii) a set $\tau_A$ which will be used for generating the *attribute type* declarations. The set $\tau_E$ 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 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 $\tau_A$ 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 $\tau_E$ and $\tau_A$, 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 6.1 shows the set $\tau_E$ (third column) built from the input document of Figure 6.1.

### 6.4.2 Expressions generation

Since a DTD element type declaration is a form of regular expression (Section 6.3), we generate a regular expression $R_e$ for each element name $e$ in the set $\tau_E$. We generate regular expressions with a GP-based procedure similar to the one described in [8].

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:

$$f_d(R) = \sum_{i=1}^{|E_e|} d(t_i, R(t_i)) \tag{6.1}$$

$$f_c(R) = s(R) + q(R) \tag{6.2}$$

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 6.2 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 6.1 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`.

### 6.4.3 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 $a$.

## 6.5 Experiments

### 6.5.1 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 [15];

**Genetic** a single XML document (683 MB) representing a protein sequence used in [15];

**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[2].

**RSS** a set of 10 XML documents (278 KB, globally), each containing a Rich Site Summary (RSS) files, obtained from an online scientific library[3].

The datasets are publicly available on our lab web site[4]. For ease of discussion, all the results are grouped as above.

### 6.5.2 Methodology

For each XML document in a group, we generated a DTD as follows: (i) we executed the pre-processing described in Section 6.4.1 and obtained $\tau_E$ and $\tau_A$; (ii) for each $E_e$ in $\tau_E$, we executed 8 different and independent GP evolutions (*jobs*) with the GP-related parameters set as in Table 6.2 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 $\tau_A$, we determinated which attributes are required or implied for $e$ (see Section 6.4.2) (v) we transformed each $R_e$ into a DTD element declaration through the post-processing described in Section 6.4.3; (vi) we obtained the DTD attribute list declarations through the post-processing described in Section 6.4.3.

A *validation task* consists in applying a standard XML validator[5] 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.

[2] http://download.gfoss.it/osm/osm/

[3] http://ieeexplore.ieee.org/

[4] http://machinelearning.inginf.units.it/data-and-tools

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

**Table 6.2:** GP parameters

| 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 6.3:** DTD generation results

| Name | # docs | Elements | | Preprocess (s) | | Generation (s) | | Errors | |
|---|---|---|---|---|---|---|---|---|---|
| | | avg | sd | avg | sd | avg | sd | avg | sd |
| Mondial | 1 | 22423 | - | 1.0 | - | 241.0 | - | 0.0 | - |
| Genetic | 1 | 21305818 | - | 1078.0 | - | 897.0 | - | 0.0 | - |
| SVG | 10 | 290 | 335 | 0.1 | 0.0 | 6.8 | 6.3 | 0.0 | 0.0 |
| OpenStreetMap | 20 | 7343984 | 7758557 | 144.9 | 151.6 | 6.7 | 0.5 | 0.0 | 0.0 |
| RSS | 10 | 244 | 66 | 0.1 | 0.0 | 17.7 | 2.0 | 0.0 | 0.0 |

### 6.5.3 Results

The salient results are summarized in Table 6.3. 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 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

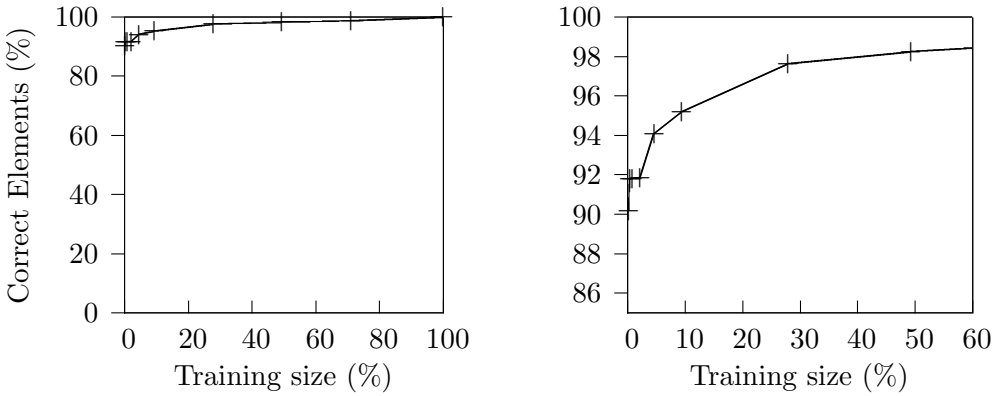



**Figure 6.2:** Generalization ability for SVG task (detail in the rightmost figure).

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 6.2 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 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 $D_{\text{SVG}}$ defined in the SVG specification[6]. 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 $\langle d'_i, D_{\text{SVG}} \rangle$

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

**Table 6.4:** Performances in presence of corrupted documents.

| 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 6.5:** Comparison between a portion of $D_{\text{RSS}}$ and one DTD generated by GP-DEI.

| 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 | <!ELEMENT channel ((title,link)\| |
| \| language \| item+ \| rating? \| image? | (((day,item+)\|(month\|year)) |
| \| textinput? \| copyright? \| pubDate? \| lastBuildDate? | \|(year\|description)))+ |
| \| docs? \| managingEditor? \| webMaster? \| skipHours? | |
| \| skipDays?)*> | |
| <!ELEMENT title (#PCDATA)> | <!ELEMENT title (#PCDATA)> |
| <!ELEMENT description (#PCDATA) > | <!ELEMENT description (#PCDATA) > |
| <!ELEMENT link (#PCDATA) > | <!ELEMENT link (#PCDATA) > |

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 $D_{\text{SVG}}$ and the number of elements which are not validated by $D_{\text{SVG}}$. The latter (FNR) is the ratio between the number of elements which are not validated by $D_i$ and are validated by $D_{\text{SVG}}$ and the number of elements which are validated by $D_{\text{SVG}}$. Table 6.4 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 $d'_1, \ldots, d'_{10}$.

Finally, in Table 6.5 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 $D_{\text{RSS}}$ found in the wild[7]. It is interesting to note that, as pointed out in Section 6.1, (i) $D_{\text{RSS}}$ 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 $D_{\text{RSS}}$.

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

## 6.6 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 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 7

# Hybrd approach for electricity prices forecasting

## 7.1 Overview

The electric power industry has shifted from a centralized structure to a distributed and competitive one. In many countries of the world, electricity markets of several forms have been established that allow consumers to select among different providers according to reliability and cost metrics. Although the legal and technical features of such markets are regulated differently in each country, the presence of *auctions* in which buyers and sellers submit their bids in terms of prices and quantities is commonplace [89].

An important form of such auctions can be found in the *day-ahead* market, in which producers and consumers present price-sensitive supply offers and demands for each hour of the next day. Each day a coordinating authority determines the outcome of the auction in terms of electricity flows and final prices. The economic relevance of these auctions makes the ability to accurately predict next-day electricity prices very important in practice, for both producers and consumers: bidding strategies are based on price forecast information hence the actual benefit obviously depends heavily on the accuracy of such information. Not surprisingly, thus, many approaches to electricity price forecasting have been explored in the recent years [105, 21, 4, 3, 79, 75, 47, 28].

In this work we examine the usefulness of Genetic Programming (GP) in the context of day-ahead electricity price forecasting. We propose two GP approaches that differ in the choice of the variables used as input of the evolutionary search and two hybrid approaches. The hybrid approaches augment the GP-built predictor with a classifier that predicts the interval to which the next prices will belong. When the predicted interval was rare in the learning set, the output of the GP-based predictor is replaced by the mean value that was observed, in the learning set, for the predicted interval. The rationale for this design is that the GP-generated predictor can hardly provide acceptable performance in regions of the input space that have been rarely seen during the learning. The classifier attempts to determine whether the system is shifting toward

the difficult-to-learn region, in which case we simply predict a constant value tailored to the specific subregion expected.

We assess our results by comparing them to a very challenging baseline that, in our opinion, may be considered as representative of the state-of-the-art for the problem. The dataset consists of hourly market clearing prices set by the California Power Exchange (CalPX) from July 5, 1999 to June 11, 2000. This period includes an important market crisis, which started on May 1, 2000, that provoked significant volatility and large variations in prices, due to bankruptcy and strong financial problems of major players in the market [65]. The forecasting methods used as baseline are those discussed in [102], which evaluates the performance of 12 widely different approaches proposed in the literature. For each approach, 24 different models are constructed and carefully tuned, one for each hour of the next day. Each model is recalibrated every day, by shifting the end of the learning set to the current day—thereby growing the learning set every day. We also include in the comparison the results from [65], which apply 4 AI-based methods to the same dataset. Even in this case, each method is calibrated differently for each hour of the day and is recalibrated every day.

We evaluate the performance of our predictors with the same error index used in these works and obtain very interesting results. The GP-based approaches exhibit slightly worse performance than those of the traditional methods. The hybrid approaches, on the other hand, provide *better* performance. In fact, they even provide a performance better than a conceptual (not implementable) forecasting method obtained by selecting in each week of the testing set the best of all the other predictors for that week.

We remark that our results have been obtained in a scenario more challenging than the baseline: (i) we construct one single predictor, valid for every hour of each day; and (ii) we never recalibrate our predictor, i.e., we use the very same learning set used in [102, 65] at the beginning of the simulation and then we leave the predictor unmodified across the entire testing set.

We believe our contribution is relevant for several reasons. First, we provide a novel solution to a problem highly relevant in practice that compares favorably to the current state-of-the-art. Second, we extend the set of application domains in which GP may outperform, or at least compete with, traditional approaches. Third, we show a simple yet effective way to cope with a dataset that do not cover the output space uniformly.

## 7.2 Our approach

### 7.2.1 Overview

We propose two techniques for day-ahead electricity price forecasting. The first technique is entirely GP-based (Section 7.2.2), while the second one is a hybrid technique that combines the output of the GP-generated predictor with the output of a second simple predictor, to be used when the system is shifting toward regions that have been rarely seen during learning (Section 7.2.3).

We denote by $P_h$ the observed price for hour $h$ and by $\hat{P}_h$ the predicted price for that

hour.

Every day at midnight the prediction machinery generates a forecast $\hat{P}_h$ for the following 24 hours, i.e., for each $h \in \{1, \ldots, 24\}$. The variables potentially available for generating $\hat{P}_h$ are:

- $P_{h-24}, \ldots, P_{h-168}$, that represent the previously observed values for the price (e.g., $P_{h-168}$ indicates the observed price one week before the generation of the prediction).
- $H_{h-24}, H_{h-48}, H_{h-72}, H_{h-96}, H_{h-120}, H_{h-144}$ and $H_{h-168}$, that represent the maximum value observed for the price in the corresponding day (e.g., $H_{h-48}$ indicates the maximum price in the day that precedes the generation of the prediction).
- $I_{h-24}, I_{h-48}, I_{h-72}, I_{h-96}, I_{h-120}, I_{h-144}$ and $I_{h-168}$, that represent the minimum value observed for the price in the corresponding day.
- $N_h, N_{h-1}, \ldots, N_{h-168}$, a set of binary values that represent whether an hour corresponds to night-time, i.e., $N_k = 1$ if $1 \leq k \leq 5$ and $N_k = 0$ otherwise.
- $I_h, I_{h-1}, \ldots, I_{h-168}$, a set of binary values that represent whether an hour corresponds to holidays.
- An enumerated variable $h \in \{1, 2, \ldots, 24\}$ that represents the hour of the day for $\hat{P}_h$.
- An enumerated variable $d \in \{1, 2, \ldots, 7\}$ that represents the day of the week for $\hat{P}_h$ (from Sunday to Saturday).

We remark that we rely only on measured values for the variable to be predicted, i.e., we do not require any exogenous variable. Existing literature, in contrast, often assumes that the prediction machinery has some exogenous variables available, e.g., temperature, actual or forecasted load and alike. Indeed, 6 of the 12 models in [102] use load forecast as exogenous variable. We believe that our approach may be more practical, simpler to implement and less dependent on "magic" tuning numbers—e.g., if temperature were to be used, where and at which hour of the day it should be taken?

Please note that the usual pattern used in literature is to predict $\hat{P}_1, \ldots, \hat{P}_{24}$ at the same time using observed values $P_0, \ldots, P_{-168}$; that is, more recent observed values are used compared to our proposed methods.

We partition the dataset in three consecutive time intervals, as follows: the *training set*, for performing the GP-based evolutionary search; the *validation set*, for selecting the best solution amongst those found by GP; the *testing set*, for assessing the performance of the generated solution.

### 7.2.2 GP approach

The set of variables potentially available to the prediction system is clearly too large to be handled by the GP search efficiently. We consider two configurations: one, that we

call *GP-baseline*, in which the terminal set consists of the same variables used in the best-performing method of the baseline work (except for any exogenous variable, such as the load) [102]. The resulting terminal set is:

$$\{P_{h-24}, P_{h-48}, P_{h-168}, I_h, N_h, L_{h-24}\}$$

The other configuration, that we call *GP-mutualInfo*, uses a terminal set that consists of variables selected by a feature selection procedure that we describe below.

The procedure is based on the notion of *mutual information* between pairs of random variables, which is a measure of how much knowing one of these variables reduces the uncertainty about the other [80]. The procedure consists of an iterative algorithm based on the training and validation portions of the dataset, as follows. Set $S$ initially contains all the 498 variables potentially available to the prediction system. Set $S_{out}$ is initially empty and contains the selected variables to be used for the GP search.

1. Compute the mutual information $m_i$ between each variable $X_i \in S$ and the price variable $Y$.
2. For each pair of variables $X_i \in S, X_j \in S$, compute their mutual information $m_{ij}$.
3. Let $X_i \in S$ be the variable with highest $m_i$. Assign $S := S - X_i$ and $S_{out} := S_{out} + X_i$. For each variable $X_j \in S$, modify the corresponding mutual information $m_j$ as $m_j := m_j - m_{ij}$.
4. Repeat the previous step until $S_{out}$ contains a predefined number of elements.

We chose to execute this feature selection procedure for selecting 8 variables. The resulting terminal set to be submitted to GP is:

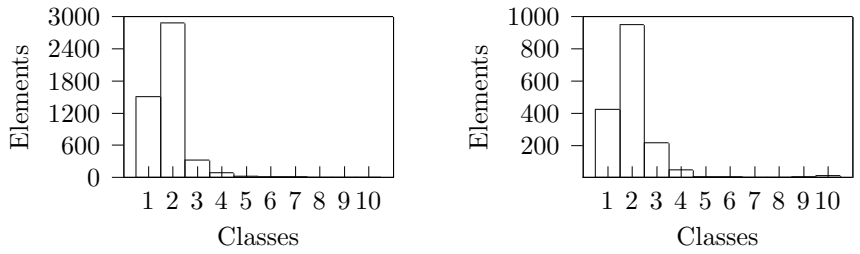
$$\{P_{h-24}, P_{h-168}, I_h, I_{h-24}, I_{h-168}, N_h, H_{h-24}, L_{h-24}, h, d\}$$

At this point we run the GP search on the training set, with parameters set as described in Section 7.3.2. Next, we compute the fitness of all individuals on the validation set. Finally, we select the individual that exhibits best performance on this set as predictor. This individual will be used along the entire testing set.

### 7.2.3 Hybrid approach

Our hybrid approach generates a GP-based predictor exactly as described in the previous section, but introduces an additional component to be used in the testing phase. This component is synthesized using the training and validation portions of the dataset, as follows.

1. We define 10 equally-sized intervals for the observed price values in the training and validation set and define each such interval to be a class.
2. We compute the mean value for each class.




(a) Training Set (b) Testing Set

**Figure 7.1:** Distribution of price values in classes.

3. We execute a feature selection procedure [42] consisting of a genetic search algorithm [38] and select 95 of the 498 variables potentially available.

4. We train a classifier for the above classes based on the variables selected at the previous step. In other words, this classifier predicts the class to which the next price value will belong. In our experiments we have used a multilayer perceptron.

The choice of the specific algorithms used at step 3 and 4 has been influenced by the software tool used for this purpose (Weka [41]).

In the testing phase, the prediction is generated as follows. We denote by $C_A$ the set of the 2 classes with more elements and by $C_B$ the set of the other classes. Let $\hat{c}$ be the predicted class for $P_i$. If $\hat{c} \in C_A$ then the predicted value $\hat{P}_i$ is the value generated by the GP predictor, otherwise $\hat{P}_i$ is the mean value computed for $\hat{c}$ (step 2 above).

The rationale of this design is that the GP-generated predictor cannot be expected to perform well in regions of the input space that have been rarely seen during the learning. The classifier attempts to determine whether the system is shifting toward the difficult-to-learn region, in which case we simply predict a constant value determined during training and tailored to the specific subregion expected.

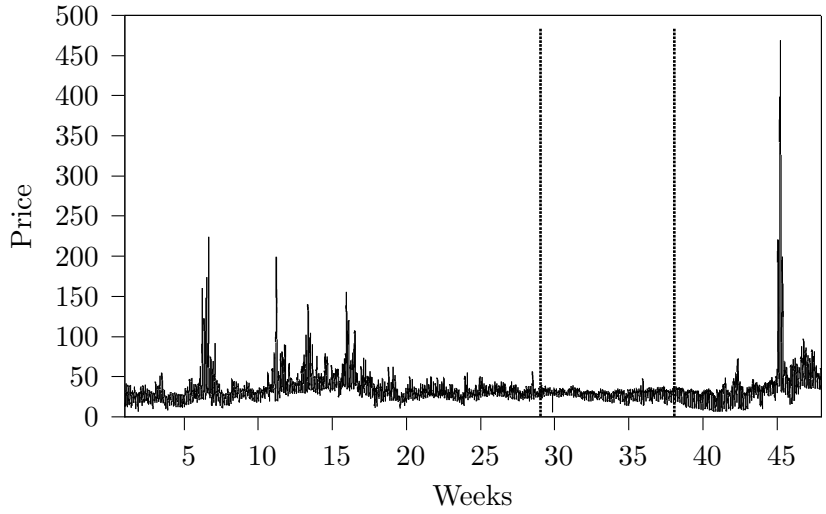
Figures 7.1(a) and 7.1(b) show the distributions of price values in the training set and in the testing set, respectively (in the validation set all values happen to belong to the first 2 classes). The percentage of elements in the 2 classes with more elements is 92% in the learning set (training and validation) and 82% in the testing set.

## 7.3 Experimental evaluation

### 7.3.1 Dataset and Baseline

As clarified in the introduction, we believe the dataset and baseline that we have used are highly challenging and may be considered as representative of the state-of-the-art. The dataset consists of hourly market clearing prices set by the California Power Exchange (CalPX) from July 5, 1999 to June 11, 2000 (Figure 7.2). This period includes a market crisis period characterized by large price volatility, that started on May 1, 2000 and lasted beyond our dataset [65].

**Figure 7.2:** Dataset used for evaluating the proposed methods. The vertical line at the right indicates the division between learning set and testing set. The vertical line at the left indicates the division between training set and validation set (used only in our approaches, see Section 7.3.2)




We use the results from [102] as main baseline. This work examines a set of widely differing approaches proposed earlier in the literature: basic autoregressive (AR), spike preprocessed (p-AR), regime switching (TAR), mean-reverting jump diffusion (MRJD), iterated Hsien-Manski estimator (IHMAR), smoothed non-parametric maximum likelihood (or SNAR) (please refer to the cited work for full details). Each approach is applied with and without load forecast as exogenous variable. For each approach, 24 different models are constructed and carefully tuned, one for each hour of the next day. In the testing phase each model is recalibrated every day, by shifting the end of the learning set to the current day—thereby growing the learning set every day. The initial learning set contains the first 9 months, from July 5, 1999, to April 2, 2000. The next 10 weeks constitute the testing set, which thus includes the market crisis mentioned above[1].

We include in the comparison also the results from [65], which applies several AI-based approaches to the same dataset[2]: autoregressive neural network (ANN), local regression (LOCAL), linear regression tree (TREE), generalized additive model (GAM) (again, please refer to the cited work for full details). This work follows the same structuring as the previous one: it uses the same learning set, it models each hour of the day separately and recalibrates each model every day.

The performance index is the Weekly-weighted Mean Absolute Error (WMAE),

[1]The cited work analyzes also another dataset from the Nordic Power Exchange (http://www.nordpoolspot.com/) augmented with hourly temperatures in Sweden. We have not yet applied our approach to this dataset.

[2]This work actually considers a longer testing set. We include here the results for the same testing set used in [102].

defined as:

$$\text{WMAE} = \frac{\sum_{h=1}^{168} |P_h - \hat{P}_h|}{\sum_{h=1}^{168} P_h}$$

where $P_h$ is the actual price for $h$ and $\hat{P}_h$ is the predicted price for that hour.

### 7.3.2 Settings

We split the dataset as in [102, 65]: the learning set contains the first 9 months, whereas the next 10 weeks constitute the *testing set*. We further split the learning data in two consecutive intervals used as described in Section 7.2.2: a *training set* from July 5, 1999, to January 30, 2000, is used for the GP search; a *validation set* from January 31, 2000 to April 2, 2000, is used for selecting the best individual produced by the GP search.

We used WMAE as fitness function. We could have used other fitness functions (e.g., squared error distances) and then assess the resulting performance on the testing set based on the performance index of interest in this work, i.e., WMAE. We have not explored this possibility in depth, but preliminary results suggest that there are no substantial differences between these two approaches. Indeed, this finding is in line with a similar assessment made in [102].

We experimented with four configurations: GP-baseline, GP-mutual Info, Hybrid-baseline (i.e., coupled with GP-baseline), Hybrid-mutual Info. The GP searches have been made with the same set of parameters, except for the composition of the terminal set, that is different for the cases GP/Hybrid-baseline and GP/Hybrid-mutualInfo (see Section 7.2.2). The functions set includes only the four basic arithmetic operators and the terminal set always includes a few basic constants: 0.1, 1, 10. During our early tests we experimented with different combinations of population size and number of generations. Concerning the former, we swept the range between 500 and 1000 individuals and found no significant differences in WMAE performance. However, we also found that the a population with 1000 individuals triplicates the computation time required by one with 500 individuals, thus we decided to select 500 as population size. Concerning the number of generations, we decided to use 1200 generations. The full set of GP-related parameters is summarized in Table 7.1.

Each GP execution consisted of 4 independent runs, executed in parallel on 4 different machines (with 500 individuals and 1200 generations each). At the end of each execution we selected the 32 individuals with best fitness on the training set, thereby obtaining a final population of 128 individuals. We evaluated the fitness of these individuals on the validation set and selected the one with best fitness as predictor to be used in the testing set.

Concerning the hybrid approach, we used the Weka tool in the standard configuration [41] and experimented with several forms of classifier: Random Tree, Random Forest Tree, SVM, Multilayer Perceptron. The latter is the one that exhibited best performance and has been used for deriving the results presented here.

Finally, a few notes about execution time: each GP search took about 34 hours on 4 identical machines running in parallel, each machine being a quad-core Intel Xeon X3323

**Table 7.1:** GP parameters

| Parameter | Settings |
|---|---|
| Populations size | 500 |
| Selection | Tournament of size 7 |
| Initialization method | Ramped half-and-half |
| Initialization depths | 1 |
| Maximum depth | 5 |
| Elitism | 1 |
| Crossover rate | 80% |
| Mutation rate | 15% |
| Number of generations | 1200 |

(2.53 GHz) with 2GB RAM; the training of the classifier took about 1 hour on a single core notebook (2 GHz), with 2GB RAM; the variable selection procedure (Section 7.2.2) took a few minutes on the same notebook.

### 7.3.3 Results

Table 7.2 presents the salient results. The first four rows contain the average WMAE along the testing set for each of the approaches that we have developed. To place these results in perspective, the next set of rows provides the same index, extracted from [102]. In particular, the first 12 rows correspond to the 6 approaches, each tested with and without predicted load as exogenous variable (models with the exogenous variable are denoted by the X suffix). Then, we provide the mean for the full set of 12 models, the mean for the 6 pure-price models only and the mean for the 6 models with exogenous variable. Finally, the row labeled Ideal gives the mean WMAE of an optimal (merely conceptual) model, constructed by selecting the best performing model in each week of the testing phase (the cited work provides the WMAE of each model in each week). The final set of rows provides the corresponding WMAE values from [65]. We excluded method LOCAL from the evaluation of mean values, as it is clearly an outlier. The row labeled Ideal has the same meaning as above whereas the row IdealBoth corresponds to selecting the best model in each week from the full set of 16 predictors provided by the cited works.

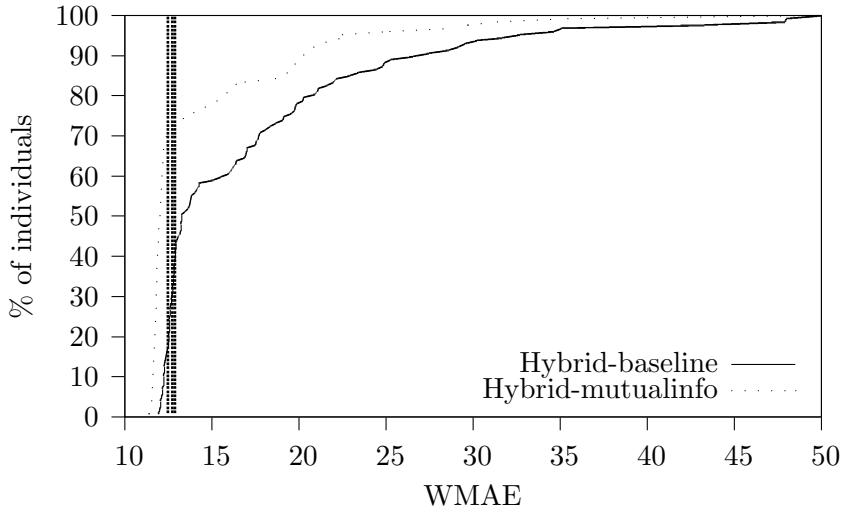
The key result is that both the hybrid methods perform better than all the other methods, including the "optimal" (and merely conceptual) predictors constructed by selecting the best predictor in each week. We believe this is a very promising result. The fact that our approaches construct one single model valid for every hour of the day and that we never recalibrate our models along the entire testing set, may only corroborate this claim.

In order to gain further insights into the ability of our hybrid methods to effectively generate accurate predictors, we evaluated WMAE across the entire testing set for all the individuals of the final population. That is, rather than selecting one single individual

**Table 7.2:** Mean WMAE results in the testing set. The first set of rows correspond to our approaches, the second set to [102], the third set to [65].

| Method | Mean WMAE (%) |
|---|---|
| GP-mutualInfo | 20.70 |
| GP-baseline | 16.17 |
| Hybrid-mutualInfo | 11.84 |
| Hybrid-baseline | 12.32 |
| AR | 13.96 |
| ARX | 13.36 |
| p-AR | 13.44 |
| p-ARX | 12.96 |
| TAR | 13.99 |
| TARX | 13.31 |
| MRJD | 15.39 |
| MRJDX | 14.67 |
| IHMAR | 14.01 |
| IHMARX | 13.37 |
| SNAR | 13.87 |
| SNARX | 13.17 |
| Mean | 13.79 |
| Mean pure-price only | 14.11 |
| Mean with load only | 13.47 |
| Ideal | 12.64 |
| ANN | 13.11 |
| LOCAL | 154499.01 |
| TREE | 14.02 |
| GAM | 13.29 |
| Mean | 13.47 |
| Ideal | 12.83 |
| Ideal both | 12.42 |

**Figure 7.3:** Distribution of mean WMAE performance for the final populations, with hybrid methods.




based on its WMAE performance in the validation set, we take all the individuals. The distribution of the respective WMAE performance is shown in Figure 7.3. Vertical lines indicate the WMAE for the three Ideal methods shown in Table 7.2.

It can be seen that the better performance exhibited by the hybrid methods is not an occasional result provoked by a single "lucky" individual: these methods consistently tend to generate individuals whose performance is better than any of the baseline methods. Indeed, the baseline performance is improved by more than half of the final population (Hybrid-baseline) and by approximately three-quarters of the final population (Hybrid-mutualInfo).

For completeness of analysis, we assessed the impact of the classifier from several points of view. Concerning the prediction mistakes performed by the classifier in the testing set, it provoked a wrong replacement of the prediction by GP 2,5% of the times, and it provoked a wrong use of the prediction by GP 10.11% of the times. The performance (mean WMAE in the testing set) that one could obtain by our Hybrid approach implemented by a perfect classifier—i.e., one that never makes any prediction mistake in the testing set—is 10.50% (mutualInfo) and 10.96% (baseline). Finally, the performance that one could obtain by always using the mean value for the class predicted by a perfect classifier is 15.49%.

From these data we observe what follows. First, attempting to improve the prediction accuracy of the classifier further is probably not worthwhile (a perfect classifier does not deliver a substantial improvement over Hybrid-mutualInfo). Second, our hybrid approach indeed boosts performance of its building blocks—classifier-based prediction and GP-based prediction: the former is slightly better than the latter, but their combination is substantially better than either of them. Third, the simple classifier-based prediction exhibits performance that is better than GP-only methods and is only slightly worse

than the 16 baseline methods.

## 7.4 Remarks

We have proposed novel GP-based methods for electricity price forecasting that are suitable for day-ahead auctions. We designed simple yet effective mechanisms for enabling GP to cope with the strong volatility that may be typical of this difficult application domain.

We assessed our proposal on a challenging real-world dataset including a period of market crisis, and compared our results against a baseline that is representative of the current state-of-the-art. Our hybrid methods performed better than all the 16 methods considered, and better than ideal (not implementable) predictors constructed by taking the best of those predictors in each week. We also showed that our methods tend to systematically generate predictors with good performance: we actually generated tens of predictors that exhibit better performance than those used as baseline.

Although our approach has certainly to be investigated further, in particular on other datasets, we believe that our results are significant and highly promising.

Chapter 8

# Estimation of tracheal pressure in mechanical ventilation control

## 8.1 Overview

High-frequency percussive ventilation (HFPV) is a non-conventional ventilatory strategy which associates the beneficial aspects of conventional mechanical ventilation (CMV) with those of high-frequency ventilation HFV [58]. HFPV acts as a rhythmic cyclic ventilation with physically servoed flow regulation, which produces a controlled staking tidal volume by pulsatile flow [58, 56]. Over the years, HFPV has proven highly useful in the treatment of several widely differing pathological conditions: closed head injury [45], patients with acute respiratory distress syndrome (ARDS) caused by burns and smoke inhalation [52, 2], newborns with hyaline membrane disease and/or ARDS [97], patients with severe gas exchange impairment [78]. The efficacy of HFPV has been demonstrated also in removing bronchial secretions under diverse conditions [30, 59].

Usage of HFPV in clinical practice involves endotracheal tubes (EET) for connecting the ventilator circuit to the airway of the patient. The pressure measured by the ventilator consists of the sum of the EET pressure drop and of the tracheal pressure dissipated to inflate lung. In order to evaluate correctly the respiratory function, in particular, to avoid barotrauma and volutrauma [85], it is mandatory to take into account precisely the real amount of pressure dissipated by these components [13, 17]. While the pressure at the ventilator end of the EET can be measured easily, measuring the tracheal pressure of a patient is more difficult and, in every day clinical practice, such a measure cannot be done invasively. For this reason, HFPV requires a model for accurately estimating the tracheal pressure value based solely on non-invasive pressure and flow measurements.

In this thesis, we describe the synthesis of such model by means of Genetic Programming (GP). We experimentally evaluated our approach by comparing our GP-generated model against two different models largely used in previous works [88, 92] on a dataset of in vitro measures. The outcomes are very important: the GP-generated models exhibit

an estimation accuracy which is sensibly higher than those of the existing models which has been crafted by human experts and have been largely used in previous works.

## 8.2 Related work

The pressure drop during mechanical ventilation $\Delta P_{\mathrm{EET}}(t) = P_{\mathrm{aw}}(t)–P_{\mathrm{tr}}(t)$—where $P_{\mathrm{aw}}(t)$ is the airway pressure measured by ventilator and $P_{\mathrm{tr}}(t)$ is the unknown tracheal pressure—has been widely studied both in adult and pediatric endotracheal tubes [13, 17, 85, 107, 27, 12, 83, 92, 88, 55]. In general, $\Delta P_{\mathrm{EET}}$ depends on flow regime, on geometric characteristics of the tube and on physical properties of the gas. The flow regime can be either laminar or turbulent with a small transitional region between these two regimes. In order to estimate $\Delta P_{\mathrm{EET}}$ and derive tracheal pressure accordingly, a model for the pressure-flow relationship characterizing EET is necessary. This model may be considered linear in the presence of laminar flow [107, 27]:

$$\Delta P_{\mathrm{EET}}(t) = R_{\mathrm{tube}}\dot{V}(t) \tag{8.1}$$

where $\dot{V}(t)$ is the flow and $R_{\mathrm{tube}}$ is the flow resistance coefficient. In most cases, though, a nonlinear pressure-flow model turns out to be more appropriate due to the presence of turbulent flow [85, 12, 83, 92, 55]:

$$\Delta P_{\mathrm{EET}}(t) = K_1\dot{V}(t) + K_2\dot{V}(t)|\dot{V}(t)| \tag{8.2}$$

where and $K_1$ and $K_2$ are the Rohrer's constants [86].

The above approaches may not be sufficiently accurate during high frequency oscillatory ventilation. Further models have been developed for such scenarios by taking into account an additional pressure drop $\Delta P_I(t)$ due to mechanical inertance $I$ and depending on the volume acceleration $\ddot{V}(t)$ [88, 92]:

$$\Delta P_{\mathrm{I}}(t) = I\ddot{V}(t) \tag{8.3}$$

## 8.3 Our approach

Our approach is based on Genetic Programming (GP). In this work, we aim at estimating the tracheal pressure $P_{\mathrm{tr}}(t)$. To this end, we use a terminal set which consists of:

- the pressure measured by ventilator $P_{\mathrm{aw}}(t)$,
- the flow $\dot{V}(t)$,
- the volume acceleration $\ddot{V}$(t),
- the percussive frequency $f_p$,
- the work pressure $P_{\mathrm{work}}$,

- random constants uniformly distributed in $[0.01, 10]$.

Note that we do not include the parameters of the lung simulator ($R_{\text{sim}}$ and $C_{\text{sim}}$, see Section 8.4.1) because their values are hardly estimable with precision during the clinical practice.

The function set is composed by the mathematical binary operators $+$, $-$, $\times$, $\div$, exp and the exponentiation pow.

We used two fitness function to be minimized: (i) the Mean Square Error (MSE) and (ii) the individual size, i.e., the total number of nodes of the individual in its tree form:

$$f_{\text{MSE}}(T) = \frac{1}{n}\sum_{i=1}^{n}(\hat{P}_T(t_n) - P_{\text{tr}}(t_n))^2 \tag{8.4}$$

$$f_{\mathcal{S}}(T) = \mathcal{S}(T) \tag{8.5}$$

where $T$ indicates a GP individual, $\hat{P}_T(t_n)$ is the value assumed by the individual $T$ at time $t_n$ and $\mathcal{S}(T)$ is the number of nodes in the individual $T$.

The choice to minimize the number of nodes is important in order to enforce a principle of parsimony, i.e., to reduce the possible proliferation of unnecessary sub-trees in individuals (also known as *bloat* [103]). We minimize the resulting *multi-objective* fitness by means of NSGA-II [31]. We performed our GP searches using a tool that we have developed in our lab [64, 9, 8]. This tool is written in Java and can run different GP searches in parallel on different machines.

## 8.4 Experiments

### 8.4.1 Experimental setup

Figure 8.1 shows the experimental setup to collect in vitro measures for this study. HFPV was provided by a volumetric diffusive respirator (VDR-4®[1]) which delivers mini-bursts of respiratory gas mixtures in the proximal airways by the Phasitron®, which is the heart of this kind of ventilation [58, 57]. In this experimental setup, pulsatile flow was delivered during inspiratory phase (In) while expiratory phase (Ex) was completely passive. The VDR-4® ventilator was set to deliver a inspiratory/expiratory (In/Ex) duration ratio of 1:1, both for the pulse and the overall respiratory cycle [58, 57].

The examined EET was connected[2] to the ventilator circuit and to a single-compartment lung simulator (ACCU LUNG)[3] that provides a physical model of the respiratory system.

Flow and pressures were acquired by a dedicated acquisition system [84]. The measurement of the flow signal $\dot{V}(t)$ was performed using Fleisch pneumotachograph[4] connected

[1] Percussionaire Corporation, USA.
[2] With a dedicated EET connector with inner diameter of 11 mm.
[3] Fluke Biomedical, USA.
[4] Type 2, Switzerland.

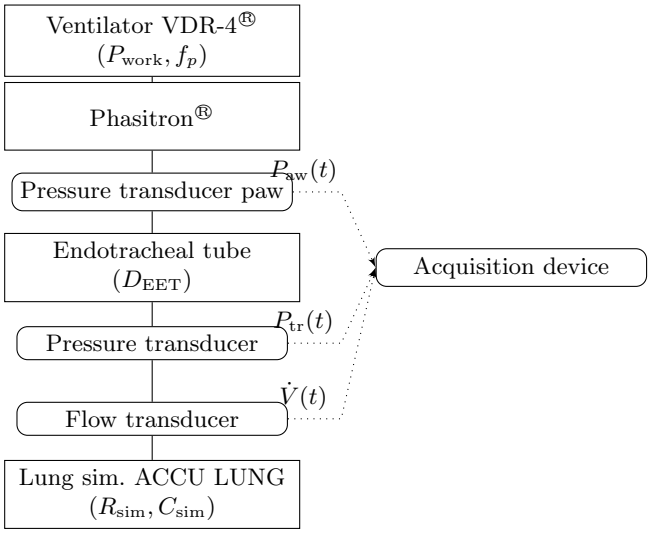



**Figure 8.1:** Diagram of the experimental setup: measurement equipment is denoted by blocks with rounded corners. Equipment parameters are shown inside corresponding blocks.

to a differential pressure transducer (0.25 INCH-D-4V[5]). The pressure signals $P_{\text{aw}}(t)$ and $P_{\text{tr}}(t)$ were measured with pressure transducers (ASCX01DN[6]) placed respectively before the EET connector and at the end of the EET, respectively. $\dot{V}(t)$, $P_{\text{aw}}(t)$ and $P_{\text{tr}}(t)$ signals were acquired at a sampling frequency of $f_s = 2000\,\text{Hz}$ with 12 bit resolution (PCI-6023E[7]). The volume acceleration $\ddot{V}(t)$ has been computed off-line as:

$$\ddot{V}(t_n) = \frac{\dot{V}(t_{n+1}) - \dot{V}(t_{n-1})}{2T_s} \tag{8.6}$$

where $T_s = \frac{1}{f_s} = 0.5\,\text{ms}$ and $\dot{V}(t_n)$ is the $n$-th measured sample of $\dot{V}(t)$.

We experimented with two different EET[8] with inner diameter of $D_{\text{EET}} = 6.5\,\text{mm}$ and $D_{\text{EET}} = 7.5\,\text{mm}$. For each EET, we considered all the 108 combinations of the following parameters:

- ventilator work pressure $P_{\text{work}}$, from $20\,\text{cm}_{\text{H}_2\text{O}}$ to $45\,\text{cm}_{\text{H}_2\text{O}}$ with increasing steps of $5\,\text{cm}_{\text{H}_2\text{O}}$;
- ventilator percussive frequency $f_p$, set to 300 cycle/min, 500 cycle/min and 700 cycle/min;
- lung simulator resistive load $R_{\text{sim}}$, set to $5\,\text{cm}_{\text{H}_2\text{O}}/(\text{L}\,\text{s})$ and $20\,\text{cm}_{\text{H}_2\text{O}}/(\text{L}\,\text{s})$;
- lung simulator compliance load $C_{\text{sim}}$, set to $10\,\text{mL/cm}_{\text{H}_2\text{O}}$, $20\,\text{mL/cm}_{\text{H}_2\text{O}}$ and $50\,\text{mL/cm}_{\text{H}_2\text{O}}$.

[5]All Sensors, USA.
[6]Honeywell, USA, with identical connectors with diameter of 20 mm.
[7]National Instruments, USA.
[8]I.D. 6.5, Rusch, Gremany and I.D. 7.5, Rusch, Gremany.

**Figure 8.2:** Measurements $P_{\text{tr}}(t)$ and $P_{\text{aw}}(t)$ for the EET with $D_{\text{EET}} = 6.5\,\text{mm}$ and the parameters combination $R_{\text{sim}} = 20\,\text{cm}_{\text{H}_2\text{O}}/(\text{L}\,\text{s})$, $C_{\text{sim}} = 10\,\text{mL}/\text{cm}_{\text{H}_2\text{O}}$, $f_p = 300\,\text{cycle/min}$ and $P_{\text{work}} = 40\,\text{cm}_{\text{H}_2\text{O}}$.

**Table 8.1:** Summary of notation

| Symbol | Type | Description | Unit |
|---|---|---|---|
| $D_{\text{EET}}$ | parameter | EET diameter | mm |
| $R_{\text{sim}}$ | parameter | lung sim. resistive load | $\text{cm}_{\text{H}_2\text{O}}/(\text{L}\,\text{s})$ |
| $C_{\text{sim}}$ | parameter | lung sim. compliance load | $\text{mL}/\text{cm}_{\text{H}_2\text{O}}$ |
| $f_p$ | parameter | HFPV percussive frequency | cycle/min |
| $P_{\text{work}}$ | parameter | HFPV work pressure | $\text{cm}_{\text{H}_2\text{O}}$ |
| $P_{\text{aw}}(t)$ | sampled signal | ventilator pressure | $\text{cm}_{\text{H}_2\text{O}}$ |
| $P_{\text{tr}}(t)$ | sampled signal | tracheal pressure | $\text{cm}_{\text{H}_2\text{O}}$ |
| $\dot{V}(t)$ | sampled signal | flow | L/s |
| $\ddot{V}(t)$ | computed signal | volume acceleration | $\text{L/s}^2$ |

In each experiment we collected the values for $\dot{V}(t)$, $P_{\text{aw}}(t)$, $P_{\text{tr}}(t)$ and $\ddot{V}(t)$ during a single inspiratory phase of a single respiratory cycle, for a duration of 4 s. Hence, for each EET and for each parameter combination, we collected a *respiratory signals set* composed of $8000 \times 4$ samples. Figure 8.2 plots $P_{\text{tr}}(t)$ and $P_{\text{aw}}(t)$ of a single respiratory signal set, i.e., only two of the four components of that signal set. Table 8.1 summarizes the symbols used in this work.

### 8.4.2 Methodology

In order to generate a model for the unknown tracheal pressure $P_{\text{tr}}(t)$ for a given EET, we proceeded as follows.

1. We randomly selected 7 respiratory signals sets and used them as training set. We randomly selected 3 other respiratory signals sets and used them as validation set. We used the remaining respiratory signal sets as testing set.

2. We executed a GP search as follows: (i) we ran 32 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 8.2; (ii) for each job and for each generation, we selected the individual with the best multi-objective fitness (according to NSGA-II) on the training set; (iii) among the resulting set of $32 \times 500 = 16000$ individuals, we selected the individual $T^*$ with the lowest $f_{\text{MSE}}$ on the validation set; (iv) we used the formula $\hat{P}_{T^*}(t)$ represented by $T^*$ as a model for $P_{\text{tr}}(t)$.

3. We repeated steps 1 and 2 five times.

**Table 8.2:** GP parameters

| Parameter | Settings |
|---|---|
| Population size | 500 |
| Number of generations | 500 |
| Selection | Tournament of size 7 |
| Initialization depths | 1–5 |
| Max. depth after crossover | 15 |
| Reproduction rate | 10% |
| Crossover rate | 80% |
| Mutation rate | 10% |

In other words, for each EET, we constructed five models for $P_{\text{tr}}(t)$ using five different learning sets.

Each GP search has been executed in parallel on 4 identical machines powered with a quad-core Intel Xeon X3323 (2.53 GHz) and 2GB of RAM.

In order to assess our results, we considered two baseline models widely used in the literature and crafted by human experts [88, 92]. The models are defined in accordance with Eq. 8.1 and Eq. 8.2 and with the addition of the volume acceleration $\ddot{V}(t)$ (Eq. 8.3), as follows:

$$\text{LM: } P_{\text{tr}}(t) = P_{\text{aw}}(t) - R_{\text{tube}}\dot{V}(t) - I\ddot{V}(t)$$
$$\text{NM: } P_{\text{tr}}(t) = P_{\text{aw}}(t) - K_1\dot{V}(t) - K_2\dot{V}(t)|\dot{V}(t)| - I\ddot{V}(t)$$

We selected the parameter values for these baseline models by means of Least Squares method applied on the same learning data available to each GP search. That is, we used a different parameter calibration for each GP search, based on the union of the training and validation sets randomly selected for that search.

The accuracy of the GP-generated models and the baseline models LM and NM was quantified by the respective mean square error MSE exhibited on the testing set.

### 8.4.3 Results

The salient results are summarized in Table 8.3. The execution time indicates the average time required for generating one of the GP-based models.

The key result is that the GP-generated model performs significantly better than the baseline in all the experiments, with the only exception of the fourth repetition for $D_{\text{EET}} = 7.5\,\text{mm}$ in which the GP-generated model performs slightly better than LM but worse than NM.

In order to gain further insights into the ability of GP to generate accurate models, we report in Figure 8.3 the cumulative distribution of the MSE on the testing set of all the best individuals selected on the validation set—i.e., at the end of each job. The baseline models are reported as vertical lines. It can be seen that the good performances

**Table 8.3:** Experiment results

| $D_{\text{EET}}$ | Repetitions (MSE %) | | | Average (MSE %) | | | Time |
|---|---|---|---|---|---|---|---|
| | GP | LM | NM | GP | LM | NM | (min) |
| 6.5 | 1.01 | 3.44 | 1.94 | 0.84 | 3.45 | 1.96 | 263 |
| | 0.78 | 3.47 | 1.92 | | | | |
| | 0.76 | 3.57 | 1.98 | | | | |
| | 0.82 | 3.48 | 1.89 | | | | |
| | 0.84 | 3.29 | 2.06 | | | | |
| 7.5 | 0.90 | 3.33 | 1.94 | 1.44 | 3.29 | 1.91 | 257 |
| | 1.06 | 3.35 | 1.95 | | | | |
| | 1.03 | 3.24 | 1.90 | | | | |
| | 3.21 | 3.27 | 1.89 | | | | |
| | 1.03 | 3.25 | 1.94 | | | | |

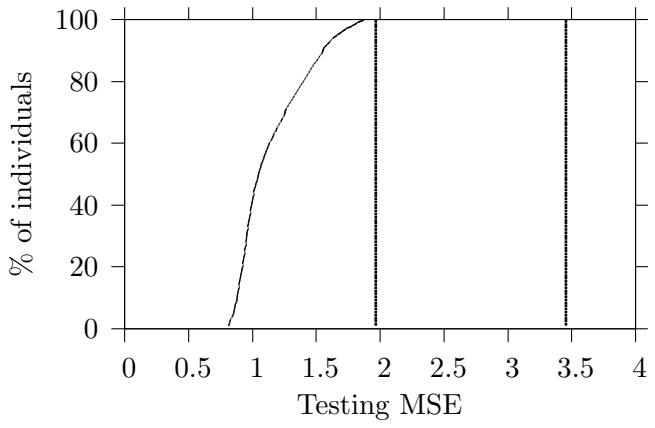
exhibited by our approach are not caused by a single lucky individual: GP is instead able to systematically produce a large set of models which perform better than the baseline models.

Figure 8.4 plots, for each of the $5 \times 32 = 160$ best individuals found at the end of each job and each repetition, the MSE on the validation vs. the MSE on the testing set. The figure shows how the performance of an individual on the validation set is a good predictor of the performance on the testing set, which hence suggests that the proposed method could be applied successfully also for different EET and parameter combinations.

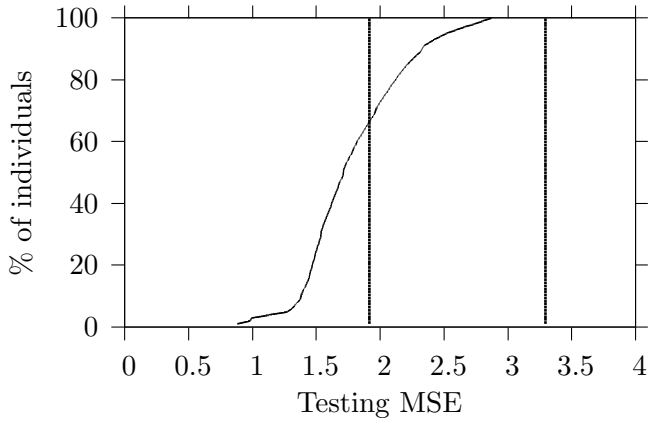
## 8.5 Remarks

We proposed a novel approach for estimating the tracheal pressure during High-frequency percussive ventilation (HFPV), a problem of uttermost importance in clinical pratice. Our approach is based on genetic programming (GP) which synthesizes a model for the tracheal pressure automatically, based on a collection of respiratory signals.

We assessed our proposal on a dataset consisting of in vitro measured respiratory signals. The results in terms of Mean Square Error are very good: the GP-generated models for the tracheal pressure perform significantly better than two other existing models—generated by human experts—largely used in earlier proposals.

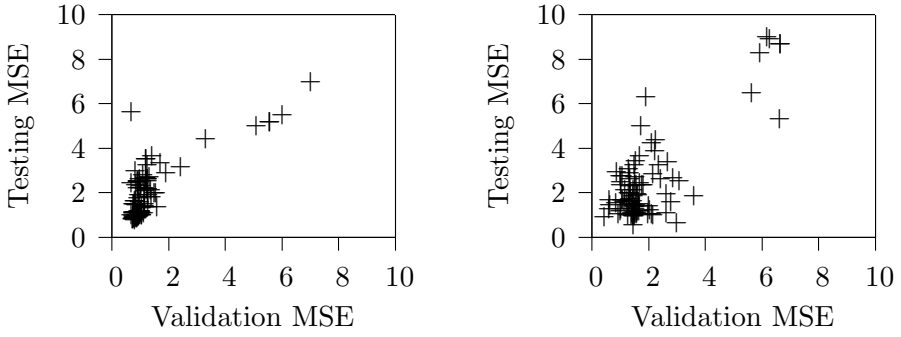



(a) $D_{\mathrm{EET}} = 6.5\,\mathrm{mm}$




(b) $D_{\mathrm{EET}} = 7.5\,\mathrm{mm}$

**Figure 8.3:** Cumulative distribution of the MSE on the testing set for the final individual of each job. Vertical lines indicate the MSE for the two baseline models NM and LM.




(a) $D_{\mathrm{EET}} = 6.5\,\mathrm{mm}$ (b) $D_{\mathrm{EET}} = 7.5\,\mathrm{mm}$

**Figure 8.4:** Validation MSE vs. testing MSE.

# Bibliography


[1] M. Ajcevic, A. De Lorenzo, A. Accardo, A. Bartoli, and E. Medvet. A novel estimation methodology for tracheal pressure in mechanical ventilation control. In *Image and Signal Processing and Analysis (ISPA), 2013 8th International Symposium on*, pages 695–699. IEEE, 2013. [cited at p. 8, 9]

[2] S. K. Alpard, J. B. Zwischenberger, W. Tao, D. J. Deyo, D. L. Traber, and A. Bidani. New clinically relevant sheep model of severe respiratory failure secondary to combined smoke inhalation/cutaneous flame burn injury. *Critical care medicine*, 28(5):1469–1476, 2000. [cited at p. 95]

[3] N. Amjady and F. Keynia. Day ahead price forecasting of electricity markets by a mixed data model and hybrid forecast method. *International Journal of Electrical Power and Energy Systems*, 30(9):533–546, 2008. [cited at p. 83]

[4] P. Areekul, T. Senjyu, H. Toyama, and A. Yona. A hybrid ARIMA and neural network model for Short-Term price forecasting in deregulated market. *Power Systems, IEEE Transactions on*, 25(1):524–530, 2010. [cited at p. 83]

[5] R. Babbar. Clustering Based Approach to Learning Regular Expressions over Large Alphabet for Noisy Unstructured. *ACM Conference on information and knowledge management CIKM 10*, pages 43–50, 2010. [cited at p. 14, 47, 58]

[6] 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. 71]

[7] 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. 13, 18, 32, 47, 57]

[8] 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. 6, 9, 32, 47, 57, 73, 75, 97]

[9] A. Bartoli, G. Davanzo, A. De Lorenzo, and E. Medvet. Gp-based electricity price forecasting. In *Genetic Programming*, pages 37–48. Springer Berlin Heidelberg, 2011. [cited at p. 8, 9, 97]

[10] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio. Automatic synthesis of regular expressions from examples. 2013. [cited at p. 6, 9, 32, 34, 37, 38, 47, 48, 50, 51, 54]

[11] M. Becchi, C. Wiseman, and P. Crowley. Evaluating regular expression matching engines on network and general purpose processors. In *ACM/IEEE Symposium on Architectures for Networking and Communications Systems*, pages 30–39. ACM, 2009. [cited at p. 11]

[12] P. Behrakis, B. Higgs, A. Baydur, W. Zin, and J. Milic-Emili. Respiratory mechanics during halothane anesthesia and anesthesia-paralysis in humans. *Journal of Applied Physiology*, 55(4):1085–1092, 1983. [cited at p. 96]

[13] A. D. Bersten, A. J. Rutten, A. E. Vedig, G. A. Skowronski, et al. Additional work of breathing imposed by endotracheal tubes, breathing circuits, and intensive care ventilators. *Critical care medicine*, 17(7):671, 1989. [cited at p. 95, 96]

[14] 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. 11, 14]

[15] 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. 11, 14, 72, 77]

[16] 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. 73]

[17] P. Bolder, T. Healy, A. Bolder, P. Beatty, and B. Kay. The extra work of breathing through adult endotracheal tubes. *Anesthesia & Analgesia*, 65(8):853–859, 1986. [cited at p. 95, 96]

[18] J. Bongard and H. Lipson. Active coevolutionary learning of deterministic finite automata. *The Journal of Machine Learning Research*, 6:1651–1678, 2005. [cited at p. 33, 47, 54]

[19] F. Brauer, R. Rieger, A. Mocan, and W. Barczynski. Enabling information extraction by inference of regular expressions from sample entities. In *ACM International Conference on Information and knowledge management*, pages 1285–1294. ACM, 2011. [cited at p. 11, 14, 19, 21, 23, 32, 47, 51, 58, 64, 65]

[20] A. Bràzma. Efficient identification of regular expressions from representative examples. In *Conference on Computational learning theory*, volume 1, pages 236–242. ACM, 1993. [cited at p. 12, 73]

[21] J. P. S. Catalao, H. M. I. Pousinho, and V. M. F. Mendes. Hybrid Wavelet-PSO-ANFIS approach for Short-Term electricity prices forecasting. *Power Systems, IEEE Transactions on*, PP(99):1–8, 2010. [cited at p. 83]

[22] A. Cetinkaya. Regular expression generation through grammatical evolution. In *International Conference on Genetic and evolutionary computation*, GECCO, pages 2643–2646, New York, NY, USA, 2007. ACM. [cited at p. 12, 13, 18, 19, 27, 28, 33, 47, 51, 57]

[23] 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. 11]

[24] K. Chen, G. Gu, J. Zhuge, J. Nazario, and X. Han. Webpatrol: automated collection and replay of web-based malware scenarios. In *ACM Symposium on Information, Computer and Communications Security*, pages 186–195. ACM, 2011. [cited at p. 11]

[25] B. Chidlovskii. Schema extraction from xml data: A grammatical inference approach. In *KRDB'01 Workshop (Knowledge Representation and Databases*, 2001. [cited at p. 73]

[26] O. Cicchello and S. C. Kremer. Inducing grammars from sparse data sets: a survey of algorithms and results. *The Journal of Machine Learning Research*, 4:603–632, 2003. [cited at p. 33, 47]

[27] G. Conti, R. De Blasi, A. Lappa, A. Ferretti, M. Antonelli, M. Bufi, and A. Gasparetto. Evaluation of respiratory system resistance in mechanically ventilated patients: the role of the endotracheal tube. *Intensive care medicine*, 20(6):421–424, 1994. [cited at p. 96]

[28] J. C. Cuaresma, J. Hlouskova, S. Kossmeier, and M. Obersteiner. Forecasting electricity spot-prices using linear univariate time-series models. *Applied Energy*, 77(1):87–106, 2004. [cited at p. 83]

[29] A. De Lorenzo, E. Medvet, and A. Bartoli. Automatic string replace by examples. In *Proceeding of the fifteenth annual conference on Genetic and evolutionary computation conference*, pages 1253–1260. ACM, 2013. [cited at p. 7, 9]

[30] K. Deakins and R. L. Chatburn. A comparison of intrapulmonary percussive ventilation and conventional chest physiotherapy for the treatment of atelectasis in the pediatric patient. *Respiratory care*, 47(10):1162–1167, 2002. [cited at p. 95]

[31] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan. A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2):182 –197, apr 2002. [cited at p. 18, 36, 50, 63, 97]

[32] B. Dunay, F. Petry, and B. Buckles. Regular language induction with genetic programming. In *Evolutionary Computation. IEEE World Congress on Computational Intelligence., IEEE Conference on*, volume 1, pages 396–400. IEEE, 1994. [cited at p. 13, 57]

[33] H. Fernau. Algorithms for learning regular expressions from positive data. *Information and Computation*, 207(4):521–541, Apr. 2009. [cited at p. 13, 73]

[34] D. Florescu. Managing semi-structured data. *Queue*, 3(8):18–24, Oct. 2005. [cited at p. 72]

[35] J. Friedl. *Mastering Regular Expressions*. O'Reilly Media, Inc., 2006. [cited at p. 15, 34, 36, 61]

[36] P. García, M. V. d. Parga, G. I. Álvarez, and J. Ruiz. Universal automata and {nfa} learning. *Theoretical Computer Science*, 407(1–3):192 – 202, 2008. [cited at p. 33]

[37] 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. 72]

[38] D. E. Goldberg. Genetic algorithms in search, optimization and machine learning, 1989. [cited at p. 87]

[39] A. González-Pardo, D. Barrero, D. Camacho, and M. R-Moreno. A case study on grammatical-based representation for regular expression evolution. In *Trends in Practical Applications of Agents and Multiagent Systems*, volume 71, pages 379–386. Springer Berlin / Heidelberg, 2010. [cited at p. 13, 18, 47, 57]

[40] S. Gulwani. Automating string processing in spreadsheets using input-output examples. In *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, POPL '11, pages 317–330, New York, NY, USA, 2011. ACM. [cited at p. 33]

[41] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explorations*, 11(1), 2009. [cited at p. 87, 89]

[42] M. A. Hall. Correlation-based feature selection for discrete and numeric class machine learning. *Proc. 17th Intern. Conf. Machine Learning*, pages 359–366, 2000. [cited at p. 87]

[43] 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. 73]

[44] T. Hruby, K. van Reeuwijk, and H. Bos. Ruler: high-speed packet matching and rewriting on npus. In *ACM/IEEE Symposium on Architecture for networking and communications systems*, pages 1–10. ACM, 2007. [cited at p. 11]

[45] J. M. HURST, R. D. BRANSON, and C. B. DEHAVEN. The role of high-frequency ventilation in post-traumatic respiratory insufficiency. *The Journal of Trauma and Acute Care Surgery*, 27(3):236–242, 1987. [cited at p. 95]

[46] E. Kinber. Learning regular expressions from representative examples and membership queries. *Grammatical Inference: Theoretical Results and Applications*, pages 94–108, 2010. [cited at p. 13, 32, 47, 57]

[47] S. J. Koopman and M. Ooms. Forecasting daily time series using periodic unobserved components time series models. *Computational Statistics & Data Analysis*, 51(2):885–903, 2006. [cited at p. 83]

[48] J. R. Koza. Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). 1992. [cited at p. 5]

[49] K. J. Lang, B. A. Pearlmutter, and R. A. Price. Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm. In *Grammatical Inference*, page 1–12. Springer, 1998. [cited at p. 47, 49, 50]

[50] W. B. Langdon, J. Rowsell, and A. P. Harrison. Creating regular expressions as mrna motifs with gp to predict human exon splitting. In *International Conference on Genetic and evolutionary computation*, GECCO, pages 1789–1790, New York, NY, USA, 2009. ACM. [cited at p. 14, 18]

[51] 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. 73]

[52] C. W. Lentz and H. Peterson. Smoke inhalation is a multilevel insult to the pulmonary system. *Current Opinion in Pulmonary Medicine*, 3(3):221–226, 1997. [cited at p. 95]

[53] 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. 13, 14, 19, 21, 23, 32, 47, 51, 58, 64, 65]

[54] 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. 19]

[55] A. Lorino, L. Beydon, C. Mariette, E. Dahan, and H. Lorino. A new correction technique for measuring respiratory impedance through an endotracheal tube. *European Respiratory Journal*, 9(5):1079–1086, 1996. [cited at p. 96]

[56] U. Lucangelo, A. Accardo, A. Bernardi, M. Ferluga, M. Borelli, V. Antonaglia, F. Riscica, and W. A. Zin. Gas distribution in a two-compartment model ventilated in high-frequency percussive and pressure-controlled modes. *Intensive care medicine*, 36(12):2125–2131, 2010. [cited at p. 95]

[57] U. Lucangelo, V. Antonaglia, W. Zin, G. Berlot, L. Fontanesi, A. Peratoner, F. Bernabè, and A. Gullo. Mechanical loads modulate tidal volume and lung washout during high-frequency percussive ventilation. *Respiratory physiology & neurobiology*, 150(1):44–51, 2006. [cited at p. 97]

[58] U. Lucangelo, V. Antonaglia, W. Zin, L. Fontanesi, A. Peratoner, F. Bird, and A. Gullo. Effects of mechanical load on flow, volume and pressure delivered by high-frequency percussive ventilation. *Respiratory physiology & neurobiology*, 142(1):81–91, 2004. [cited at p. 95, 97]

[59] U. Lucangelo, V. Antonaglia, W. A. Zin, M. Confalonieri, M. Borelli, M. Columban, S. Cassio, I. Batticci, M. Ferluga, M. Cortale, et al. High-frequency percussive ventilation improves perioperatively clinical evolution in pulmonary resection*. *Critical care medicine*, 37(5):1663–1669, 2009. [cited at p. 95]

[60] S. M. Lucas and T. J. Reynolds. Learning deterministic finite automata with a smart state labeling evolutionary algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(7):1063–1074, 2005. [cited at p. 12, 33, 48, 49, 51, 52, 54]

[61] E. Medvet and A. Bartoli. Brand-related events detection, classification and summarization on twitter. In *Web Intelligence and Intelligent Agent Technology (WI-IAT), 2012 IEEE/WIC/ACM International Conferences on*, volume 1, pages 297–302. IEEE, 2012. [cited at p. 20, 52, 64]

[62] E. Medvet, A. Bartoli, and G. Davanzo. A probabilistic approach to printed document understanding. *International Journal on Document Analysis and Recognition IJDAR*, pages 1–13–13, 2010. [cited at p. 20]

[63] E. Medvet, A. Bartoli, G. Davanzo, and A. De Lorenzo. Automatic face annotation in news images by mining the web. In *Proceedings of the 2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology-Volume 01*, pages 47–54. IEEE Computer Society, 2011. [cited at p. 9]

[64] E. Medvet, C. Fillon, and A. Bartoli. Detection of web defacements by means of genetic programming. In *Information Assurance and Security, 2007. IAS 2007. Third International Symposium on*, pages 227–234. IEEE, 2007. [cited at p. 97]

[65] E. F. Mendes, L. Oxley, and M. Reale. Some new approaches to forecasting the price of electricity: a study of californian market. http://ir.canterbury.ac.nz/handle/10092/2069. RePEc Working Paper Series: No. 05/2008. [cited at p. 84, 87, 88, 89, 90, 91]

[66] A. Menon, O. Tamuz, S. Gulwani, B. Lampson, and A. Kalai. A machine learning framework for programming by example. In *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, pages 187–95, 2013. [cited at p. 33]

[67] 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. 71]

[68] R. Miller and A. Marshall. Cluster-based find and replace. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 57–64. ACM, 2004. [cited at p. 58]

[69] R. Miller and B. Myers. Lapis: Smart editing with text structure. In *CHI'02 extended abstracts on Human factors in computing systems*, pages 496–497. ACM, 2002. [cited at p. 58]

[70] R. Miller and B. Myers. Multiple selections in smart text editing. In *Proceedings of the 7th international conference on Intelligent user interfaces*, pages 103–110. ACM, 2002. [cited at p. 58]

[71] R. Miller, B. Myers, et al. Lightweight structured text processing. In *Proceedings of 1999 USENIX Annual Technical Conference*, pages 131–144, 1999. [cited at p. 58]

[72] 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. 72, 73]

[73] E. Minkov, R. C. Wang, and W. W. Cohen. Extracting personal names from email: applying named entity recognition to informal text. In *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. 19, 64]

[74] 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. 72]

[75] T. D. Mount, Y. Ning, and X. Cai. Predicting price spikes in electricity markets using a regime-switching model with time-varying parameters. *Energy Economics*, 28(1):62–80, 2006. [cited at p. 83]

[76] P. Norvig. xkcd 1313: Regex golf. http://nbviewer.ipython.org/url/norvig.com/ipython/xkcd1313.ipynb, Jan. 2014. [cited at p. 31, 32, 38, 39]

[77] R. Paleari, L. Martignoni, E. Passerini, D. Davidson, M. Fredrikson, J. Giffin, and S. Jha. Automatic generation of remediation procedures for malware infections. In *Usenix Security Symposium*, 2010. [cited at p. 11]

[78] S. M. Paulsen, G. W. Killyon, D. J. Barillo, M. A. CROCE, and S. PAULSEN. High-frequency percussive ventilation as a salvage modality in adult respiratory distress syndrome: A preliminary study. discussion. *The American surgeon*, 68(10):852–856, 2002. [cited at p. 95]

[79] D. J. Pedregal and J. R. Trapero. Electricity prices forecasting by automatic dynamic harmonic regression models. *Energy Conversion and Management*, 48(5):1710–1719, 2007. [cited at p. 83]

[80] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(8):1226–1238, 2005. [cited at p. 86]

[81] 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. 11]

[82] P. Prasse, C. Sawade, N. Landwehr, and T. Scheffer. Learning to Identify Regular Expressions that Describe Email Campaigns. In *International Conference on Machine Learning (ICML)*, 2012. [cited at p. 14]

[83] D. J. Prezant, T. K. Aldrich, J. P. Karpel, and S. S. Park. Inspiratory flow dynamics during mechanical ventilation in patients with respiratory failure. *American Journal of Respiratory and Critical Care Medicine*, 142(6 Pt 1):1284–1287, 1990. [cited at p. 96]

[84] F. Riscica, U. Lucangelo, M. Ferluga, and A. Accardo. In vitro measurements of respiratory mechanics during hfpv using a mechanical lung model. *Physiological Measurement*, 32(6):637, 2011. [cited at p. 97]

[85] P. Rocco and W. Zin. Modelling the mechanical effects of tracheal tubes in normal subjects. *European Respiratory Journal*, 8(1):121–126, 1995. [cited at p. 95, 96]

[86] F. Rohrer. Der strömungswiderstand in den menschlichen atemwegen und der einfluss der unregelmässigen verzweigung des bronchialsystems auf den atmungsverlauf in verschiedenen lungenbezirken. *Pflügers Archiv European Journal of Physiology*, 162(5):225–299, 1915. [cited at p. 96]

[87] B. Ross. Probabilistic pattern matching and the evolution of stochastic regular expressions. *Applied Intelligence*, pages 285–300, 2000. [cited at p. 13]

[88] S. Schumann, M. Krappitz, K. Möller, R. Hentschel, G. Braun, and J. Guttmann. Pressure loss caused by pediatric endotracheal tubes during high-frequency-oscillation-ventilation. *Respiratory physiology & neurobiology*, 162(2):132–137, 2008. [cited at p. 95, 96, 100]

[89] G. B. Sheblé. *Computational auction mechanisms for restructured power industry operation*. Springer Netherlands, 1999. [cited at p. 83]

[90] 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. 72]

[91] 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. 11]

[92] M. Sullivan, J. Paliotta, and M. Saklad. Endotracheal tube as a factor in measurement of respiratory mechanics. *Journal of applied physiology*, 41(4):590–592, 1976. [cited at p. 95, 96, 100]

[93] 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 Conference*, pages 374–376. Morgan Kaufmann, 1998. [cited at p. 13, 57, 73]

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

[95] M. Tomita. Dynamic construction of finite automata from examples using hill-climbing. *Cognitive Science Conference*, pages 105–108, 1982. [cited at p. 13, 57]

[96] 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. 11]

[97] G. C. Velmahos, L. S. Chan, R. Tatevossian, E. E. Cornwell, W. R. Dougherty, J. Escudero, and D. Demetriades. High-frequency percussive ventilation improves oxygenation in patients with ards. *CHEST Journal*, 116(2):440–446, 1999. [cited at p. 95]

[98] 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. 72]

[99] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. [cited at p. 71]

[100] W3C. *W3C XML Schema Definition Language (XSD) 1.1*. [cited at p. 71]

[101] N. Walkinshaw, B. Lambeau, C. Damas, K. Bogdanov, and P. Dupont. STAMINA: a competition to encourage the development and assessment of software model inference techniques. *Empirical Software Engineering*, 18(4):791–824, Aug. 2013. [cited at p. 48, 54]

[102] R. Weron and Misiorek. Forecasting spot electricity prices: A comparison of parametric and semiparametric time series models. *International Journal of Forecasting*, pages 744–763, 2008. [cited at p. 84, 85, 86, 88, 89, 90, 91]

[103] P. A. Whigham and G. Dick. Implicitly controlling bloat in genetic programming. *Evolutionary Computation, IEEE Transactions on*, 14(2):173–190, 2010. [cited at p. 97]

[104] W. Wieczorek. Induction of non-deterministic finite automata on supercomputers. *Journal of Machine Learning Research-Proceedings Track*, 21:237–242, 2012. [cited at p. 33]

[105] L. Wu and M. Shahidehpour. A hybrid model for Day-Ahead price forecasting. *Power Systems, IEEE Transactions on*, 25(3):1519–1530, 2010. [cited at p. 83]

[106] 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. 13, 32, 47, 57]

[107] W. Zin, L. Pengelly, and J. Milic-Emili. Active impedance of respiratory system in anesthetized cats. *Journal of Applied Physiology*, 53(1):149–157, 1982. [cited at p. 96]