Automatic Web Data Extraction based on Genetic Algorithms and Regular Expressions

David F. Barrero and David Camacho and Maria D. R-Moreno

[1] Computer Science Department
Universidad de Alcalá
Madrid, Spain, dfbarrero@aut.uah.es
[2] Computer Science Department
Universidad Autónoma de Madrid
Madrid, Spain, david.camacho@uam.es
[3] Computer Science Department
Universidad de Alcalá
Madrid, Spain mdolores@aut.uah.es

Abstract. Data Extraction from the World Wide Web is a well known, unsolved, and critical problem when complex information systems are designed. These problems are related to the extraction, management and reuse of the huge amount of Web data available. These data usually has a high heterogeneity, volatility and low quality (i.e. format and content mistakes), so it is quite hard to build reliable systems. This chapter proposes an Evolutionary Computation approach to the problem of automatically learn software entities based on Genetic Algorithms and Regular Expressions. These entities, also called *wrappers*, will be able to extract some kind of Web data structures from examples.

Key words: Regular expressions, Genetic Algorithms, Evolutionary Computation, grammatical rules

1 Introduction

Flexible and scalable mechanisms are needed for the integration of information in order to obtain the necessary data from available sources. However, if these sources are not structured, for instance being relational-based, or no design has been previously made by an expert (i.e. a database designer) it is usally difficult to build, and maintain those mechanisms. The previous situation becomes a critical issue when talking about the World Wide Web, considered as a highly heterogeneous data source.
Web Data Extraction (WDE) is a well known and unsolved problem. Also it is related to the extraction, management and reuse of a huge amount of Web data available. These data usually has a high heterogeneity, volatility and low quality. One popular approach to address this problem is related to the concept of *wrappers*. The wrappers [5] are specialized programs that automatically extract data from documents and convert the stored information into a structured format.

The main contribution of this work is a novel approach to the WDE based on Genetic Algorithms (GA) [3] which are used to automatically evolve wrappers. The main difference with other closer approaches [1, 4, 6] is the utilization of Regular Expressions using a multiagent system to generate them and extract information [2]. A Regular Expression, or simply *regex*, is a powerful way to identify a pattern in a particular text. Any regex is written in a formal language, that is translated into a particular syntax like POSIX or Perl, and later processed by a regex engine such as Perl, Ruby or Tcl. Regular expressions are used by many text editors, utilities, and programming languages to search and manipulate text based on patterns.
This approach considers the basic (evolved) regex as the atomic extraction element. The representation, genetic operators and fitness function are designed in order to obtain simple extraction elements that are later used, shared, and integrated by a set of information extraction agents. A multi-agent semantic integration plataform named Searchy [1] is used to deploy and test the evolved regex. This approach has to find answers for two important questions. First, how the regex can be represented taking into account its particular features, i.e. vocabulary, syntaxis, grammar and semantic relationships between the grammatical syntaxis and the patterns that it can extract. Second, how once a particular individual is found, it can be combined, or integrated, with others to build a new data extraction (regex).
The following corresponds to the structure of this chapter. Section 2 describes the basic concepts in GA and its application to wrappers and regular expressions. Section 3 explains how a variable length population of agents can support the evolution of regular expressions. Section 4 shows how a specific information agent uses simple gramatical rules to combine, and integrate, the evolved atomic regular expressions. Section 5 shows the experimental results obtained for a set of web documents. Finally, some conclusions and future lines of work are outlined.

2 Genetic Algorithms and its application in wrappers and regular expressions

This section briefly explains basic concepts related to GA and regex that later will be used to automatically obtain the wrappers.

2.1 Genetic Algorithms

From the AI point of view, GA can be seen as a stochastic search algorithm inspired in the biological evolution. GA code the solution of a problem using a string called chromosome or individual, each chromosome represents a point in the search space [3]. If the GA is successful, the individuals will evolve exploring the search space until a global solution is found and the individuals will converge in that solution. The success or failure of a GA depends on the four principal parameters: Genome codification, genetic operators, selection strategy and fitness function.

Genome codification is a key subject in any GA. Each chromosome contains genetic information that codes a solution, therefore it will need a mechanism to mappings between the solution (phenotype) and the gentic code (genotype). Fig. 1 represents an example of binary fixed-length coding. The individual represented is the string *[rc]at*. Each attribute in the individual (i.e. each character in the string) is coded by four bits in the chromosome. The piece of chromosome that codes one attribute is called *gen*. Thus, in the example one *gen* codes one character using four bits.

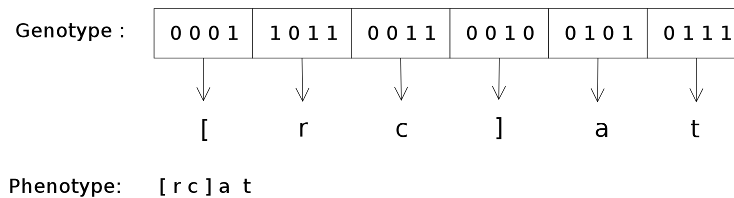

Fig. 1: Chromosome coding example

Once the coding has been defined, it is necessary to modify the genetic code of the population in order to explore the solution space. Genome modifications are done by the *genetic operators*. There are two main types of genetic operations: recombination and mutation. Recombination, also known as *crossover*, aims to imitate biological sexual reproduction. It consists of interchanging the genetic code of two individuals. A simple recombination algorithm is the one-point crossover, that is, it interchanges two chunks of chromosomes cutting them in a random point. Another key genetic operator is *mutation*, it introduces random changes in the genome that can generate new attributes in the phenotype not presented previously.

It is necessary to introduce a *selection strategy* in order to improve the population in the successive iterations of the GA (generations). It is analogous to the biological natural selection. The goal of the selection strategy is to generate a selective pressure, this means to force good chromosomes to have more probabilities to reproduce than bad ones. However, there is not a clear non-ambiguous meaning for "good" and "bad" yet.

Goodness and badness are two fuzzy concepts that cannot be used in a scientific context without a precise definition. GA defines good and bad using a *fitness function*. It is a basic piece of any GA, and it is usually one of the most challenging problems that must be faced in order to successfully implement a GA. In some cases defining a fitness function is a trivial issue; however in other problems the definition of the fitness function is more complex. This is the case of the regex evolution.

2.2 Regular expressions

Regular expressions [8] conform a powerful tool to define string patterns. Then, using regex makes possible to manipulate strings according to a potentially quite complex pattern. An extended and well known use of regex is to define sets of files in many user interfaces. The string *rm *.jpg* means in a UNIX shell delete all the files whose name ends in *.jpg*. Actually **.jpg* is a regex representing the set of all strings that ends in *.jpg*.
Many practical applications have been found for regex, especially in the UNIX community, that has achieved a long experience using this tool. Indeed, regex is a basic feature of shell commands like *ls grep* and some programming languages largely used by the UNIX community like Perl or AWK.
Regex is a powerful tool, with a wide range of applications but generation of regex is a tedious, error prone and time consuming task, especially when dealing with complex patterns that require complex regex. Reading and understanding a regex, even if it is not very complex, is far from being an easy task. In order to ease regex generation, several assistant tools have been developed, but writing regex is still a problematic task. An automatic way to generate regex using Machine Learning techniques is a desirable goal that could likely exploit the potential that regex provides. Our approach proposses two stages for the generation of regex. In a first stage a multiagent system is used to evolve a variable length regex able to extract data from documents that follow a known pattern. Then, a second stage that uses two or more evolved specialized regex to compose a complex regex able to extract and integrate several types of data.

3 How agents support data mining: Variable length population

The first stage of the data extraction mechanism proposed in this paper deals with the automatic generation of a basic regex. The aim is to use supervised learning to automatically generate a regex in an evolutionary fashion. A multiagent system (MAS) is used in order to generate basic regex able to extract information to conform a pattern, such as phone numbers or URLs. The agents share a training set composed by positive and negative examples that are used to guide the evolutionary process until the regex is generated.
Extraction capabilities from a regex are closely related to its length, and the length of the regex is determined by the length of the chromosome. Traditional fixed-length GA introduce an arbitrary constrain to the size of the evolved regex that should be avoided for many reasons. The GA should be able to self adapt its genome length without human intervention. One solution might be the use of variable-length genomes, though our interest is an intrinsic parallel solution like a MAS.
Regex are generated by a MAS that unfolds a variable length genome, where subsets of agents use a fixed length genome, as it is shown in fig. 2. Each agent runs a GA containing a population whose individuals own

a chromosome of fixed length, and can evolve by its own with a high degree of independence that conform a microevolution. Agents are not isolated thus their populations are influenced by other populations by means of emigration: a part of the population can emigrate from one agent to another agent every generation, so the evolution of one subpopulation is affected by the evolution of other agents. The result is that the total population of the MAS presents a macroevolution. Microevolution and macroevolution are different problems that must be addressed individually.

3.1 Macroevolution

The MAS is actually a way to implement variable-length GA, in which the sets of agents containing populations of different length evolve as a whole. The mechanism that makes this macroevolution possible is the population interchange among agents. An agent containing a population with a chromosome of length n always clones a number of individuals to a population with a chromosome size $n + k$, where k is the gen size (see fig. 2). Thus, the genetic operation that modifies the length of the chromosome is performed when the individual is emigrating, adding a new chunk in a random position of the chromosome. The chunk that is inserted into the genome is a non-coding region, i.e., a chunk that codes an empty character and therefore, it does not affect the phenotype. Otherwise the potentially good genetic properties of the individual might be lost.

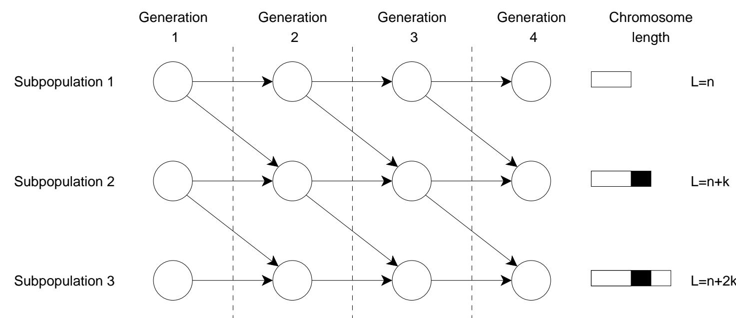

Fig. 2: Population interchange among agents

An important issue is the selection process of the individuals that emigrate and the selection of the individuals that are replaced in the target population. Both selections are done using a tournament, however, there is a difference. The tournament of the individuals that emigrate is won by the individual with the best fitness, meanwhile the tournament between the individuals to be replaced is won by the individual with the

worst fitness. In this way, individuals with high fitness in a population have more chances to emigrate than individuals with a bad fitness. On the other hand, chromosomes with low fitness will likely be replaced by better immigrant chromosomes.
Each agent in the MAS contains chromosomes of different sizes and they send or receive a bunch of individuals each generation, which unfolds a population using the same parameters, strategy and fitness function.

3.2 Microevolution

A classical fixed-length GA is run within each agent, evolving a population of chromosomes in a microevolution.It can be described in terms of genome codification, genetic operators, selection strategy and fitness function.
The GA implemented in the MAS uses a binary genome divided in several gens of fixed length. Each gen represents a symbol from an alphabet composed by a set of valid constructed regular expressions. It is important 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, so, we will call them atomic regex.
Genetic operators used in the evolution of regular expressions are the mutation and crossover. Since the codifications rely in a binary representation the mutation operator is the common inverse operation meanwhile the recombination is performed with a one-point crossover. These genetic operators do not modify the genome length; chromosomes modify their length only when an individual is migrating to another agent. The selection mechanism used is the tournament selection.
From a formal point of view, the fitness function is defined as follows. Given a set of positive examples, P, with M elements, and a set of negative examples, Q, with N negative examples, let $p \in P$ be an element of P, and $q \in Q$ an element of Q, we define $\Omega = \{\omega_0, \omega_1, ..., \omega_n \mid \omega_i \in P \cup Q, n = N + M\}$ as the set of elements contained by P and Q; therefore any element of P or Q belongs also to Ω. Let the set of all regular expressions be R, and r an element of R. Then, we can define a function φ_r, $\varphi_r(\omega)\colon \Omega \times R \longrightarrow \mathbb{N}$ as the number of characters of ω that are matched by the regex r.
Finally the fitness function $\mathfrak{F}\colon Q \longrightarrow \mathbb{R} \in [-1, 1]$ is defined as:

$$\mathfrak{F}(r) = \frac{1}{M} \sum_{p_i \in P} \frac{\varphi_r(p_i)}{\mid p_i \mid} - \frac{1}{N} \sum_{q_i \in Q} M_r(q_i) \tag{1}$$

where $\mid \omega_i \mid$ is the number of characters of ω_i and $M_r(q_i)$ is:

$$M_r(q_i) = \begin{cases} 1 \; if \; \varphi_r(q_i) > 0 \\ 0 \; if \; \varphi_r(q_i) = 0 \end{cases} \tag{2}$$

Since true positives are calculated based on the characters, and false positives have no intermediate values, the fitness function presents an intrinsic bias: it is more sensible to false positives than to true positives. It is important to point out that the maximum fitness that an individual

can achieve is 1 and it is given when all the positive examples have been completely retrieved and no negative example has been matched. If a chromosome obtains a fitness of 1, it will be named an ideal chromosome. From the evolution of each specialized MAS we obtain a basic regex able to extract strings matching a pattern. Each MAS requires a training set and, eventually, an appropriate alphabet of atomic regex. Once the basic regex has evolved, it is possible to build more complex regex in the second stage of the extraction process.

4 Composition of basic regex

We use the grammatical rules provided by the regex notation in a composition agent to integrate the basic evolved regex from the first stage. The composition agent uses a manually created rule database to integrate two or more basic regex. The agent applies the grammatical rules to the input regex obtaining a set of composed, potentially complex, regex. They might be not suitable to extract information properly, so, regex created by the grammatical rules have to be filtered in order to select the valid ones. We use the traditional data mining F-measure to automatically evaluate its extraction capabilities and select the composed regex.

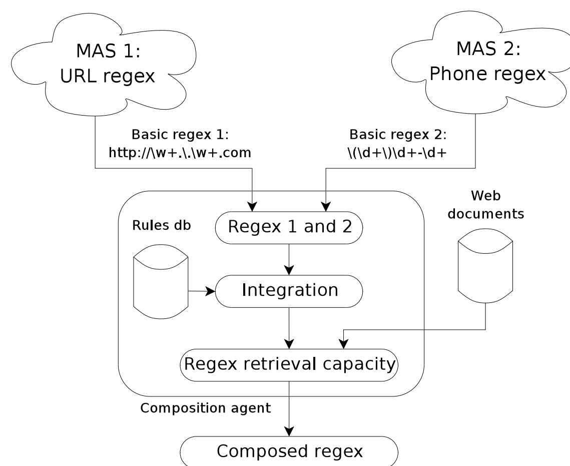

Fig. 3: Composition agent architecture

A graphical representation of the composition process can be seen in fig. 3, where an example of regex composition is depicted. The composition agent takes two basic regex from the output of two evolutive MAS (see section 3.2) and applies a set of grammatical rules stored in a database to generate a set of composed regex.

Suppose that the composition agent takes *http://\w+.\.\w+.com* and *\(\d+\)\d+-\d+* as basic regex, and the aim is to compose them using a subset of regex operators, for example, |, *(,), +* and *?*, then it is possible to define a database of grammatical rules in the composition agent such as:

```
Rule 1: X|Y
Rule 2: XY
Rule 3: X+Y?
Rule 4: (Y)+|((X)+|foo)
```

Where X and Y are *http://\w+.\.\w+.com* and *\(\d+\)\d+-\d+*. The composition agent applies the grammatical rules to the input regex generating the following set of composed regex:

```
Composed 1: http://\w+\.\w+\.com|\(\d+\)\d+-\d+
Composed 2: http://\w+\.\w+\.com\(\d+\)\d+-\d+
Composed 3: http://\w+\.\w+\.com+\(\d+\)\d+-\d+?
Composed 4: (\(\d+\)\d+-\d+)+\((http://\w+\.\w+\.com)+|foo)
```

Since the regex composition has used a brute force approach, not all the composed rules are supposed to be able to correctly extract data. Therefore, it is necessary to select at least one valid regex. This is done by the regex retrieval capacity that evaluates the generated regex calculating the F-measure using a dataset composed by several documents (see equation 1.3). This measure is based on the weighted *harmonic mean* from classical Information Retrieval *Precision* (P) and *Recall* (R) values. Of course, other extraction quality measures such as F_β or E are also valid. Based on these quantitative measures, automatic estimation of the best composed regex is possible.

$$F_{measure} = \frac{2PR}{P + R} \tag{3}$$

5 Experimental evaluation

The experimental evaluation has been divided into three stages with different goals. The first stage is the setup of the experiment, in which several tests were carried out in order to set the basic GA parameters, necessary for the second stage in which the regex evolution uses a MAS. Finally, some grammatical rules are used with the evolved regex and they are automatically evaluated in order to obtain a final composed regex.
Some initial experiments were carried out to acquire knowledge about the behaviour of the regex evolution. In order to achieve this goal a

single GA was used with the same configuration required by the MAS. Due to the lack of space, only the most significant results are enumerated without further discussion.
Despite the differences between phone and URL, both evolved regex have similar behaviours, in this way it is possible to extrapolate experimental results. The best results are achieved with mutation probabilities between 0.01 and 0.02 thus an average mutation probability of 0.015 was fixed to carry out the rest of the experiments. Tournament size has shown to have a remarkable impact to obtain a faster convergence while avoiding local maximum. Experiments have demonstrated that a tournament size of three is a good balance. Variation in the size of the population shows the usual GA behaviour, a population with fifty individuals is a good trade-off between convergence speed and computational resources.

5.1 Results: regex evolution

The second experimental stage aims to use a MAS with subsets of agents where populations of chromosomes with different lengths are evolving. Six subsets of agents have been used with chromosomes sizes of 6, 9, 12, 15, 18 and 21 bits organized in chunks of three bits. The emigration strategy has been set as described in section 3.2. Agents in the MAS run a GA with the parameters obtained in the experiment setup phase and following the same experimental procedure.
Fig. 4 depicts the evolution of the average fitness of each subset of agents for the phone numbers regex evolution. Since the results for the URL regex evolution are analogous, no figure is included. It should be noticed the close relationship between the convergence speed and the chromosome size. The longer is the chromosome, the longer it takes to converge because the chromosome codes a solution in a bigger search space. Another fact that influences the difference in the convergence speed is found in the limited speed of propagation of good chromosomes along the MAS.

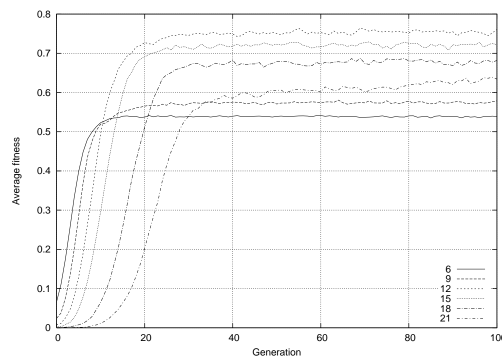

Fig. 4: Average fitness for the phone number regex evolution

Fig. 4 shows an interesting fact in relation to the different fitness convergente values in the agents for the phone number evolution. shows is the different fitness convergence values in the agents for the phone number regex evolution. A population with a chromosome length of 6 bits presents a fast convergence to 0.54. With 6 bits only very small phenotypes can be represented, just two symbols, so only part of the examples can be extracted, achieving a maximum fitness of 0.54. This fact is also found in populations of size 9, but with less dramatic effects. The populations with chromosomes of length 12 present a very important growth in the fitness. The reason is that the shortest ideal regex must be coded with at least 12 bits.

Table 1: Evolved regular expressions

Evolved regex (URL)	**Fitness**	**Evolved regex (Phone)**	**Fitness**
http://-http://http://	0	\w+	0
conwww\.http://com-www\.com	0	\(\d+\)	0.33
/\w+\.	0.55	\(\d+\)\d+	0.58
http://\w+\.\w+\	0.8	\(\d+\)\d+-\d+	1
http://\w+\.\w+\.com	1		

5.2 Results: regex composition

Two basic regex have been selected for the third evaluation stage, where they are composed and its extraction capabilities are measured using precision, recall and F-measure.
A dataset was needed to measure the precision and recall. The experiment used a dataset composed by ten documents from different origins containing URLs and phone numbers, mixed and alone. Table 3 shows basic information about the dataset and its records. Documents one, two and three are composed by examples extracted from the training set. The rest of documents are web pages retrieved from the Web; however documents five and six were transformed to a plain text format in order to remove all the URLs.
The calculus of precision and recall use the total number of records in the document, i.e., the sum of URLs and phone numbers, regardless the evaluated regex. It means that the result is strongly biased against regex that are not able to extract both URLs and phone numbers. It should be noticed also that an extracted string is true if and only if it matches exactly the records, otherwise it has been computed as a false positive.
Results, as can be seen in table 2, are quite satisfactory for the preprocessed documents, i.e., documents one to five, but measures get worse for real raw documents. X has a perfect precision; meanwhile Y has a poor average precision of 0.41. It can be explained looking at Y. This regex has the form *http://\w+\.\w+\.com*, which means that it only extracts the protocol and the host name from the URLs, but it cannot extract the path, a common part of URLs found in the Web.

A special case is the regex XY, a direct concatenation of X and Y. This regex extracts URLs followed directly by a phone number; obviously, such situation is not likely to happen. So, it is unable to extract any record (for this reason these results are not shown in the table). After all, the best balance between precision and recall is achieved by the composed regex $(X) \mid (Y)$, with a precision of 0.63 and a recall of 0.62.

Table 2: Extraction capacity of basic and composed regex. It is calculated using traditional precision (Prec.) and Recall values. The table shows the Retrieved elements (Retr) and the True Positives (TPos) detected.

	X				**Y**				**(X)|(Y)**			
	Retr	**TPos**	**Prec.**	**Recall**	**Retr**	**TPos**	**Prec.**	**Recall**	**Retr**	**TPos**	**Prec.**	**Recall**
Document 1	5	5	1	1	0	0	-	-	5	5	1	1
Document 2	0	0	-	-	5	5	1	1	5	5	1	1
Document 3	5	5	1	0.5	5	5	1	0.5	10	10	1	1
Document 4	99	99	1	1	0	0	-	-	99	99	1	1
Document 5	10	10	1	1	0	0	-	-	10	10	1	1
Document 6	0	0	-	-	43	6	0.14	0.12	43	6	0.14	0.12
Document 7	20	20	1	0.21	773	12	0.16	0.12	97	32	0.33	0.33
Document 8	37	37	1	0.05	668	76	0.11	0.11	705	113	0.16	0.16
Document 9	24	24	1	0.13	88	1	0.01	0.01	112	25	0.22	0.14
Document 10	0	0	-	-	49	23	0.47	0.45	49	23	0.47	0.45
Average			1	0.56			0.41	0.33			0.63	0.62

Precision and recall balance can be quantified with the F-measure (shown in table 3). The $(X) \mid (Y)$ regex obtained the best value followed not far by X. As it was expected, the composition agent selectes $(X) \mid (Y)$ based on the F-measure.

Table 3: Document record types and F-measure of regexes.

	URL	Phone	**X**	**Y**	**(X)|(Y)**
Document 1	0	5	1	-	1
Document 2	5	0	-	1	1
Document 3	5	5	0.67	0.67	1
Document 4	0	99	1	-	1
Document 5	0	10	1	-	1
Document 6	0	51	-	0.12	0.12
Document 7	77	20	0.35	0.14	0.33
Document 8	436	37	0.09	0.11	0.16
Document 9	241	24	0.23	0.01	0.17
Document 10	51	0	-	0.46	0.46
Average			0.62	0.35	0.69

6 Conclusions

An innovative approach for data extraction based on regex evolution and grammatical composition of regex has been presented. We have shown that it is possible to use a GA to evolve regex in a MAS and to apply grammatical rules to the evolved regex in order to generate a composition of regular expressions with the capacity to extract different records of data.
Using a MAS to simulate a variable-length genome population has showed to be a successful way to generate a variable-length chromosome evolution. Each agent is able to evolve a population and the MAS presents a macroevolution that tends to generate regex correctly sized.
However, the experiments carried out show some limitations. The linear nature of the GA codification is not the best option to represent a hierarchical structure such as a regex. The result is a natural difficulty to define a fine-grained fitness function able to evaluate not only all the regex, but also its parts. For these reasons the next step to follow is to use other evolutionary algorithms, such as Genetic Programming and Grammatical Evolution that overcome this limitation.
Finally, grammatical rules offer a simple way to automatically compose basic regex and select the best composed regex measuring its F-measure with a set of documents.

Acknowledgements

This work has been supported by the research projects TIN2007-65989, TIN2007-64718 and Junta de Castilla-La Mancha project PAI07-0054-4397.

References

1. David Camacho, Maria D. R-Moreno, David F. Barrero, and Rajendra Akerkar. Semantic wrappers for semi-structured data extraction. *Computing Letters (COLE)*, 4(1), 2008.
2. Longbing Cao, Chao Luo, and Chengqi Zhang. Agent-mining interaction: An emerging area. In *AIS-ADM*, pages 60–73, 2007.
3. John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, April 1992.
4. Marat Kanteev, Igor Minakov, George Rzevski, Petr Skobelev, and Simon Volman. Multi-agent meta-search engine based on domain ontology. In *AIS-ADM*, pages 269–274, 2007.
5. Nicholas Kushmerick. Wrapper induction: Efficiency and expressiveness. *Artificial Intelligence*, 118:2000, 2000.
6. M. Michalowski, J.L. Ambite, S. Thakkar, R. Tuchinda, C.A. Knoblock, and S. Minton. Retrieving and semantically integrating heterogeneous data from the web. *IEEE Intelligent Systems*, 19(3), 2004.
7. Ken Thompson. Programming techniques: Regular expression search algorithm. *Commun. ACM*, 11(6):419–422, 1968.