UDC 519.863, DOI:10.2298/CSIS100118013B

Distributed Parameter Tuning for Genetic Algorithms

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

[1] Department of Computer Engineering. University of Alcalá.
Ctra Madrid-Barcelona, Km. 33,6.
28871 Alcalá de Henares (Madrid), Spain.
{david,mdolores}@aut.uah.es

[2] Department of Computer Science. Autonomous University of Madrid.
C/Francisco Tomás y Valiente, n 11, 28049 Madrid, Spain.
{antonio.gonzalez,david.camacho}@uam.es

Abstract. Genetic Algorithms (GA) is a family of search algorithms based on the mechanics of natural selection and biological evolution. They are able to efficiently exploit historical information in the evolution process to look for optimal solutions or approximate them for a given problem, achieving excellent performance in optimization problems that involve a large set of dependent variables. Despite the excellent results of GAs, their use may generate new problems. One of them is how to provide a good fitting in the usually large number of parameters that must be tuned to allow a good performance.

This paper describes a new platform that is able to extract the Regular Expression that matches a set of examples, using a supervised learning and agent-based framework. In order to do that, GA-based agents decompose the GA execution in a distributed sequence of operations performed by them. The platform has been applied to *Language induction* problem, for that reason the experiments are focused on the extraction of the regular expression that matches a set of examples. Finally, the paper shows the efficiency of the proposed platform (in terms of fitness value) applied to three case studies: emails, phone numbers and URLs. Moreover, it is described how the codification of the alphabet affects to the performance of the platform.

Keywords: Genetic Algorithms, parameter tuning, agents.

1. Introduction

Sometimes system administrators, programmers or the data mining community need to retrieve some strings which satisfy a given pattern, for processing logs, or detecting spam. Instead of review all the information to find those strings, users can use some pattern matching tools. One of them is called Regular Expressions, or *regex* [8]. Although they provide a powerful and flexible notation to define and retrieve patterns from text, the syntax and the grammatical rules of these regex notations are not easy to use, and even to understand.

Language Induction [10, 20, 5] is a well known problem in Machine Learning and it consists of learning a grammar from a set of samples. There are several approaches from the Formal Languages and Theoretical Computer Science perspectives [23], however it is still an open problem. A recent approach to language induction is provided by the Evolutionary Computation community, where regex are evolved by means of evolutionary algorithms like Genetic Programming (GP) [15] or Genetic Algorithms (GAs) [19].

In this paper we present a method based on GA able to generate automatically regex from a set of positive and negative samples and we propose a new chromosome codification based on messy Genetic Algorithms (mGA) [11] and crossover operators. To carry out with the experimental phase, an existing multiagent framework, named Searchy [3], is adapted to allow the implementation of ours GA-based agents. This framework has a double goal. On one hand, to reduce the execution time of the experiments and, on the other hand, to improve the search capacity in the space problem considered, allowing agents to find better solutions.

The paper is structured as follows. First, a review of Regex, Variable Length Chromosomes, mGA and MAS is provided. Sections 3 and 4 present the codification and crossover operators respectively. The agent-based framework used in the experimentation is presented in section 5. Then, they are evaluated in section 6. Finally, some conclusions and future research lines are outlined.

2. Introduction

The Intelligent Agents and Multi-Agent Systems (MAS) research fields have experimented a growing interest from different research communities like Artificial Intelligence (AI), Software Engineering, Psychology, etc... Those research fields try to solve two distinct goals.

The first goal is to define and design software programs (usually called agents) which implement several characteristics like autonomy, proactiveness, coordination, language communication, etc... This goal tries to obtain an adaptive and intelligent program which is able to provide the adequate request to the inputs received from the environment [14].

The second goal is to create societies of agents. It is possible to coordinate several of those agents to build complex societies. When considering those societies new issues arise, like social organization, cooperation, knowledge representation, coordination, or negotiation. In this situation it is possible to speak about Multi-Agent Systems and the previous problems can be studied within different perspectives [16].

One of the aims of this paper is to study the use of a MAS within a GA framework. The merger of those perspective is not a new research area. There are some work related with this idea as [22, 17, 18, 9]. Nevertheless, the approach taken on [17, 18, 9] is quite different from the one of this work. While the aim, in the cited work, is the use of a MAS that benefits from a Genetic Algorithm, in this work the roles are interchanged, and Genetic Algorithm is benefited from the MAS, to evolve regex.

2.1. A Brief Introduction to Regular Expressions

Regular Expressions can be described as a particular kind of notation for describing sets of character strings. They provide a compact and expressive notation for describing patterns of text. When a particular string is in the set described by a regex, it is often said that the regex matches the string. Most of those characters in the pattern simply match themselves in a target string, so the regular expression ”www” matches that sequence of three letters wherever it occurs in the target. However, very few characters are used in patterns as *meta-characters* to indicate repetition (i.e. +), grouping (i.e. |), or positioning (i.e. $). The powerful pattern matching facilities provided by regex in different programming languages such as Perl, PHP, JavaScript, PCRE, Python, Ruby, or Java have not been conveniently exploited by the programmers or the computer scientists due the difficulty to write and understand the syntax, as well as the semantic meaning of those regular expressions.

Following the basic wildcards from IEEE POSIX Basic Regular Expressions (BRE) standard, and POSIX Extended Regular Expressions (ERE) notation, four wildcards are considered in this work:

– **Plus +**. Repeats the previous item once or more. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is matched only once.
– **Start ***. Repeats the previous item zero or more times. Greedy, so as many items as possible will be matched before trying permutations with less matches of the preceding item, up to the point where the preceding item is not matched at all.
– **Question mark ?**: Makes the preceding item optional. Greedy, so the optional item is included in the match if possible.
– **Pipe** |: Causes the regex engine to match either the part on the left side, or the part on the right side. Can be strung together into a series of options.

An example might be useful to better understand regex. The regex *pet* only match the string ”pet”, however if the plus symbol is introduced, *pet+*, the regex matches strings ”pet” as well as ”pett” and ”petttt”. If the start symbol is used instead of the plus, the regex *pet** matches all the previous strings as well as ”pe”. If the question mark is used, the regex *pet?* only matches two strings, ”pe” and ”pet”. Finally, the regex ”pet|r” can match ”pet” and ”per”. It should be noticed that meta-characters only affect the precedent character. In case it is needed to affect a group of characters, a parenthesis should be used.

The study of extracting the regex that describes a given set of examples is not new. Some authors have used Genetic Algorithm to perform this task [7],

[12], [2], nevertheless this work is completely different due to the use of messy Genetic Algorithm with a Multi Agent platform.

The following table, Table 1, shows an example of a regex that describes a basic URL. This is a good example the regex contains all the wildcards explained in this section. Table lists some strings matched by the regex and some other which are not recognized.

Regex	(http(s)?|ftp)://(a-z)+(\.(a-z)+)*\.com
Recognize	http://myweb.com
	https://my.web.is.secure.com
	ftp://webftp.com
Not Recognize	http://myweb01.com
	ftps://.com
	ftp://WeBfTp.com
	https://my..web.com

Table 1: Example of a regex and the strings that the regex matches and the strings not recognized.

2.2. Variable Length Chromosomes and messy GA review

GAs are part of the Evolutionary Computation, a computing paradigm inspired in the biological process of evolution. It can be considered as a stochastic search algorithm that represents the solutions of a problem as chromosomes using some codification. Chromosomes explore the search space through two genetic operators: mutation and sexual reproduction (*crossover*). The metric of how well a chromosome solves a problem is given by a fitness function. The genetic material contained in the chromosome is named genotype meanwhile the realization of such genetic material is referred as phenotype.

The number of variables involved in a GA problem is closely related to the chromosome length. Sometimes it is possible to determine the number of parameters that certain problem requires, and therefore to determine the chromosome length to introduce in the GA. However, there are many problems in which such approach is not an adequate solution because it is not possible to limit the size of the solution. The number of nodes in a neural network or the size of an evolved program in GP [15] are not easy to set prior to the execution of the algorithm. Of course, it is possible to set a maximum number of variables. But this approach sets an arbitrary limitation to the complexity of the solution. A more desirable solution is to use an algorithm able to adapt the size of the chromosome dynamically. This is the goal of the Variable-Length Genomes (VLGs) [13].

The main difference between fixed-length chromosomes and VLGs is the crossover operator. The simplest crossover used in VLGs is *cut and splice*.

Given two chromosomes (likely with different lengths), this crossover operator selects a random point in each chromosome and use it to divide them in two parts, then the parts are interchanged.

An early work in VLGs is the one developed by Goldberg with the messy GA [11]. It is a variable-length, gene position independent representation. Basically, mGA decouples the position of the genes from the phenotype that they represent and thus any gene can be placed anywhere within the chromosome. It is done representing each bit with a tuple $(locus, value)$ where the position, or locus, of the bit is specified as well as its value. Common genetic operators such as bit inversion mutation and cut and splice crossover are then applied to the chromosome constructed in this way.

Codification in mGA may generate two special situations that must be handled. Since the locus is coded within the allele, it might happen that not all the genes are defined, generating a problem called *underspecification*. Original mGA handles this situation by means of templates. A second problem arises when there are several alleles coding the same gene, i.e., the *overspecification*. mGA solves overspecification by means of a first-come-first-served philosophy.

mGA defines three phases: initialization, primordial and juxtapositional phase. The initialization phase deals with the generation of the initial population that feed the GA. It is composed by the set of all chromosomes of size k, where k is the gene size. This algorithm generates an unnecessary number of individuals, so the second phase filters the individuals trying to select individuals with a high density of good building blocks. The selected individuals are used then as the initial population in the GA. The generational based evolution is done in the juxtapositional phase.

3. Chromosome codification

There are a number of questions that must be answered in order to successfully implement a GA. One of these questions is how to represent in the chromosome the problem that is addressed. In this section three codifications able to represent a regex in a binary chromosome are presented: one based on a plain VLG and two codifications inspired in mGA.

The lexical approach that we have adopted requires an alphabet Σ of *atomic regex* x_i such as $\Sigma = \{x_0, x_1, ..., x_N\}$. Σ is constructed using the positive and negative samples. Atomic regex are identified applying Lempel-Ziv law [25]. This law states that texts are not composed by a uniform distribution of tokens, instead, a few tokens appear many times while many tokens have a reduced weight in the text. We build tokens using a set of symbols to divide the samples and then, those tokens that appear more times are selected to be part of the alphabet. The second subset of Σ is composed by a fixed set of symbols. This is an automatic and domain independent method that can be used with almost any codification schema.

3.1. Plain Variable-Length Genomes

Despite the inherent difficulty to determine *a priori* the length of the regex, it is possible to imitate a VLG by means of an island model [1] with immigration of individuals. It can take benefits of a parallel algorithm implemented with agents, however the population must be increased and it imposes a maximum length for the chromosomes. In this context selecting a codification able to deal with VLG seems to be a natural solution.

A simple way to compose and evolve the set of atomic regex in Σ is the traditional VLG, a binary chromosome of arbitrary length that is recombined using cut and splice, as it was described in sec. 2.2. The correspondence between the genes and the atomic regex in Σ is done as follows. Each gene contains l_g bits that code an integer number $i < 2^{l_g}$, then the gene represents the element x_{imodN} of Σ. Initial population has chromosomes randomly created with lengths uniformly distributed between a minimum chromosome length l_{min} and a maximum length l_{max}. The rest of the paper will use the term plain VLG to mean the codification schema described in this section.

3.2. Modified messy GA and biased messy GA

Plain VLG provides a simple variable-length genome coding, however variable-length genomes usually present some problems. One of them is the tendency to bloat the chromosome length, as Chu observed [4]. Another problem is the *genetic linkage*, i.e., the tendency of some alleles to remain joint due, for example, to crossover biases [21]. A solution to genetic linkage is mGA because it decouples the position of the gene in the chromosome from its semantics. It is still a simple codification and it seems to be a good choice to study how genetic linkage affects the regex evolution in VLGs. However, some modifications are needed in order to use mGA in the context of our work.

Like original mGA does, in our proposal genes are coded as $(locus, value)$, however $value$ follows the same coding scheme as the one described in 3.1 instead of being a single bit. It represents a symbol $x_i \in \Sigma$. We have integrated the original mGA initialization and primordial phases into one phase that generates Σ from the data set using Lempel-Ziv law, following the same philosophy that the used with VLG codification. With this approach there is no need to implement the primordial phase because the building blocks are integrated in Σ.

Initial mGA and revised versions of the algorithm such as fast mGA [6], do not generate the initial population randomly. There is rather a slight control about how to initialize them, as it was described in sec. 2.2. We propose also a modification of the initialization. Given a random number l uniformly distributed between l_{min} and l_{max}, a chromosome with l/l_g genes is created. $value$ is filled with a random value that codes an atomic regex following the same mechanism than plain VLG, while $locus$ takes a value from 0 to $l/l_g - 1$.

A second modification to the mGA named biased messy GA (bmGA) is also proposed. The biased messy GA instead of initializing the loci field with positions from 0 to $l/l_g - 1$, they are initialized with a biased loci, and thus their values range from $bias$ to $l/l_g + bias - 1$, where $bias$ can take several values. However, we have used $bias = 2^{l^{locus}}/2$, i.e., alleles begin to be placed in the middle position.

4. Recombination operators

The main role of the crossover is to recombine good chunks of chromosomes generating offspring [24] with better genetic information. Some authors have argued that the crossover performs better when it recombines two similar chromosomes [13], however this point is controversial and there are not a general consensus in the GA community. In the context of regex evolution, the disruptive properties is crossover is a main issue because of the rough nature of the fitness, a very small difference in the chromosome might lead to a dramatic change in the fitness. Following these ideas it seems natural to hypothesize that using a less destructive crossover operator will increase the performance of the GA in regex evolution.

The goal of the new crossover mechanism is to use the knowledge about the codification to recombine chromosomes in a less destructive way compared with the cut and splice crossover. Crossover is not directly performed with the chromosomes, instead an intermediate table is constructed. Our crossover proposal is divided in five phases as described.

1. *Integer chromosomes construction*. Alleles in the chromosomes (including their loci and values) are transformed into an integer representation. The order in which the alleles appear is respected.
2. *Intermediate table construction*. The intermediate table is a table composed by three columns and as many rows as the sum of not underspecified genes in the chromosomes. One column contains the sorted loci while the latter two columns contain each one the values (if any) defined for such locus.
3. *Crossover*. The intermediate table can be seen as two chromosomes, and thus any traditional crossover operators (one point, two points and uniform crossover) can be applied just interchanging the values of the chromosomes columns in the table.
4. *Recombined integer chromosomes construction*. Two integer chromosomes are constructed using the recombined intermediate table, it is the inverse operation of the phase two. It should be noticed that because of the lack of genetic linkage the position of the alleles is irrelevant for the crossover and thus their position can be changed without loss of relevant information.
5. *Recombined binary chromosomes construction*. The integer chromosomes are representated with a binary codification.

An example of modified one-point crossover is shown in Fig. 1. Two chromosomes are recombined in the example. Both use seven bits to code each gene, divided in three bits for the locus and four bits for the value. Chromosome

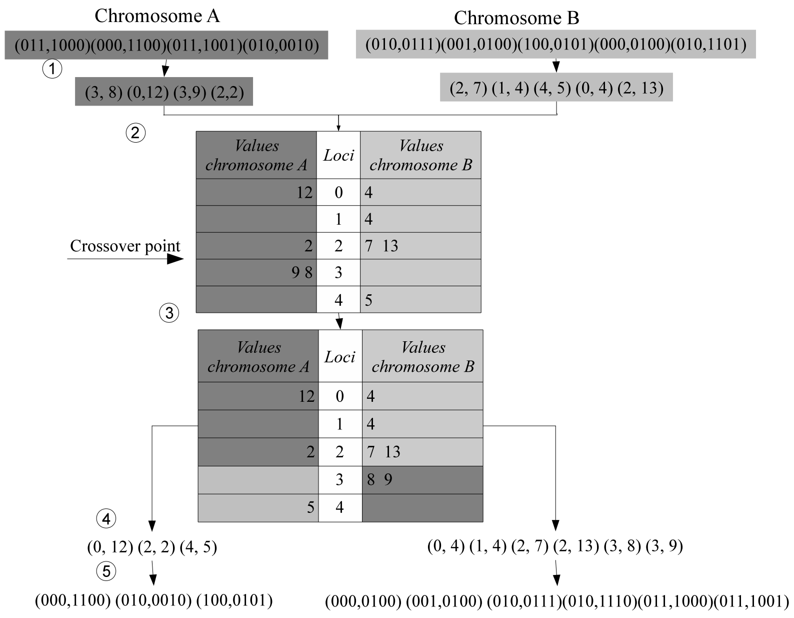


Fig. 1: Modified one point crossover example

A is composed by four alleles and chromosome B is composed by five alleles. Chromosome A and B present overspecification as well as underspecification. The intermediate table is constructed and, as it can be seen in the figure, underspecified genes correspond to empty cells. On the other hand, overspecified genes correspond to cells with several sorted values. A random point is used to interchange cells in the table, generating the recombined chromosomes. Any other traditional crossover mechanism may also be applied.

5. The GA evaluation framework

Due to the high number of GA runs that must be performed and the parallel nature of the GA, the set of experiments were run in a MAS that decomposes the GA evolution in a sequence of operations performed by different agents. This MAS has been deployed using the Searchy platform [3]. In this way the experimentation can be divided into different simple operations that are composed and executed in parallel, increasing the performance and the search capability of the algorithm.

There are six roles defined in the MAS: control, population, crossover, fitness evaluation, codification and alphabet agent. Each role is implemented using a specialized agent and there are several agents to implement crossover and codification. A description of each role is briefly presented.

1. *Control Agent*. It is responsible for the execution of the experiment, and has to fulfill some tasks, such as the initialization of Population Agents and control the execution of the experiments. It also gathers measures from the populations and generate statistics, averaging the measures of all the GA executions.
2. *Population Agent*. This agent contains a population of individuals represented by a binary chromosome. It also performs the generational evolution of the population using the services provided by the crossover, fitness evaluation and coding agents services.
3. *Crossover Agent*. A crossover agent is an agent that performs a crossover between two chromosomes. Actually, there are four different crossover agents that implement the four crossover operators under study. Cut and splice crossover can be performed in any codification under study while modified one, two and any point crossover requires a mGA or bmGA.
4. *Fitness Evaluation Agent*. This is an agent that, given a string regex is able to evaluate its extraction capabilities using a training set. It should be noticed that since it takes a string as input, this agent in not affected by the chromosome codification.
5. *Codification Agent*. The codification generates the phenotype associated to a given chromosome, i.e., it transforms a chromosome into a string that contains a regex. This regex is used by the Population Agent prior to evaluate any individual's fitness. There are two codification agents, the Plain VLG Coding Agent and the mGA Coding Agent. Since the only difference between mGA and bmGA is the initialization of the populations there is no need to use a bmGA Coding Agent.
6. *Alphabet Agent*. The alphabet agent takes as input the set of positive examples and using the Lempel-Ziv law identifies a set of tokens that are used to generate the atomic regex alphabet. The alphabet is used by the Codification Agents to generate the string regex.

Fig. 2 depicts the MAS architecture. First, the Control Agent (1) initializes several Population Agents (2) and associate each population with a Crossover Agent (3) and a Codification Agent (4). In this way each Population Agent contains an experiment involving a certain crossover operator and codification. Once the Population Agents have been initialized they evolve their populations for a number of generations, then they return to the Control Agent several measures. The Control Agent repeats this process a given number of times and then averages the measures.

The Alphabet Agent (5) reads the positive examples and generates the alphabet once, then it is provided to the Coding Agents which set a correspondence between each element in the alphabet and the codification used in the genome. It should be noticed that no agent with the exception of the Coding Agents need to know how the chromosome is coded, they manipulate the chromosome as a sequence of bits. The only agent that does not require a binary

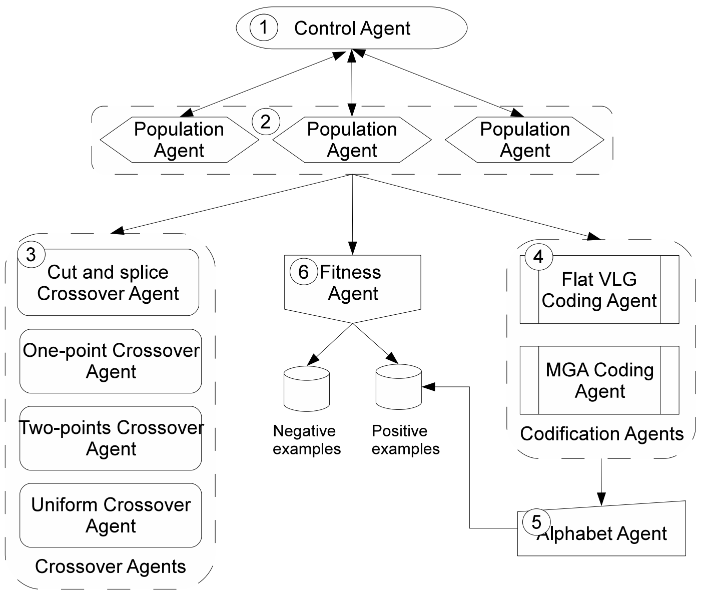


Fig. 2: MAS architecture used in the evaluation of the proposed crossover operators and codifications.

chromosome is the Fitness Agent (6) because it receives the regex in form of string, instead of a binary chromosome.

6. Experimental study

This section describes the behaviour and extraction capabilities of the coding and crossover mechanisms described in sections 3 and 4 using a MAS.

6.1. Experimental setup

Experiments have been carried out in three phases: parameter tuning, codification and genetic regex. In the first one, several GA runs are performed with different parameters to select the optimum parameters in order to use them in the remaining experiments. Then the codification (VLG, mGA and bmGA) and genetic operators (modified one-point, two-points and uniform crossover) presented in this paper are studied.

Three case studies are used in the experiments where regex able to extract emails, phone numbers and URLs are evolved. These are three well known problems in data mining literature. Each case study uses a dataset with positive examples that have been divided into a training set and a testing set. Meanwhile the negative examples are shared among the study cases. Due to the stochastic nature of the GAs, each experiment has been run one hundred times, and the data has been averaged.

The fitness evaluation that has been used in all the experiments can take values from 0 to 1, where 1 is the maximum fitness that any chromosome can achieve. The calculus of the fitness is performed as follows. 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 the characters correctly extracted. In this way, the maximum fitness that a chromosome can achieve is one, and it happens when the phenotype has extracted correctly all the elements of positive examples while no element of the negative examples has been matched. Let us name it as ideal individual. Then, an ideal individual is able to extract correctly all the elements of positive examples while no element of the negative examples is matched.

6.2. Experimental results

The first experimental phase is a study about the behaviour of the algorithms under study under different parameter settings, whose aim is to select the GA parameters. Results are shown in Table 2. Optimum parameter values are similar for all the investigated algorithms with one notable exception, the mutation probability. Algorithms that use cut and splice crossover operator (VLG and bmGA cs) have an optimum mutation probability around one order of magnitude lower than the others algorithms (bmGA with any form of our proposed crossover). The higher disruptive capabilities of cut and splice operator compared to the proposed crossover operator may explain this difference. Parameters shown in Table 2 are the ones used along the rest of the experimentation described in this paper.

A second set of experiments were executed to study the performance of the three described codifications. In order to obtain comparable results, a cut and splice recombination operator has been used in all the experiments belonging to this second experimental stage. The three case studies yield similar experimental results, as can be seen in Fig. 3. Results suggest that plain VLGA achieves higher best fitness, however Fig. 3(b) shows, in generation 60, a slightly higher fitness for bmGA. In any case, plain VLG increases its best fitness faster than messy codifications due to its smaller chromosome: plain VLG does not need to codify the locus.

Compared to mGA, bmGA performs slightly better, specially in the phone numbers case study (see Fig. 3(b)). The better performance of bmGA compared to mGA in our experiments can be explained by the dynamics of the construction of the phenotype. Using a pure mGA, the first position of an atomic regex is 0, and thus the regex cannot be expanded to the left because there is no natural number lower than 0. BmGA places the first regex in $bias$ and thus by

Table 2: Parameters for the experiments carried out using a basic VLG (VLG), bmGA with cut and splice crossover (mbGA cs), bmGA with modified one-point, two-points and uniform crossovers (bmGA one, bmGA two and bmGA uni)

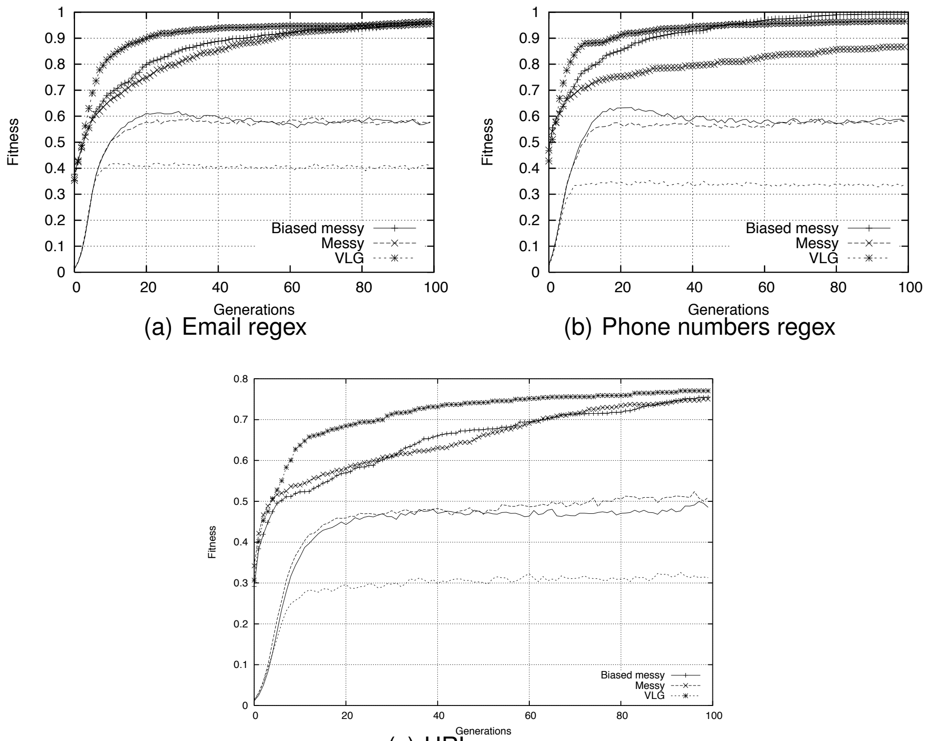
Settings	VLG	bmGA cs	bmGA one	bmGA two	bmGA uni
Mutation probability (P_{mut})	0.005	0.002	0.02	0.01	0.015
Population size (n)	50	50	50	50	50
Tournament size (t)	3	3	3	3	3
Min. chromosome length (l_{min})	4	9	9	9	9
Max. chromosome length (l_{max})	40	90	90	90	90
Gene length (l_g)	4	9	9	9	9
Loci length (l^{loci})	-	5	5	5	5
Values length (l^{values})	-	4	4	4	4
Crossover probability (P^c)	-	-	-	-	0.3

Table 3: Comparison of crossover operators for email regex induction: Cut and splice crossover (cs), modified one-point (one), two-points (two) and uniform crossovers (uni).

	cs	one	two	uni
Best fitness	0.96	0.94	0.9	0.94
Avg. fitness	0.58	0.42	0.58	0.46
Prob. ideal	0.86	0.78	0.64	0.77

Table 4: Comparison of crossover operators for phone number regex induction: Cut and splice crossover (cs), modified one-point (one), two-points (two) and uniform crossovers (uni).

	cs	one	two	uni
Best fitness	0.99	0.97	0.95	0.98
Avg. fitness	0.58	0.43	0.6	0.43
Prob. ideal	0.90	0.77	0.66	0.80



(a) Email regex (b) Phone numbers regex

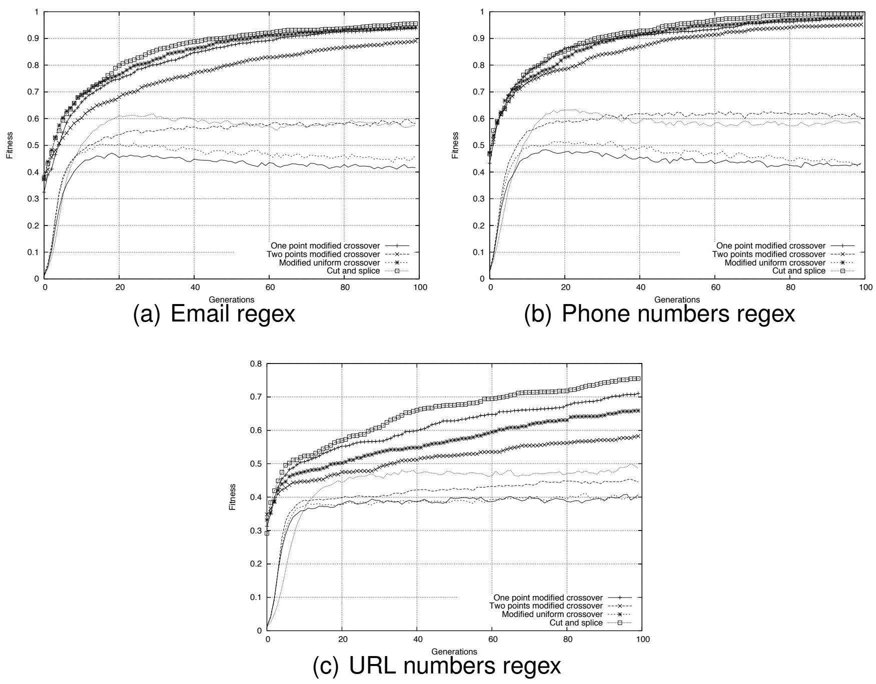
(c) URL regex

Fig. 3: Comparison of codifications in regex evolution. Best individual and average fitness are shown.

means of mutation and crossover regex can grow to the left, avoiding premature convergence.

The third group of experiments deals with the empirical study of several crossover mechanisms for messy algorithms. Experiments showed that bmGA performs better than mGA, and therefore bmGA is used in this section to compare several crossover operators, including cut and splice and the proposed operators described in sec. 4. Tables 3 and 4 show the best fitness, mean fitness and probability of finding an ideal individual for both case studies being investigated. Results show that the crossover operator has a limited effect in the fitness. Cut and splice seems to outperform the other operators, however it would be desirable to use hypothesis contrast to proof it.

The evolution of best and average fitness for the crossover operators under study are depicted in Fig. 4 for the three study cases: email (a), phone number (b) and URL evolution (c). It can be seen that the crossover operator has a limited effect the in fitness evolution, cut and splice performs slightly better that the other operators, however there is a small difference. The operator that per-



(a) Email regex (b) Phone numbers regex

(c) URL numbers regex

Fig. 4: Dynamics of fitness for crossover operators under study with a bmGA codification.

forms worse is the two points crossover operator, while there is no substantial difference between the modified one-point and uniform crossover.

7. Conclusions and future work

We have presented a method to generate regular expressions using supervised learning and an agent based testing framework. The distributed testing framework used has been a satisfactory approach due to the enhanced performance and easy composition of tasks involved in the GA. Additionally a brief empirical analysis of how different codifications and crossover mechanism influence the evolution of regex has been presented. The set of experiments carried out showed that the best performance is achieved with a direct codification of the alphabet using a plain VLG.

These results leads us to conclude that there are some intrinsic limitations in the evolution of regex regardless of the codification and crossover operator used. The main one is the linear nature of GAs that incited us to use a lexicographical codification of regex, there is not a trivial way to code regex operators that affect a groups of characters or nested operators. From this point of view, it seems that the use of pure GAs to evolve grammars and languages has serious constrains.

Acknowledgments

This work has been partially supported by the Spanish Ministry of Science and Innovation under the projects COMPUBIODIVE(TIN2007-65989), V-LeaF (TIN2008-02729-E/TIN), Castilla-La Mancha project PEII09-0266-6640 and HADA (TIN2007-64718).

References

1. Barrero, D.F., Camacho, D., R-Moreno, M.D.: Data Mining and Multiagent Integration, chap. Automatic Web Data Extraction Based on Genetic Algorithms and Regular Expressions. Springer (Aug 2009)
2. Barrero, D.F., González, A., R-Moreno, M.D., Camacho, D.: Variable length-based genetic representation to automatically evolve wrappers. International Conference on Practical Applications of Agents and Multi-Agents System, vol. 2, p. To Appear. Springer (2010)
3. Barrero, D.F., R-Moreno, M.D., López, D.R., García, Ó.: Searchy: A metasearch engine for heterogeneus sources in distributed environments. In: Proceedings of the International Conference on Dublin core and Metadata Applications. pp. 261–265. Madrid, Spain (Sep 2005)
4. Chu, D., Rowe, J.E.: Crossover operators to control size growth in linear GP and variable length GAs. In: Wang, J. (ed.) 2008 IEEE World Congress on Computational Intelligence. IEEE Computational Intelligence Society, IEEE Press, Hong Kong (1-6 Jun 2008)
5. Cicchello, O., Kremer, S.C.: Inducing grammars from sparse data sets: a survey of algorithms and results. J. Mach. Learn. Res. 4, 603–632 (2003)
6. Deb, K.: Binary and floating-point function optimization using messy genetic algorithms. Ph.D. thesis, Tuscaloosa, AL, USA (1991)
7. Dunay, B.D., Petry, F., Buckles, B.P.: Regular language induction with genetic programming. In: Proceedings of the 1994 IEEE World Congress on Computational Intelligence. pp. 396–400. IEEE Press, Orlando, Florida, USA (27-29 June 1994)
8. Friedl, J.E.F.: Mastering Regular Expressions. O'Reilly & Associates, Inc., Sebastopol, CA, USA (2002)
9. Galinho, V.J.P., Thierry, G., Franck, L., Alain, C., Rouen, I.D., Blondel, P.E.: Genetic algorithms in a multi-agent system (1997)
10. Gold, E.M.: Language identification in the limit. Information and Control 10(5), 447–474 (1967)
11. Goldberg, D., Deb, K., Korb, B.: Messy genetic algorithms: motivation, analysis, and first results. Complex Systems 3(3), 493–530 (1989)
12. González-Pardo, A., Barrero, D.F., Camacho, D., R-Moreno, M.D.: A case study on grammatical-based representation for regular expression evolution. International Conference on Practical Applications of Agents and Multi-Agents System, vol. 2, p. To Appear. Springer (2010)
13. Harvey, I.: The saga cross: the mechanics of recombination for species with variablelength genotypes. In: In R. Manner & B. Manderick, (Eds.), Parallel Problem. pp. 269–278. North-Holland (1992)

14. Jennings, N.R., Sycara, K.: A roadmap of agent research and development (1998)
15. Koza, J.R.: Genetic Programming: On the Programming of Computers by Means of Natural Selection (Complex Adaptive Systems). The MIT Press (December 1992)
16. Lander, S.E., L, S.E., Lesser, V.R.: Understanding the role of negotiation in distributed search among heterogeneous agents (1993)
17. Lymperopoulos, D.G., Tsitsas, N.L., Kaklamani, Dimitra, I.: A distributed intelligent agent platform for genetic optimization in cem: Applications in a quasi-point matching method. IEEE Transactions on Antennas and Propagation 55(3), 619–628 (March 2007)
18. Maione, G., Naso, D.: A genetic approach for adaptive multiagent control in heterarchical manufacturing systems. IEEE Transactions on Systems, Man, and Cybernetics Part A: Systems and Humans 33(5), 573–588 (September 2003)
19. O'Neill, M., Ryan, C.: Grammatical evolution. IEEE Transactions on Evolutionary Computation 5(4), 349–358 (August 2001)
20. Parekh, R., Honavar, V.: Grammar inference, automata induction, and language acquisition. In: Handbook of Natural Language Processing. pp. 727–764. Marcel Dekker (1998)
21. Rana, S.: The distributional biases of crossover operators. In: Proceedings of the Genetic and Evolutionary Computation Conference. pp. 549–556. Morgan Kaufmann Publishers (1999)
22. Ricardo Aler, D.C., Moscardini, A.: The effects of transfer of global improvements in genetic programming. Computing and Informatics (formerly: Computers and Artificial Intelligence) 23(4), 377–394 (2004)
23. Sakakibara, Y.: Recent advances of grammatical inference. Theor. Comput. Sci. 185(1), 15–45 (1997)
24. Spears, W.M.: Crossover or mutation. In: Foundations of Genetic Algorithms 2. pp. 221–237. Morgan Kaufmann (1993)
25. Ziv, J., Lempel, A.: A universal algorithm for sequential data compression. IEEE Transactions on Information Theory 23(3), 337–343 (1977), http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=1055714

David F. Barrero is a Lecturer at the Computer Engineering Department of the University of Alcal. He holds a Telecommunications Engineering degree and he recived a MSc. in Computer Science from the University of Alcal. Nowdays he is PhD candidate at the same University. He has previously spent six months in the Centre National d'Etudes Spatiales (CNES) in Toulouse, France. His research focuses on experimental methods in Evolutionary Computation and automatic learning of regular expressions.

Antonio González-Pardo holds a BSc in Computer Science from Universidad Carlos III de Madrid (2009). Nowadays, he is a Computer Science PhD candidate at Escuela Politcnica Superior (UAM) ¡http://www.ii.uam.es¿. Currently he is involved with ACIDA interest research group and GNB group ¡http://arantxa.ii.uam.es/main
research interests are related to Genetic Algorithms, Information Theory (compression-based metrics), information extraction (grammatical and evolutionary-based of regular expressions) and Signature Neural Networks.

David Camacho is an Associate Professor in the Computer Sciennce Department at Universidad Autnoma de Madrid (Spain). He received a Ph.D. in Computer Science (2001) from Universidad Carlos III de Madrid for his work on coordination of planning heterogeneous agents to solve problems with information gathered from the Web. He received a B.S. in Physics (1994) from Univesidad Complutense de Madrid. He has published over 50 journal, books, and conference papers. His research interests include Multi-Agent Systems, Distributed Artificial Intelligence, Web Service Technologies, Knowledge representation, Automated Planning and Machine Learning. He has also participated in several projects about automatic machine translation, optimising industry processes, MultiAgent technologies and Intelligent Systems. He is the managing editor of the International Journal of Computer Science & Applications (IJCSA), and has been selected as a chairman and member of the organizing committee for several international conferences.

María D. R-Moreno received a M.S. degree in Physics from the Universidad Complutense de Madrid (Spain) and her PhD in Computer Science from Universidad de Alcalá in Madrid (Spain). She has spent one year at NASA Ames Research Center as a postdoc, and nine weeks as a Research Visitor at ESA's European Space Research and Technology Centre (ESTEC).She is currently Associate Professor in the Departamento de Automatica at the Universidad de Alcala. She has previously worked as an Associate professor and as a consultant. She has publications in prestigious journal such as AI Magazine, Expert Systems with Application or IEEE Transactions on Knowledge and Data Engineering.

Dr. R-Moreno is actively collaborating in ESA projects and participating with research groups at NASA. She has served in the program committee of several international AI conferences and reviewer of the IEEE Transactions on Knowledge and Data Engineering journal. She is member of the Editorial Board of IJCSA and IAENG Journals, and member of the Experts Group of the Editorial Idea Group Inc. Her research focuses on Automated AI Planning & Scheduling, Monitoring and Execution applied to real applications (i.e. aerospace, e-learning or the web) and Genetic Programming.

Received: January 18, 2010; Accepted: May 3, 2010.