Variable Length-based Genetic Representation to Automatically Evolve Wrappers

David F. Barrero[1], Antonio González[2], María D. R-Moreno[1], David Camacho[2]

Abstract The Web has been the star service on the Internet, however the outsized information available and its decentralized nature has originated an intrinsic difficulty to locate, extract and compose information. An automatic approach is required to handle with this huge amount of data. In this paper we present a machine learning algorithm based on Genetic Algorithms which generates a set of complex wrappers, able to extract information from the Web. The paper presents the experimental evaluation of these wrappers over a set of basic data sets.

1 INTRODUCTION

The Web has been (and is) a success. However the explosion of contents originated in the late 90's, combined with the lack of centralized organization, has made the Web something like the "Wild West". Information is generated and stored without well defined policies and mechanisms to locate and extract it. This scenario makes locating and extracting information a hard task, thus automated solutions have to be created. In this context of unstructured information is where wrappers take an important role. Wrappers are specialized programs that automatically extract data from documents and convert the information stored into a structured format (Camacho et al., 2008). In this paper we propose the use wrappers that extract information using regular expressions (Friedl, 2002), or simply *regex*.

From a practical perspective, a regex is a string that defines a pattern, and thus it can be used to perform tasks such as pattern identification or string extraction. Regex

[1] Departamento de Automática. Universidad de Alcalá.
Ctra Madrid-Barcelona, Km. 33,6. 28871 Alcalá de Henares (Madrid), Spain.
{mdolores,david}@aut.uah.es

[2] Departamento de Informática. Universidad Autónoma de Madrid.
C/Francisco Toms y Valiente, n 11, 28049 Madrid, Spain.
{antonio.gonzalez,david.camacho}@uam.es

have been widely used by sysadmins and programmers for decades to process log files, validate user inputs or extract data from the Web. Regular expressions are a powerful tool, however they have a pronunced learning curve. A regular expression, in Formal Language Theory, is a characterization of a regular language (Brookshear, 1989) and thus a regex fully describes a DFA or NFA.

The problem of automata learning from examples is a well known problem, first described by (Gold, 1967), and is generally known as *language induction*. It is usually recognized that Evidence Driven State Merging (EDSM) (Lang, 1998) algorithms have good performance. However, in some contexts, using a regex is more natural, and its syntax can be used as a more convenient representation of the underlying automata (Petry et al., 1994).

We aim to extract information with a wrapper able to learn a regex from a set of examples. Our approach uses Genetic Algorithms (GA) (Holland, 1992; Goldberg, 1989). The algorithm is feed with two sets of strings: one containing samples of the pattern that are meant to be extracted and another one with strings that should not be extracted. The algorithm has been integrated within an multiagent-based information extraction and integration tool called Searchy (Barrero et al., 2005). We have previously used a GA with an islands model (Barrero et al., 2009) to generate regex, which lead to complex and domain dependent setups. In this paper we aim to simplify the wrapper through a Variable-Length Genetic Algorithm (VLGA) (Ramsey et al., 1998).

This article is structured as follows. The evolutionary regex wrapper is presented in the second section while some experiments carried out by the regex wrapper are shown in section 3. Then some conclusions and future work are presented.

2 EVOLUTIONARY REGULAR EXPRESSION WRAPPER

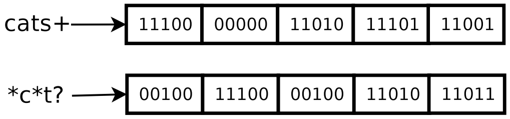
Wrapper technology allows to isolate the problem of Information Extraction in complex and distributed systems. It encapsulates the pattern, or patterns, to be handle and any kind of method, algorithm, or programming language can be used. The wrapper may use any algorithm able to extract information, for instance, GAs.

GA is a type of algorithm that is inspired in the biological process of evolution to address a wide range of Artificial Intelligence (AI) problems. Natural evolution involves a population of individuals characterized by their genetic code. These individuals are under a selective pressure where those ones that are better adapted to the environment have more chances to survive. This selective pressure, in conjunction with changes to the genetic code made by mutation and sexual reproduction, forces the evolution of the population.

A GA have to set, at least, a population of potential solutions characterized by an artificial chromosome, a mechanism to measure the quality of the solution that the chromosome represents, and a set of genetic operators. The canonical GA defines a binary chromosome of fixed length (Holland, 1992). When the complexity of the solution is not known, using VLGS might be more suitable.

The VLGA implemented in the proposed wrapper uses a binary chromosome divided in several genes of fixed length. Each gene codes a symbol from an alphabet composed by a set of valid regular expressions constructions. It is worth to point out that the alphabet is not composed by single characters but by any valid regex. These simple regular expressions are the building blocks of all the evolved regex and cannot be divided, thus, we will call them atomic regex. The position (or *locus*) of a gene determines the position of the atomic regex. Gene in position *i* is mapped in the chromosome to regex transformation as an atomic regex in the position *i*. Figure 1 represents two examples of codification. It represents how the regex *cats+* and *c*ts*?* can be coded in the GA with a gene size of 5 bits. *cats+* is able to match two strings, cat as well as *cats*, meanwhile the regex *c*ts*?* represents the set of strings whose first character is *c*, the second character is any one, the third and fourth characters are, respectively, *t* and *s* and after it can contain, or not, any character. We use classical mutation and crossover operators. Since the codification relies in a binary representation, the mutation operator is the inverse operation meanwhile the recombination is performed with a cut and splice crossover. Given two chromosomes, this crossover selects a random point in each chromosome and use it to divide it in two parts, then the parts are interchanged. Obviously, the resulting chromosomes will likely be of different lengths. Cut and splice is the genetic operator that generates chromosomes length diversity since mutation does not modify the chromosome length. Selection is done by means of a tournament with a size described in section 3. Initial population is generated randomly with chromosome lengths uniformly distributed between two values.

Fig. 1 Example of chromosome encoding.



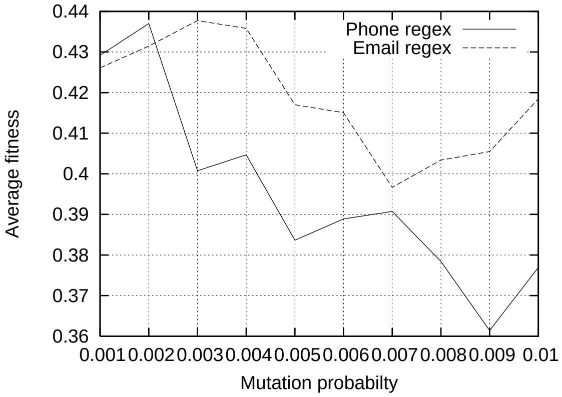
The fitness function is a key subject in the construction of a GA. In our case, for each positive example, the proportion of extracted characters is calculated. Then the fitness is calculated subtracting the average proportion of false positives in the negative example set to the average of characters correctly extracted. In this way, the maximum fitness that a chromosome can achieve is one. It happens when the regex has extracted correctly all the elements of positive examples while no element of the negative examples has been matched. An individual with a fitness value of one is called *ideal individual*.

The implementation of the evolutionary regex was done as an agent-based wrapper platform (Barrero et al., 2005). This platform provides a set mapping rules that ease the process of transform unstructured data into RDF. When an agent with the evolutionary regex wrapper is run, the wrapper first generates a valid regex for each term executing the described VLGA. Once a suitable regex is generated, the wrapper can begin to extract records from any text file accessible thought HTTP or FTP. The extraction capabilities of the described wrapper are evaluated in the next section.

3 EVALUATION

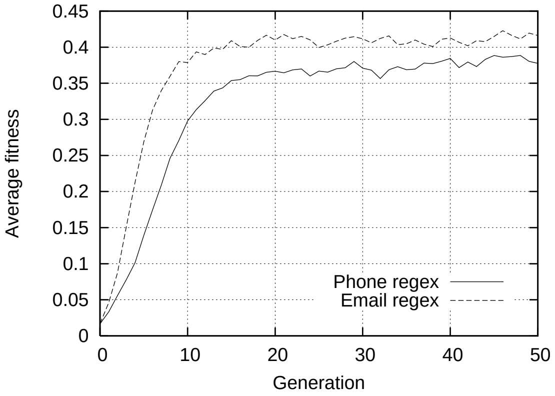
Evaluation has been divided in three phases: a first one where the GA parameters are selected; a second phase that generates the regex and a third one where the wrapper uses the evolved regex to extract data. Experiments have used two datasets to evolve two regex, one able to extract emails and another one able to extract US phone numbers. A common issue when working with GAs is parameter tunning, which is the concern of the first experimental evaluation. GAs are characterized by a large number of parameters, and, althought they are quite robust, their performance might be affected by the parameter setting. So, some initial experiments were carried out to acquire knowledge about the behavior of the regex evolution and select the GA parameters to use within the wrapper. Parameters are related in a complex non-linear way. The experiment measured, for each mayor parameter, the mean fitness in a range values of that parameter. Then, the value with higher fitness is selected. The parameters that have been set in this way are the mutation probability, population size, tournament size and elitism size.

Fig. 2 Fitness versus mutation probability.



Experiments showed that, despite the differences between the phone and email records, both have similar behaviors. In this way it is possible to extrapolate the experimental results and thus to use the same GA parameters. Setup experiments showed that a good performance is achieved with a mutation probability of 0.002 (see Fig. 2) and a tournament size of 2 individuals. A population composed by fifty individuals is a good trade-off between computational resources and convergence speed. It has been randomly generated with a chromosome size that ranges from 3 to 24 bits. Due to the stochastic nature of GAs, all experiments were run one hundred times and the measures have been averaged. Once the main parameters have been set, they can be used by the evolutionary regex wrapper. Figure 3 depicts the evolution of email and phone regex average fitness. It can be seen that the behavior of both fitness are similar, however email regex fitness converges faster, achieving a fitness value slightly higher than phone regex. The percentage of successful runs is 94% in case of phone regex, and it is increased to the 97% for email regex. Some evolved regex and their associated fitness can be seen in Table 1.

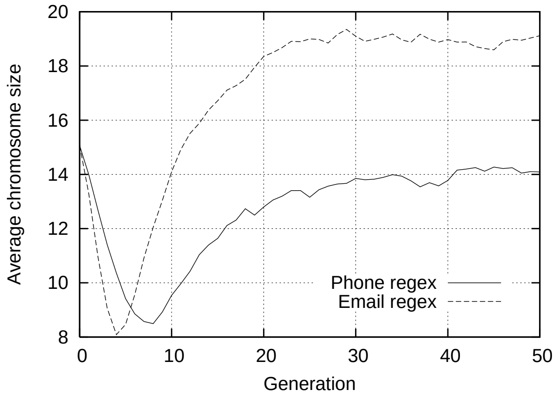
Fig. 3 Evolution of mean fitness.



The dynamics of the chromosome length can be observed in the Fig. 4, where the average chromosome length is depicted. It is clear that there is a convergence of the chromosome length and thus chromosome bloating (Chu and Rowe, 2008) is not present. This fact can be explained considering the lack of non-coding and overlapping regions in the chromosome coding, i.e., an ideal individual can only be achieved thought a certain chromosome length, so longer or shorter individuals achieve lower fitness and therefore they are discarded by the action of selective pressure. High fit individuals can be achieved only by chromosomes of a determined size, even if it is not explicit in the fitness function. This correlation between fitness and chromosome size is confirmed by a comparison between Figs. 3 and 4. Email fitness has a faster convergence and it also reaches its minimum chromosome length earlier.

Fig. 4 shows another interesting behavior. When the evolution begins, the average chromosome length tends to decrease until it reach a minimum, then it begins to increase to converge into a fixed value. It can be explained by the relationship between chromosome length and its fitness. Long chromosomes generates long regex, and thus the regex extracts more restricted patterns, decreasing the fitness associated with the regex. In early generations, individuals have not suffered evolution and thus

Fig. 4 Mean chromosome length.



its genetic code has a strong random nature. As a result long chromosomes are discarded in early stages of the evolutive process. Once the population is composed by individuals with basic regex, the recombination increases the complexity of individuals merging good chromosomes chunks, increasing the average chromosome length. The observed behavior is aligned with the literature about VLGA (Burke et al., 1998; Hutt and Warwick, 2007).

Table 1 Evolved regular expressions

Evolved regex (email)	Fitness	Evolved regex (phone)	Fitness
\w+\.-com\d+@	0	\w+	0
\w+\.	0.26	\(\d+\)	0.33
\w+@\w+\.	0.78	\(\d+\)\d+	0.58
\w+@\w+\.com	1	\(\d+\)\d+-\d+	1

When the GA has finished,the wrapper selects an ideal individual and use it to extract data. In this third experimental phase extraction capabilities are evaluated by means of the precision, recall and F-measure. Experiments use a dataset composed by six documents containing phone numbers and emails. Table 2 shows basic information about the dataset and its records. Examples have been divided in a trainning set and a testing set. Documents one, two and three are composed by the testing set. The rest of the documents are web pages retrieved from the Web without further manipulation. It should be noticed that an extracted string is computed as a correct extraction if and only if it matches exactly the record, otherwise it has been computed as a false positive.

The results, as can be seen in Table 2, are satisfactory for the synthetic documents, but precision and recall get worse for real raw documents. All the measures score one for documents one to three, meaning that the GA has generated ideal regex for the cases covered by the training set. Precision and recall in document four, that has been fetched from the Web, achieves a value of one because phone numbers in this document match perfectly the pattern used by the training set (\(*\d+\)\d+-\d+*).

Documents five and six have slightly worse results. Precision associated with the extraction of email records is limited by the pattern that the regex can extract, *\w+@\w+\.com*. Using this regex, the record *name.lastname@example.com* is extracted as *lastname@example.com* and thus it generates a false positive. In addition, emails that contains special symbols and numbers that are not matched, generating also a false positive. Precision in the extraction of phone numbers in the document six has been affected by some phone numbers that contains extensions. Since the extracted record has not been correct, it has been computed as a false positive.

Recall, on the other hand, is perfect in phone numbers extraction, however it is lower in email extraction with documents five and six. In this case the limitation is imposed by emails containing a domain with more than two levels, the evolved regex in unable to extract them, thus records such as *name@it.example.com* cannot be extracted. This limitation is related to the linear nature of the coding used in the GA.

Table 2 Extraction capacity of evolved regex. The table shows the precision (P), recall (R) and F-measure (F).

			Email regex			**Phone regex**		
	Email	Phone	**P**	**R**	**F**	**P**	**R**	**F**
Document 1	5	0	1	1	1	-	-	-
Document 2	0	5	-	-	-	1	1	1
Document 3	5	5	1	1	1	1	1	1
Document 4	0	99	-	-	-	1	1	1
Document 5	862	0	0.79	0.51	0.62	-	-	-
Document 6	88	83	0.92	1	0.96	0.8	0.8	0.8
Average			0.93	0.89	0.89	0.95	0.95	0.95

4 CONCLUSIONS AND FUTURE WORK

A method to generate regex using GAs with variable-length chromosomes has been described. It has been shown that simple regex can be evolved using two sets of examples. Experiments have shown that the chromosome length automatically converges with a minimum number of parameters. Additionally, the study cases under study follows similar patterns and optimum parameters. However, there are some remarkable limitations in this approach. The most notable limitation is the linear nature of codification in GA. Coding the hierarchical regex structure with a linear chromosome yields to unnatural mapping that might be a barrier to generate complex regex. A natural step in future research in this topic is to evolve complex regex with Grammatical Evolution.

Acknowledgements This work has been partially supported by the Spanish Ministry of Science and Innovation under the projects Castilla-La Mancha project PEII09-0266-6640, COMPUBIODIVE (TIN2007-65989), and by V-LeaF (TIN2008-02729-E/TIN).

References

Barrero, D., R-Moreno, M., López, D., and García, O. (2005). Searchy: A metasearch engine for heterogeneus sources in distributed environments. In *Proceedings of the International Conference on Dublin core and Metadata Applications*, pages 261–265, Madrid, Spain.

Barrero, D. F., Camacho, D., and R-Moreno, M. D. (2009). *Data Mining and Multiagent Integration*, chapter Automatic Web Data Extraction Based on Genetic Algorithms and Regular Expressions. Springer.

Brookshear, J. G. (1989). *Theory of computation: formal languages, automata, and complexity*. Benjamin-Cummings Publishing Co., Inc., Redwood City, CA, USA.

Burke, D. S., Jong, K. A. D., Grefenstette, J. J., Ramsey, C. L., and Wu, A. S. (1998). Putting more genetics into genetic algorithms. *Evolutionary Computation*, 6:387–410.

Camacho, D., R-Moreno, M. D., Barrero, D. F., and Akerkar, R. (2008). Semantic wrappers for semi-structured data extraction. *Computing Letters (COLE)*, 4(1):1–14.

Chu, D. and Rowe, J. E. (2008). Crossover operators to control size growth in linear GP and variable length GAs. In Wang, J., editor, *2008 IEEE World Congress on Computational Intelligence*, Hong Kong. IEEE Computational Intelligence Society, IEEE Press.

Friedl, J. E. F. (2002). *Mastering Regular Expressions*. O'Reilly & Associates, Inc., Sebastopol, CA, USA.

Gold, E. M. (1967). Language identification in the limit. *Information and Control*, 10(5):447–474.

Goldberg, D. (1989). *Genetic algorithms in search, optimization and machine learning*. Addison-Wesley.

Holland, J. H. (1992). *Adaptation in natural and artificial systems*. MIT Press, Cambridge, MA, USA.

Hutt, B. and Warwick, K. (2007). Synapsing variable-length crossover: Meaningful crossover for variable-length genomes. *IEEE Trans. Evolutionary Computation*, 11(1):118–131.

Lang, K. J. (1998). Evidence driven state merging with search.

Petry, F. E., Dunay, B. D., and Buckles, B. P. (1994). Regular language induction with genetic programming. In *International Conference on Evolutionary Computation*, pages 396–400.

Ramsey, C. L., Jong, K., Grefenstette, J., Wu, A. S., and Burke, D. S. (1998). Genome length as an evolutionary self-adaptation. In Eiben, A. E., Bäck, T., Schoenauer, M., and Schwefel, H. P., editors, *Parallel Problem Solving from Nature – PPSN V*, pages 345–353, Berlin. Springer.