DE GRUYTER OPEN

STUDIES IN LOGIC, GRAMMAR AND RHETORIC 56 (69) 2018

DOI: 10.2478/slgr-2018-0049

Automatic Generation of Regular Expressions for Extracting Attribute Values of Medical Products

Tomasz Łukaszuk[1], **Mariusz Ferenc**[2]

[1] Faculty of Computer Science, Bialystok University of Technology, Poland

[2] Ediom sp. z o.o., Poland

Abstract. Resources of professional companies operating on the medical services market contain data from a huge number of transactional documents. This allows them to collect and process, among other actions, information about medical products. Organized data is obviously more valuable. In this paper, the possibility of supporting the process of organizing information is considered, with the goal to extract values of attributes of medical products from brief descriptions in transactional documents. This helps to build a structured product specification and makes it possible to make comparisons between products. For this purpose, an approach based on regular expressions and their generation with the use of the genetic algorithm is proposed. The results presented in the paper show a great potential of the presented method.

Introduction

Nowadays, online shops offer an ever-expanding range of products. The ability to compare them objectively is a significant advantage for customers. Many stores recognize this need and allow customers to compile a list of product attributes so that they can decide more easily on the best product according to their needs.

The facilities described above, however, mainly concern common types of products such as electronic devices, sports equipment, and food products. It is much more difficult to compile a list of attributes of products not purchased by ordinary customers, or of specialized products. Medical products are an important group of such products. As a rule, they are not of interest to ordinary customers, but rather to different types of health care units.

The market for medical products and services is of a huge financial value. As a result, there are companies on the market that offer health care units various types of services and intermediation in purchasing the necessary products. For such intermediaries, transactional documents containing, inter alia, brief descriptions of the products on sale constitute an important source of information about medical products.

With the very large number of medical products' descriptions, it is difficult to process them all manually, which is why machine processing support is needed. Regular expressions are an important aid in text processing and extraction of relevant fragments from it (Thompson, 1968). Once the appropriate regular expression is constructed, it can be used for many descriptions to extract the fragments that represent the values of medical products' attributes of interest.

In this paper, a method for generating regular expressions based on descriptions of products constituting a training sample is presented. Descriptions from the training sample were analyzed by an expert who indicated their important fragments. A genetic programming approach is studied to automatically generate a regular expression that extracts expert-indicated fragments in descriptions of medical products.

The remaining part of this paper is structured as follows: Section "Related works" discusses other works connected with the subject matter of this paper, particularly extraction of product value from textual descriptions and regular expression learning. Section "Problem to solve" contains the problem description. In Section "Methods", a relatively detailed algorithm for generating regular expressions is presented. Section "Experimental evaluation" reports experiments performed with the use of the developed algorithm and presents their results. Conclusions are presented in Section "Conclusions and Future Works".

Related Works

A good deal of important research into retrieving information from text documents has been carried out (see e.g. Banko et al., 2007; Chang et al., 2006; Köpcke et al., 2012; Sarawagi, 2008; Wu & Weld, 2010). This group also includes works aimed at extraction of product attributes and their values.

Ghani et al. (2006) presents an approach to the problem of extraction of attribute-value pairs from textual product descriptions on retail store websites. The authors consider both the implicit (defined by experts) and explicit (existing in the text) attributes and formulate both types of extraction as classification problems. They use semi-supervised learning algorithms, thus reducing the need for initial tagged data, which is expensive to obtain.

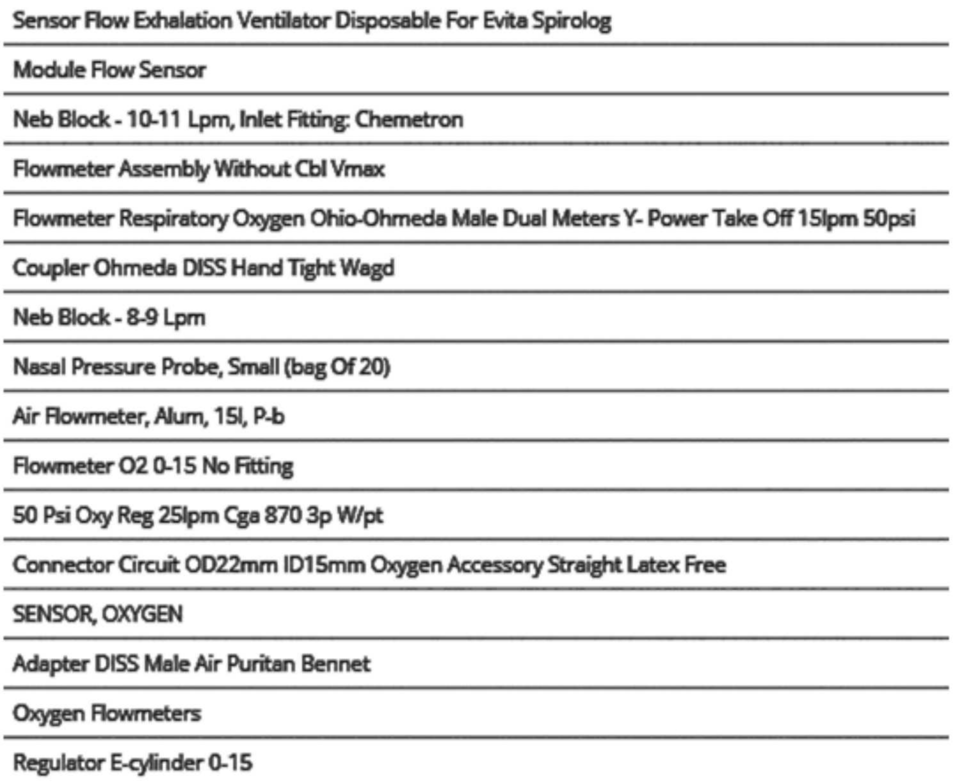
A more NLP-oriented approach is proposed in the work of Popescu and Etzioni (2007). Their system, called OPINE, attempts to identify product features and user opinions based on noun phrases derived from online user reviews. First they extract noun phrases as candidate attributes and then compute the PMI (pointwise mutual information) between the noun phrases and salient context patterns. Unfortunately, in informal texts, the formulation approach does not work well. In the case in question, product description may consist of loosely combined words, phrases and numbers, usually grammatically incorrect from the formal point of view.

In terms of its objective and methodology, this paper is the closest to the work of Petrovski et al. (2014). Their goal was to extract product attributes from e-shops' product offers by means of regular expressions in order to build well-structured product specifications. For this purpose, they present a technique for learning regular expressions. The problem considered in that article is almost identical to the one discussed in this paper. The genetic algorithm approach is also similar. The difference is in the representation of the regular expression and other components of the genetic algorithm.

Problem to Solve

The authors assume that the input data is a collection of short text descriptions of medical products. These descriptions are derived from sales documents, e.g. invoices issued by sellers to medical units. It is also assumed that the descriptions were pre-selected by experts so that they refer to products belonging to a single category, e.g. surgical gloves, intraocular lenses, or microplates. Examples of descriptions of medical products are shown in Figure 1.

Each product is described with the same set of known attributes A_i $(i = 1, 2, \dots, a)$. It is assumed that an expert is able to specify the value of a specific attribute A_i of a product in a given product description or indicate that there is no value of attribute A_i in the product description. Pairs consisting of a product description in which the expert found the value of attribute A_i and the value indicated by the expert are called *positive examples*. Pairs consisting of a product description in which the expert determined that there is no value of attribute A_i and the empty string are called *negative examples*.



Sensor Flow Exhalation Ventilator Disposable For Evita Spirolog
Module Flow Sensor
Neb Block - 10-11 Lpm, Inlet Fitting: Chemetron
Flowmeter Assembly Without Cbl Vmax
Flowmeter Respiratory Oxygen Ohio-Ohmeda Male Dual Meters Y- Power Take Off 15lpm 50psi
Coupler Ohmeda DISS Hand Tight Wagd
Neb Block - 8-9 Lpm
Nasal Pressure Probe, Small (bag Of 20)
Air Flowmeter, Alum, 15l, P-b
Flowmeter O2 0-15 No Fitting
50 Psi Oxy Reg 25lpm Cga 870 3p W/pt
Connector Circuit OD22mm ID15mm Oxygen Accessory Straight Latex Free
SENSOR, OXYGEN
Adapter DISS Male Air Puritan Bennet
Oxygen Flowmeters
Regulator E-cylinder 0-15

Figure 1. Examples of descriptions of medical products belonging to the "Flowmeters" category

The problem consists in constructing a regular expression R able to extract all the values indicated by the expert in *positive examples* and, at the same time, not to extract anything in *negative examples*. In the case where it is not possible to construct a regular expression that gives the correct result for all examples, the regular expression R should generate the least possible number of errors. Errors are situations when the regular expression R extracts a different value than the one indicated by the expert, or does not extract anything in *positive examples*; or when the regular expression R extracts a value in *negative examples*.

Methods

The authors' approach to solving the problem presented in the "Problem to solve" section is based on the genetic algorithm. However, the algorithm in question is not the genetic algorithm in the classical sense (Koza, 1992), due to the fact that the authors' algorithm does not contain too many elements of randomness, therefore the selection of next-generation parents is completely deterministic. The remaining steps of the algorithm, described in detail in subsequent paragraphs, correspond to the scheme of the classical genetic algorithm (Figure 2).

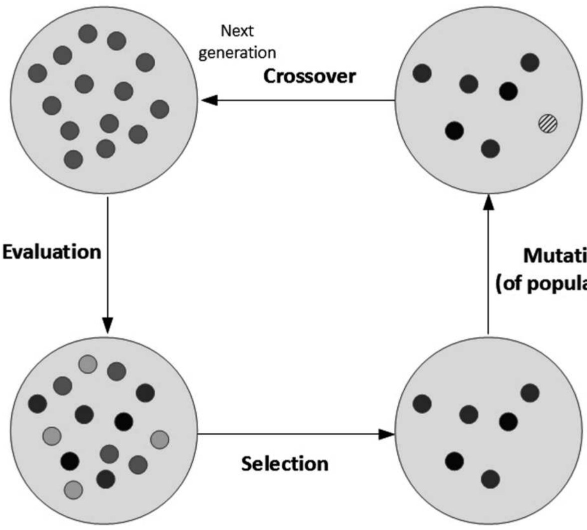


Figure 2. Diagram of the genetic algorithm for generating regular expressions

The genetic algorithm is an iterative optimization procedure. In each iterative loop, the following steps are carried out sequentially: evaluation, selection, and creation of the next generation of individuals through the use of genetic operators, i.e. crossover and mutation. The procedure ends when the new generation is no better than the previous one, or when a satisfactory result in terms of the value of the evolutionary function has been achieved. In addition, the classical genetic algorithm has many randomness aspects, particularly at the step of selection and in the use of genetic operators.

To solve a problem using the genetic algorithm, the key issue is to define the representation (Koza, 1992). In the authors' algorithm, an individual belonging to the population is the regular expression R, whose syntax and semantics are based on Perl Compatible Regular Expressions (PCRE) (Hazel, 2012), albeit with some limitations, detailed below.

The regular expression R consists of an ordered sequence of elements e_i $(i = 1, 2, \ldots, n)$:

$$R = e_1 \cdot e_2 \cdot \ldots \cdot e_n$$

where $\cdot$ means concatenation operation. Element e_i may be:

- terminal – a single character, a number or a string, including predefined character classes `\d` (any single digit), `\w` (any single word character), `\s` (any single whitespace), the wildcard `.` (any single character),
- character class – a set of allowed or disallowed characters, placed in parentheses `[]`,
- either element – any number of elements e_k $(k = 1, 2, \ldots, m)$ connected by operator `|`; it allows to match a string with one of elements e_k.

In addition, each element e_i has a specified count of occurrences: once or not at all (`?`), one or more (`+`), zero or more (`*`) or exactly one (no sign).

According to the above representation, the regular expression `[+-]?\d+\.\d+` that allows to find a real number in the string, consists of the following elements:

- e_1 – character class with characters `+` and `-` and count of occurrences `?`,
- e_2 – terminal `\d` with count of occurrences `+`,
- e_3 – terminal `\.` with count of occurrences exactly one,
- e_4 – terminal `\d` with count of occurrences `+`.

In addition, the regular expression R should have exactly one capturing group, i.e. a subsequence of elements e_i placed in parentheses `(` and `)`. The value extracted by the expression R from the examined string is a substring included in the capturing group. The regular expression `[+-]?(\d+)\.\d+` finds the real number, but the result of this extraction is only the integer part of the found real number.

Another key aspect of the genetic algorithm is to determine the evaluation criteria of the quality of individuals (Koza, 1992). In the discussed case, the chosen criterion is the quality of regular expressions. The fitness function should be a monotonous function that receives the regular expression R as input and returns a numeric value corresponding to the quality of R.

The regular expression R is used for each *positive example* and each *negative example*. As a result of applying R to a single example, the first matching substring found by R is taken. There are also cases where R is unable to find anything in a particular example. The following parameters are specified:

TP –	number of *positive examples* where R correctly found the value,
FN –	number of *positive examples* where R did not find the value or found an invalid substring,
FP –	number of *negative examples* where R incorrectly found a substring,
TN –	number of *negative examples* where R correctly found nothing.

Based on these counts, a value between -1 and 1 is assigned to the regular expression R by calculating the Matthews correlation coefficient (*MCC*) (Matthews, 1975):

$$MCC = \frac{TP \cdot TN + FP \cdot FN}{\sqrt{(TP + FP)(TP + FN)(TN + FP)(TN + FN)}}$$

The *MCC* value equaling 1 represents the perfect situation, i.e. R correctly found the values in all *positive examples* and correctly found nothing in all *negative examples*. The *MCC* value equaling 0 is a result no better than a random result. The *MCC* value equaling -1 indicates a total disagreement between the received and the expected results, i.e. R found nothing or incorrect substrings in all *positive examples* and incorrectly found substrings in all *negative examples*.

For the purpose of implementation, the final form of the fitness function is corrected as follows:

$$\mathit{fitness}(R) = 100 - 100 \cdot MCC + 0.001 \cdot \mathit{length}(R),$$

where *length*(R) returns the number of characters in the regular expression R.

The *fitness*(R) values are within the range $(0, \sim 200)$. The smaller the *fitness*(R) value, the better the regular expression R. In addition, if two regular expressions find exactly the same substrings, the shorter is the better one.

After selecting the best individuals from the current population, they are used to obtain the next generation. The new generation individuals are formed by means of crossover of individuals of the present population (Forrest, 1993).

In the authors' approach, k ($k \leq 100$) regular expressions $R_1, R_2, \ldots, R_k$ with the lowest values of the fitness function are chosen from the current population. Best regular expressions $R_1, R_2, \ldots, R_k$ are used to create crossovers. Each best regular expression R_i ($1 \leq i \leq k$) creates a crossover with each of the remaining $k - 1$ regular expressions R_j ($j \neq i$). The scheme of generating crossover of two regular expressions R_i and R_j is shown in below equation. The result of the crossover operation is the regular expression $R_{i \times j}$, which is able to find at least those substrings found by expression R_i or R_j.

$$R_i = e_{i1} \cdot e_{i2} \cdot (e_{i3} \cdot e_{i4} \cdot e_{i5}) \cdot e_{i6}$$

$$R_j = e_{j1} \cdot (e_{j2}) \cdot e_{j3}$$

$$R_{i \times j} = e_{i1}? \cdot (?: e_{i2}|e_{j1}) \cdot ((?: e_{i3}|e_{j2}) \cdot e_{i4}? \cdot e_{i5}?) \cdot (?: e_{i6}|e_{j3})$$

The second most commonly used genetic operator, apart from crossover, is mutation. In the classic genetic algorithm, mutation is a random tweak of a part of an individual (Koza, 1992).

In the authors' approach, individual regular expressions are not mutated. Instead, mutation is performed on the whole population. This works in the following manner: after choosing k best regular expressions $R_1, R_2, \ldots, R_k$ from the current population, and before creating their crossovers as the next generation, a random regular expression R_{k+1} is appended to the pool of best expressions. The regular expression R_{k+1} is the best matched random regular expression selected from 10,000 random regular expressions, according to the *fitness*(R) function. The mutated population is used to create a new generation of regular expressions.

The algorithm can create quite complex regular expressions, especially as a result of crossover. Some of their elements can be removed or simplified without affecting the found values. The simplified regular expression assumes exactly the same *fitness*(R) value as the expression before simplification. Simplification of regular expressions takes place after the best expressions from the current population are selected. The expressions used to create crossovers are simplified expressions.

Finally, what remains to be clarified are the issues of initiating and stopping the iteration process. The initial population of regular expressions is created on the basis of *positive examples*. For each such example, one regular expression is built and added to the initial population. Each character in the substring indicated by the expert as a value is replaced by the appropriate predefined character class `\d`, `\w` or `\s`. In addition, the literal words preceding and following the substring indicated by the expert are added to the regular expression as terminals. The quick rule of building individuals of the initial population is presented in Table 1.

Table 1. Examples of the creation of regular expressions added to the initial population

Positive example	Initial population regex
amplate skirt **384** wells flex plates blue	`skirt\s(\d\d\d)\swells`
plate well .2ml **96**u	`2ml\s(\d\d)u`

The iterative process stops when a regular expression for which the value of the *fitness*(R) function is less than 1 is obtained, or when no regular expression with a smaller *fitness*(R) function value than in the previous iteration can be obtained in the next iteration.

Experimental Evaluation

In this section, an experimental evaluation of the authors' approach described in the previous section is presented. Two experiments were conducted and an attempt was made to construct regular expressions that would allow to extract fragments representing the values of the considered product attributes from short product descriptions.

The first experiment was of an academic character. It was intended to confirm the correctness of the authors' approach on a relatively simple case, in which it was not difficult for a person to construct an appropriate regular expression.

Table 2. Input data for experiment 1 – selected products from the "Microplates" category

Product description	Value
plate 34 microwell **96** well microtite polystyrene disposable styrene vinyl bottom nonsterile	96
plate 23 assay **96** well vinyl alpha numeric gird round u bottom	96
amplate skirt **384** wells flex plates blue	384
amplate **96** wells flex. plates pp blue	96
plate tissue culture **12** well multidishes and microwell thermo scientic biolite	12
plate tissue culture 330ul **96** well clear polystyrene standard bottom flat bottom nontreated lid sterile	96
plate **48** well 5ml assay microwell polypropylene flat rectangular bottom nonsterile	48
plate assay 0.19ml **96** well clear half area polystyrene untreated flat bottom	96
plate well .2ml **96** u	96
axygen storage microplates	
plate multiwell 100 um clear nonsterile multiscreen mesh	
lid plate assay sterile rigid styrene with short lip for bar code reader without notch	
plate microplate type p4	
v-bottom collection plates	
well plate bioblock blue	
system plate 0.5ml topas with glass flat bottom vial multitier	

The input data are descriptions of products assigned to the "Microplates" category (Table 2). One of the attributes that characterizes the plate is the number of wells. The values of the attribute are integers. In 9 descriptions, the desired fragments, which should be found by the resulting regular expression, are indicated (*positive examples*). In the remaining 7 descriptions, the resulting regular expression should not find anything, as there is no attribute value in the description (*negative examples*).

The regular expression resulting from the procedure described in the "Methods" section is formulated as follows:

```
(\d\d(?:\d)?)\s(?:u|well|wells)
```

The expression used for input descriptions gives an error-free result, i.e. in all descriptions where the desired value to be found is marked, the expression finds the value, while in all descriptions where the value is not marked, the expression returns an empty answer. The obtained result, i.e. the effective regular expression, initially confirms the validity of the authors' approach.

The second experiment is more closely related to a practical situation, i.e. one that is likely to happen in real life.

The input data are descriptions of products assigned to the "Dental drills" category (Table 3). The attribute under consideration is the length of the drill bit measured in millimeters. The attribute can take values that are integers, as well as values expressed as a range with minimum and maximum lengths. The input data consists of 121 examples, 66 *positive examples* and 55 *negative examples.*

Table 3. Input data for experiment 2 – products from the "Dental drills" category

Product description	Value
drill twist l**27**mm od1.9mm without stop nonradiolucent dental end for variax hand locking plate system	27
mini-line angular drill att.dental shank	
drill **28**mm right angle assort gates glidden nti	28
...	...
drill twist 4.2 short tiger	
drill twist 4.70mm 5 sm blue for implant	
drill twist 3.35 **8–13**mm sp	8–13
...	...
drill dental l**25**mm od3.3mm short bone level tapered prole stainless steel	25
drill twist l**10–18**mm od2.4–2.8mm step parallel disposable	10–18
zimmer dental drill stop kit	

In this case the task was so complex that the procedure described in the "Methods" section failed to generate a regular expression that would correctly find the values in all the input descriptions. The best regular expressions, i.e. those with the lowest fitness value are as follows:

```
(?:5mmx|l)(\d(?:\d)?(?:\-|\.)?(?:\d)?(?:\d)?)(?:\s|mm|mml)
(?:\+)?(?:5mmx|\s|l)(\d(?:\-\d|\s)(?:\.|\d)?(?:\d)?)(?:mm|mml)
```

The first one was able to find a value correctly in 35 *positive examples*, whereas in the remaining 31 *positive examples* and in all the 55 *negative examples*, the expression returned an empty value. The second expression correctly found the value in 56 *positive examples*, whereas in 10 *positive examples* and in all the 55 *negative examples* it returned an empty value; in the case of 2 *positive examples*, however, it found a value other than the correct one. The last situation is the most undesirable, as it is less of an error not to find the value than to find an incorrect value.

Conclusions and Future Works

The paper presents the concept of a procedure that allows to generate a regular expression to extract the value of attributes of medical products from their text descriptions. The approach is based on the genetic algorithm and the authors' elaborations in the area of representing the regular expression and assessing its quality, as well as that of the genetic operators of crossing and mutation.

The conducted experiments presented in the paper showed the effectiveness and considerable possibilities of the proposed method. At the same time, the course and results of the experiments highlighted the shortcomings and possible directions for improvement of the proposed approach. One problem to be solved in the future is calculation time. In essence, a genetic algorithm is not a computationally efficient procedure, but some of its elements can be implemented using parallel programming. Another improvement may consist in adapting the procedure to the possibility of generating several outputted regular expressions, each covering only some of *positive examples.*

Acknowledgments

This work was supported by the grant S/WI/2/2018 from Bialystok University of Technology founded by Ministry of Science and Higher Education.

R E F E R E N C E S

Banko, M., Cafarella, M. J., Soderland, S., Broadhead, M., & Etzioni, O. (2007). Open information extraction from the web. In *Proceedings of the Twentieth International Joint Conference on Artificial Intelligence* (pp. 2670–2676). IJCAI, Hyderabad, India.

Chang, C.-H., Kayed, M., Girgis, M. R., & Shaalan, K. F. (2006). A survey of web information extraction systems. *IEEE Transactions on Knowledge and Data Engineering*, *18*(10), 1411–1428.

Forrest, S. (1993). Genetic algorithms – principles of natural selection applied to computation. *Science*, *261*(5123), 872–878.

Ghani, R., Probst, K., Liu, Y., Krema, M., & Fano, A. (2006). Text mining for product attribute extraction. *ACM SIGKDD Explorations Newsletter*, *8*(1), 41–48.

Hazel, P. (2012). *PCRE – Perl-compatible regular expressions (original API)* [Library Functions Manual]. Retrieved from http://pcre.org/pcre.txt

Köpcke, H., Thor, A., Thomas, S., & Rahm, E. (2012). Tailoring entity resolution for matching product offers. In *Proceedings of the 15th International Conference on Extending Database Technology* (pp. 545–550). ACM, Berlin, Germany.

Koza, J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection* (Vol. 1). Cambridge, MA: MIT Press.

Matthews, B. W. (1975). Comparison of the predicted and observed secondary structure of t4 phage lysozyme. *Biochimica et Biophysica Acta (BBA) – Protein Structure*, *405*(2), 442–451.

Petrovski, P., Bryl, V., & Bizer, C. (2014). Learning regular expressions for the extraction of product attributes from e-commerce microdata. *Proceedings of the Second International Conference on Linked Data for Information Extraction*, *1267*, 45–54.

Popescu, A.-M., & Etzioni, O. (2007). Extracting product features and opinions from reviews. In A. Kao & S. R. Poteet (Eds.) *Natural Language Processing and Text Mining* (pp. 9–28). London: Springer.

Sarawagi, S. (2008). Information extraction. *Foundations and Trends in Databases*, *1*(3), 261–377.

Thompson, K. (1968). Programming techniques: Regular expression search algorithm. *Communications of the ACM*, *11*(6), 419–422.

Wu, F., & Weld, D. S. (2010). Open information extraction using Wikipedia. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics* (pp. 118–127). Association for Computational Linguistics, Uppsala, Sweden.