Can A Machine Replace Humans In Building Regular Expressions? A Case Study

Alberto Bartoli, Andrea De Lorenzo, Eric Medvet and Fabiano Tarlao
Machine Learning Lab, University of Trieste, Italy
Publisher version: http://dx.doi.org/10.1109/MIS.2016.46

Abstract

Regular expressions are routinely used in a variety of different application domains. Building a regular expression involves a considerable amount of skill, expertise and creativity. In this work we investigate whether a machine may surrogate these qualities and construct automatically regular expressions for tasks of realistic complexity. We discuss a large scale experiment involving more than 1700 users on 10 challenging tasks. We compared the solutions constructed by these users to those constructed by a tool based on Genetic Programming that we have recently developed and made publicly available. The quality of automatically-constructed solutions turned out to be similar to the quality of those constructed by the most skilled user group; and, the time for automatic construction was similar to the time required by human users.

Introduction

Regular expressions are routinely used in a variety of different application domains and are widely viewed as one of the fundamental tools that should be in a programmer's toolbox. Building a regular expression tailored to a specific problem is often difficult, tricky and time-consuming, though. In March 2016 web site Stack Overflow, the most popular Question & Answer programming forum, features more than 140,000 questions on this topic with "regex" being the 25-th most popular question tag in a set including more than 44,000 tags. Nearly all of the question tags which are more popular than "regex" refer to a specific programming language or library—"arrays" is the only general tag more popular than "regex", while "ajax" and "json" are only slightly more popular than "regex".

There is no doubt that writing a regular expression requires a considerable amount of skill, expertise and creativity by the programmer. In this work we investigate whether a machine may surrogate these qualities and construct automatically regular expressions for tasks of realistic complexity. We address this question based on a large scale experiment involving more than 1700 users on 10 challenging tasks. We asked users to construct a regular expression based on a few examples of the desired behavior and then compared their solutions to those obtained with an automatic tool that we recently developed and described in full detail in earlier works [1,2]. Our tool is based on Genetic Programming. Both the users and the tool were given the very same information: examples of the desired behavior without any hint about the structure of the target expression. We compared the results along two axes: quality of the solution assessed on a hold-out testing set and the time required for constructing the solution.

The quality of automatically-constructed solutions was very similar to the quality of solutions constructed by (self-proclaimed) experienced users; and, the time that our tool took to construct a solution was similar to that of humans performing the same task. The machine was thus able to indeed surrogate expertise and creativity of programmers in a traditionally difficult synthesis activity (see also the sidebar).

Problem Statement

Regular expressions are often used for binary *classifying* strings, depending on whether a string matches or does not match the pattern encoded by the expression. We consider instead *extraction* problems in which it is also required to identify all the substrings matching the specified pattern. Extraction is more general than classification in the sense that a solution to the former is also a solution to the latter, while the opposite is not true—a string could include many instances of the specified pattern; the knowledge that at least one instance of the pattern occurs somewhere in the string may not help very much in actually locating all those instances.

To specify the problem we need a few definitions. A regular expression applied on a string *s* deterministically extracts zero or more substrings from *s*, that we call *extractions*. The problem input consists of a set of *examples*, where an example is a string *s* coupled with a (possibly empty) set X_s of non-overlapping substrings of *s*. Set X_s represents the *desired extractions* from *s*, i.e., all the substrings in X_s are to be extracted whereas any other substring of *s* is not to be extracted. We do not make any assumptions on either the length or the internal structure of string *s*, which may be a text line, or an email message, or a log file, and so on. In practice, substrings in X_s may be specified easily by annotating portions of *s* with a GUI (see next section).

The problem consists of learning a regular expression $\widehat{r}$ whose extraction behavior is consistent with the provided examples: for each example, $\widehat{r}$ should extract from each string *s* all and only the desired extractions X_s. Furthermore, $\widehat{r}$ should capture the pattern describing the extractions, thereby generalizing beyond the provided examples. In other words, the examples constitute an incomplete specification of the extraction behavior of an ideal and unknown regular expression $r^\star$. The learning algorithm should infer the extraction behavior of $r^\star$.

Our Tool

Our tool is available as a live web app[1] and in source code on GitHub[2]. Internally it is based on Genetic Programming (GP) and described in full detail in [1,2]. Space precludes a complete description, hence we provide only a brief outline. We evolve a population of 500 regular expressions, represented by abstract syntax trees, by applying classical genetic operators such as mutation and crossover for 1000 iterations. We generate the initial population partly at random and partly based on the desired extractions, i.e., for each desired extraction *x* we generate 4 different regular expressions with a deterministic heuristics ensuring that all these expressions extract *x*. We drive evolution by means of a multiobjective optimization algorithm based on the length of regular expressions (to be minimized) and their extraction performance computed on the learning data (to be maximized). We use a separate-and-conquer heuristics for discovering automatically whether the extraction task may be solved by a single regular expression or whether a set *R* of multiple regular expressions, to be eventually joined by an “or” operator, is required [2]. In particular, every 200 iterations we check whether the currently best regular expression r_i exhibits perfect precision on a subset *X* of the desired extractions. In that case, we remove *X* from the set of desired extractions, we insert r_i in *R* and we let the search continue (*R* is initially empty). Finally, we join all the elements in *R* and the best regular expression upon the end of the search by an “or” operator.

A screenshot of the web app is given in Figure 1. The user may load examples as UTF-8 files and then annotate text in these files graphically to identify desired extractions. The number of examples is irrelevant; what matters is the number of desired extractions: 10–20 usually suffice to

[1] http://regex.inginf.units.it/
[2] https://github.com/MaLeLabTs/RegexGenerator

obtain good solutions. We used 24 in the experiment described below. Examples and the resulting expressions may be saved for later analysis and reuse.

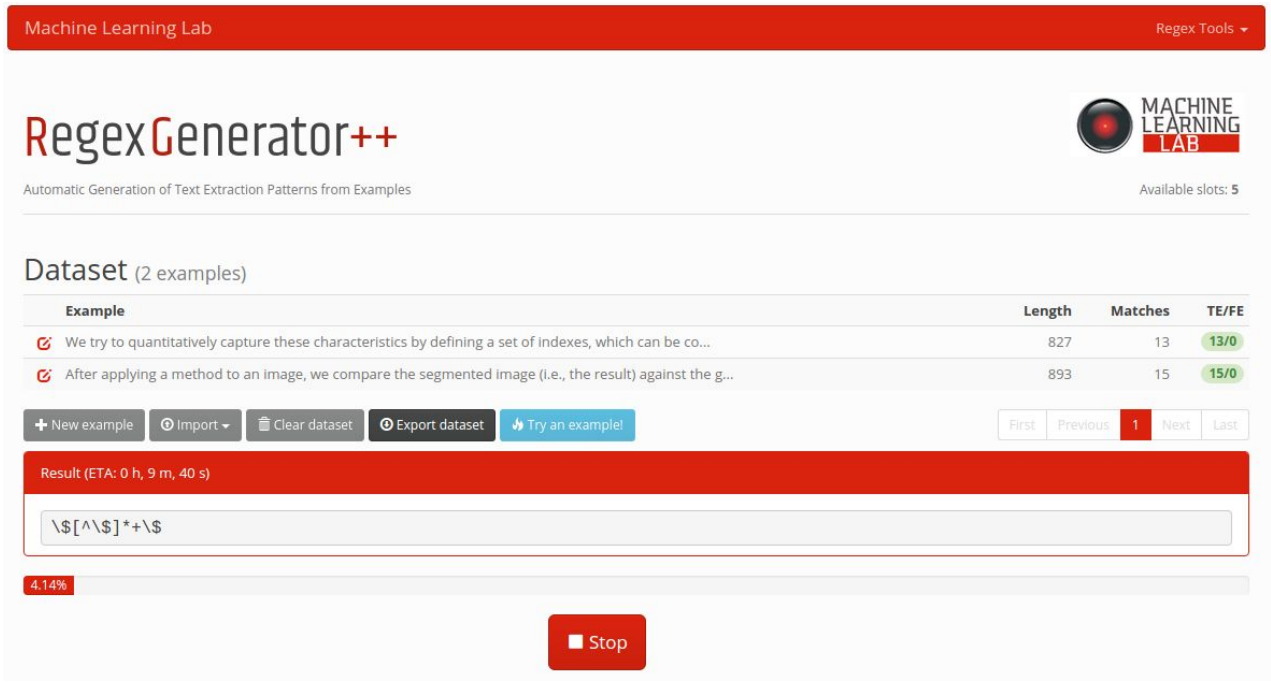

Figure 1: *Snapshot of our tool taken during a search. The tool shows the best solution currently found.*

The Challenge Platform

For our experiment, we developed a *challenge web app* for assisting human operators in the task of developing a regular expression for text extraction based on examples of the desired behavior[3]. The challenge web app starts by presenting concise instructions (*"write a regular expression for extracting text portions which follow a pattern specified by examples"*) and asks the user to indicate his/her perceived level of familiarity with regular expressions: novice, intermediate, or experienced. Then, the challenge web app proposes a sequence of *extraction tasks*. Each task is presented as a text area in which the substrings to be extracted are highlighted.

The user writes a regular expression in a dedicated input field and the challenge web app highlights, with negligible latency, the substrings extracted by the expression along with the corresponding extraction mistakes. An example is in Figure 2. The user may refine the regular expression interactively, that is, he may modify the expression at will and obtain an immediate feedback about the modified expression. We emphasize that the interactive nature of the challenge web app should make it easier for human operators to solve the proposed tasks, both in terms of quality of solutions and time required for their construction.

The challenge web app also shows the F-measure on the current task. To avoid the need of understanding what the F-measure actually represents, the user is informed that a value of 100% means a perfect score on the task. The user is not required to obtain a perfect F-measure before going to the next task and could even leave a task completely unanswered. Furthermore, the user need not execute all the tasks in a single session: when connecting, the challenge web app presents to the user the extraction task he was working on when disconnecting. The challenge web app records, for each task and for each user, the authored regular expression and the overall time spent on the task, excluding disconnection intervals.

In practice, users craft regular expressions in many ways. They may describe them using natural language, examples of matching strings, or with a combination of both. Users' descriptions may

[3] The web app is available at http://play.inginf.units.it/

be underspecified, in the sense that they do not specify how every possible input sequence should be classified, and their descriptions can be refined during several iterations. The challenge web app specifies an extraction tasks solely by means of examples. This is necessarily an approximation of user behavior, but it nevertheless preserves the essence of the problem of constructing a regular expression, and it is simple for users to understand. The annotations can be done quickly, which is important because the challenge web app presented examples already annotated but a user willing to use our tool instead of crafting a regex would have to annotate examples. In the experiments described in the next section, we considered extraction tasks specified with 24 desired extractions; annotating the corresponding data took 1.5–2.5 minutes, depending on the task.

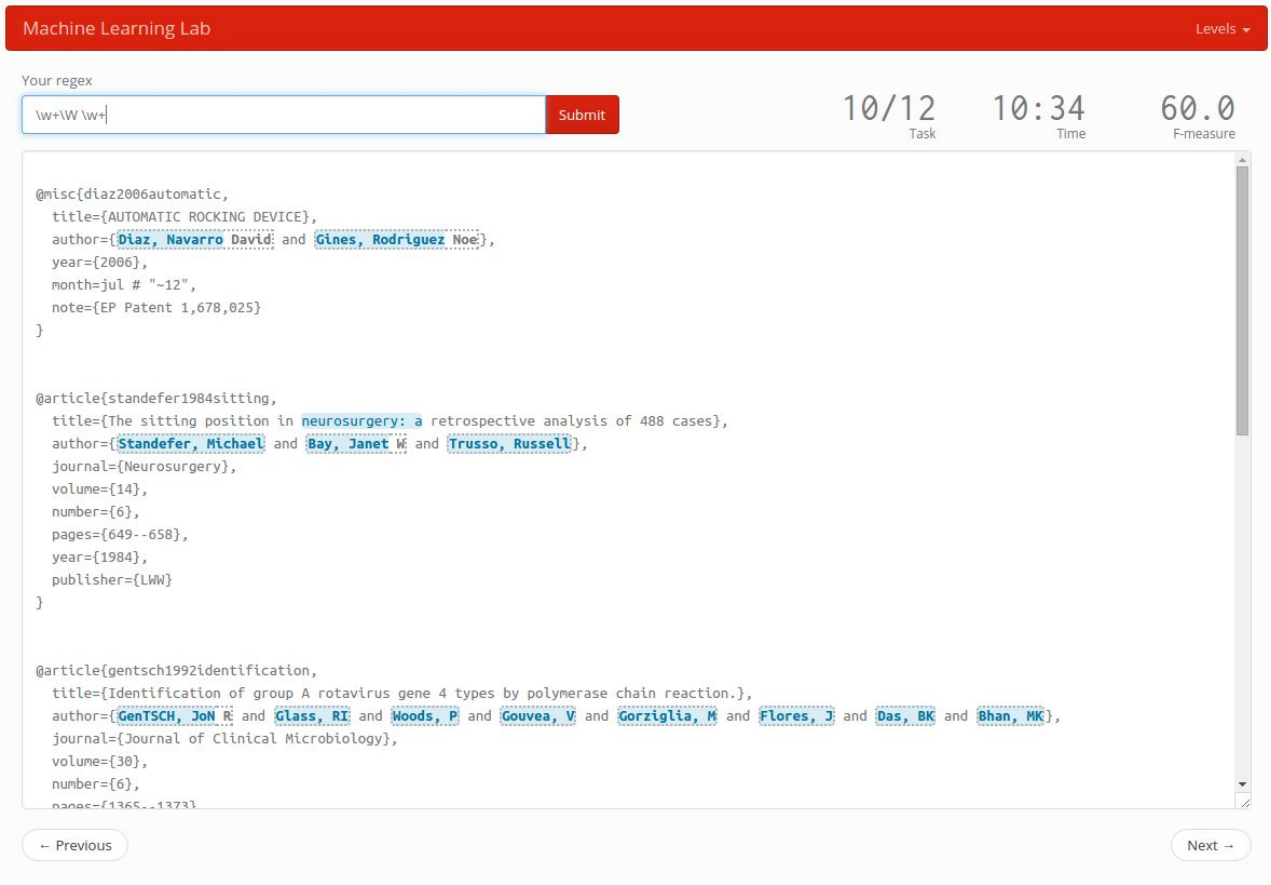

Figure 2 *Snapshot of the challenge web app presented to users. The user has inserted the regex "*`\w+\W \w+`*" and the webapp highlights the extractions of this regex in blue: it can be seen that this regex results in undue extractions (i.e., highlighted text outside of the dashed boxes) and missed extractions (i.e., non highlighted text within the dashed boxes).*

Procedure

We constructed 10 challenging extraction tasks, summarized in Table 1. Task names consist of the corpus name followed by the name of the entity type to be extracted:

- ReLIE-HTML: portions of a subset of the 50,000 web pages obtained from the publicly available University of Michigan Web page collection (used also in [1,3]).
- ReLIE-Email: portions of the 10,000 emails obtained from the publicly available Enron email collection (used also in [3,4]).
- Cetinkaya-HTML: full HTML source of 3 web pages (used also in [1,5]).
- Cetinkaya-Text: plain text of the above web pages after rendering (used also in [1,5]).
- Log: log entries collected from our lab firewall (used also in [1]).
- Web: full HTML source of a richer collection of web pages than Cetinkaya.
- BibTeX: BibTeX elements obtained by querying Google Scholar.
- References: references in the Springer LNCS format obtained from the BibteX corpus.

Task name	Number of characters (x 10^3)	Number of desired extractions
ReLIE-HTML/All-URL	4,240	502
Click here to access index history <http://www.intox.com/SubscriberServlet/subscriberservlet.class?operation=powerIndexForm&hub=All>. * volume represents sell-side only * Hub High Low Wtd Avg Index Change ($) Vol (Mwh) Ciner		
ReLIE-Email/Phone-Number	4,240	499
3784 SSWB (734) 763-6276 ddavies@umich.edu </td> abs Client Services Center at:</FONT><TD align=middle><FONT face=Arial size=2> 734/936-2598 (Local), 800/862-7284 (Michigan Only) or 800/537-7284 (Outside Michigan) </FONT> </TD>		
Cetinkaya-HTML/HREF	154	214
Project Gutenberg Scitation		
Cetinkaya-Text/All-URL	39	168
Fedora Extras http ftp rsync ftp://ftp7.br.FreeBSD.org/pub/FreeBSD/ (ftp) ftp://ftp3.de.FreeBSD.org/pub/FreeBSD/ (ftp) ftp://ftp.is.FreeBSD.org/pub/FreeBSD/ (ftp / rsync)		
Log/IP	4,126	75,958
Jan 13 05:49:47: ACCEPT service dns from 74.125.189.23 to firewall(pub-nic-dns), prefix: "none" (in: eth0 74.125.189.23(00:80:38:fa:8a:7e):51027 -> 140.105.63.158(00:00:76:fe:75:e2):53 UDP len:80 ttl:49)		
Log/MAC	4,126	38812
Jan 13 17:44:52: DROP service 68->67(udp) from 172.45.240.237 to 217.70.177.60, prefix: "spoof iana-0/8" (in: eth0 216.34.90.16(00:21:91:fe:a2:6f):68 -> 69.43.85.253(00:07:e1:7c:53:db):67 UDP len:328 ttl:64)		
Web-HTML/Heading	4,541	1,083
e se non fosse che 'n sul passo d'Arno [modifica] Infrastrutture e trasporti Visite Libero.HF.adjust800 = function () {		
Web-HTML/Heading-Content	4,541	1,083
e se non fosse che 'n sul passo d'Arno <h2>[<a		

href="/w/index.php?title=Torino&action=edit&section=47" title="Modifica la sezione Infrastrutture e trasporti">modifica] Infrastrutture e trasporti</h2> <h5>Visite</h5> Libero.HF.adjust800 = function () {		
Bibtex/Author	54	589
@inproceedings{arellano2004study, title={Study of the structure changes caused by earthquakes in Chile applying the lineament analysis to the Aster (Terra) satellite data.}, author={Arellano-Baeza, A and Zverev, A and Malinnikov, V}, booktitle={35th COSPAR Scientific Assembly},		
References/Lead-Author	30	198
130. Andrews, D.G., Holton, J.R., Leovy, C.B.: Middle atmosphere dynamics. Number 40. Academic press (1987)		

Table 1. *Extraction tasks. A short snippet with each desired extraction highlighted in green outlines the nature and difficulty of each task. Note that the snippet of Web-HTML/Heading contains two desired extractions that are adjacent but separate.*

For each task, we randomly selected a set of examples containing 24 desired extractions (note that this corresponds, for each task, to a very small portion of the full corpus) and embedded the corresponding set in the web app. We published a post on Reddit encouraging users to challenge themselves[4]. Next, we executed our tool by using the very same set of examples as the learning set. We repeated each execution four times and averaged the performance indexes (see next section).

We chose not to distribute different sets of examples to different users because we did not expect to receive thousands of submissions and in a preliminary experiment we observed that many tasks were left unanswered. We thought that presenting different data to different users might have not allowed collecting a meaningful set of results. We have assessed the performance of our tool also with different learning sets, by executing a 5-fold procedure on each task. The resulting slight difference in the actual values of the indexes was negligible. We included two simple tasks at the beginning of the task sequence aimed solely at allowing users to practice and familiarize with the web app interface. We did not include these tasks in the analysis (their results are qualitatively similar to those of the other tasks, though).

Results

We gathered results from a large population: 1,764 users participating from July 23-rd 2015 to September 20-th 2015. These users qualified themselves as follows: 44% novice, 38% intermediate, and 18% experienced. Users completed 10,439 out of the 17,640 tasks. Novice users completed 52% of the tasks, intermediate users 61%, and experienced users 71%.

We analyze results along two axes: quality of the solution assessed with F-measure and the time required for constructing the solution. We report average values for each category of users by taking into account only completed tasks with construction time between percentiles 1% and 99% (Figure 3). Execution times for our tool have been obtained on a 6-core Intel Xeon 2.4 GHz with 32 GB RAM.

The key finding is that, on average, our tool delivered solutions with F-measure almost always greater than or equal to the one obtained by each category of human users, both on the learning data and on the testing data. Furthermore, on average, the time required by our tool was almost always smaller than the time required by human operators. We believe these results are

[4] https://www.reddit.com/r/programming/comments/3eblji/how_good_are_you_in_writing_regex_challange/

remarkable and highly significant. Indeed, we are not aware of any similar tool exhibiting such human-competitive performance indexes.

By looking at the actual distributions of F-measure[5], one may always find a significant fraction of humans which obtain better results than our tool. In other words, while our tool is not systematically better than humans, it does deliver F-measure that is comparable to humans and that, on average, is even better. Actual distributions of construction time indicate that our tool tends to be systematically faster than most humans on most tasks. This indication is also statistically significant.

The only task in which our tool delivers unsatisfactory F-measure on the testing data, despite a very good value on training data, is ReLIE-Email/Phone-Number. A closer inspection of the dataset shows that, for this task, the training data happens not to be adequately representative of the testing data, in particular, concerning substrings that look like phone numbers but are not. Executing our tool on a larger training set result in F-measure around 85%.

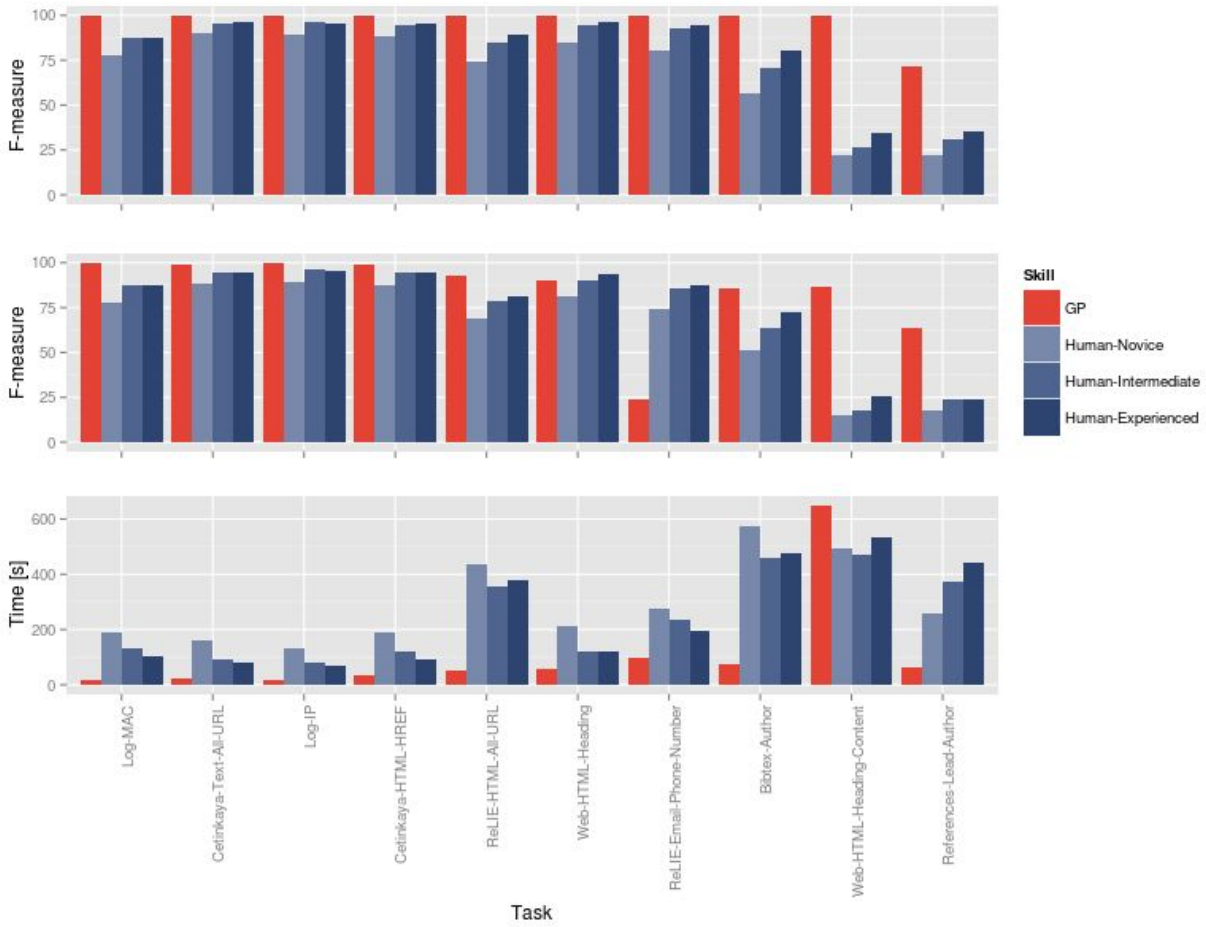

***Figure 3**: F-measure on the learning set (upper chart), F-measure on the remaining part of each dataset, i.e., on a hold-out testing set (middle), and the construction time (lower chart).*

Concluding remarks

While we do not claim that a tool like ours may be effective in each and every possible application of regular expressions, we do believe to have provided strong indications that a machine may indeed constitute a practically viable tool for synthesizing regular expressions from scratch. In our challenging tasks, the machine has proven its ability to surrogate the expertise and skills required

[5] Space constraints preclude a more detailed statistical analysis, that can be found at http://machinelearning.inginf.units.it/data-and-tools/can-a-machine-replace-humans-in-building-regular-expressions-a-case-study.

by human programmers. We believe that this result is relevant in itself and, more broadly, as a further demonstration of the practical capabilities of Genetic Programming techniques even on commodity hardware.

An issue that we have not yet addressed is *readability* of the solutions. While this property is orthogonal to F-measure, it may nevertheless be important in practice: users might not trust a result that they do not fully understand or whose behavior in corner cases might be difficult to predict. As an aside, these remarks apply also to other popular machine learning paradigms, e.g., neural networks. Manual inspection of a few solutions suggest that human operators tend to construct shorter solutions, but we could not find any clear cut between the categories: even automatically-constructed solutions may be very compact and highly readable; and, there is ample variability between operators with task difficulty playing a key role.

We plan to assess readability of the solutions as part of a broader investigation on this important question: what are the key differences between solutions constructed by human programmers and automatically-constructed solutions? Is it possible to distill such differences—for example including readability—into a fitness definition capable of driving the evolutionary search toward regions of the solution space closer to those explored by human operators? We believe that ideas of this kind may provide an exciting line of research in evolutionary computing.

Acknowledgements

The authors are grateful to reviewers for their numerous constructive comments.

References

1. A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet and E. Sorio, “Automatic Synthesis of Regular Expressions from Examples.” *Computer* 47 (12): pp. 72–80, Dec. 2014.
2. A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, “Learning text patterns using separate-and-conquer genetic programming,” in *18th European Conference on Genetic Programming*, pp. 16-27, Apr. 2015.
3. Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish, “Regular expression learning for information extraction,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing. 2008*, pp. 21–30.
4. F. Brauer, R. Rieger, A. Mocan, and W. M. Barczynski, “Enabling information extraction by inference of regular expressions from sample entities,” in *Proceedings of the ACM Conference on Information and Knowledge Management*. 2011, pp.1285–1294.
5. A. Cetinkaya, “Regular expression generation through grammatical evolution,” in Proceedings of the *ACM Conference on Genetic and Evolutionary Computation*. 2007, pp.2643–2646.

Author biographies

Alberto Bartoli is an associate professor of Computer Engineering at the University of Trieste, Italy. His research interests include machine learning applications, evolutionary computing and security. He received a PhD in Computer Engineering in 1993 from the University of Pisa, Italy. bartoli.alberto@univ.trieste.it

Andrea De Lorenzo is a research associate at the University of Trieste, Italy. His research interests include evolutionary computing, computer vision and machine learning applications. He received a PhD in Computer Engineering in 2014 from the University of Trieste. mimmuz2k5@gmail.com

Eric Medvet is an assistant professor at the University of Trieste, Italy. His research interests include web and mobile security, genetic programming and machine learning applications. He received a PhD in Computer Engineering in 2008 from the University of Trieste. emedvet@units.it

Fabiano Tarlao is a PhD student at the University of Trieste, Italy. His research interests include web and mobile security, genetic programming and machine learning applications. He received a degree in Electrical Engineering in 2010 from the University of Trieste. ftarlao@gmail.com

Related Work Sidebar

The problem of synthesizing a regular expression automatically, based solely on examples of the desired behavior, has attracted considerable interest, for a long time and from different research communities.

An important line of research considered *classification* problems in *formal* languages [1,2,3]. These works aimed at inferring an acceptor for a regular language based on sample strings described by the language and on sample strings not described by the language. Learning of *deterministic finite automata (DFA)* from examples was also a very active area [4,5,6]. Such research, however, usually considered problems that were not inspired by any real world application [5] and the applicability of the corresponding learning algorithms to other application domains is unexplored [6]. The problem setting typical in this field of research considered short sequences of binary symbols, with training data drawn uniformly from the input space. Settings of this sort do not fit the needs of practical text processing applications, which have to cope with much longer sequences of symbols, from a much larger alphabet, not drawn uniformly from the space of all possible sequences.

Entity extraction on realistic business- or web-related data has been considered for improving a regular expression to be initially provided by the user [7,8,9], as well as for inferring an expression fully from scratch [10,11,12]. Our proposal cited in the main text falls in the latter category and advances significantly over those approaches, in terms of improved quality of the solutions and smaller amount of training data required. Indeed, we are not aware of any other approach that could use human operators as a baseline. Approaches tailored to very specific domains have also been proposed, e.g., [13,14].

An approach for optimizing expressions constructed by expert developers was recently proposed in [15], consisting of a loop in which the behavior of candidate solutions is assessed in *crowdsourcing* followed by an evolutionary optimization of the best solutions found so far. This approach is aimed at investigating the possibility of crowdsourcing difficult programming tasks specified by examples of desired behavior. The experiment that we discuss in this work considers more complex extraction tasks on much larger datasets and analyzes solutions constructed in a fully automatic way.

Finally, we mention recent proposals for automating submissions to regex writing challenges consisting in writing the shortest regular expression that matches all strings in a given list and does not match any string in another given list [16,17]. Such proposals aim at merely *overfitting* examples without inferring any general pattern.

Sidebar references

1. A. Brazma, “Efficient identification of regular expressions from representative examples,” in *Proceedings of the Sixth ACM Conference on Computational Learning Theory*, 1993, pp. 236–242.

2. F. Denis, “Learning regular languages from simple positive examples,” *Machine Learning*, vol. 44, no. 1-2, 2001, pp. 37–66.
3. H. Fernau, “Algorithms for learning regular expressions from positive data,” *Information and Computation*, vol. 207, no. 4, 2009, pp. 521 – 541.
4. K. J. Lang, B. A. Pearlmutter, and R. A. Price, “Results of the abbadingo one DFA learning competition and a new evidence-driven state merging algorithm,” in *Grammatical Inference*. Springer, 1998, p. 1–12.
5. O. Cicchello and S. C. Kremer, “Inducing grammars from sparse data sets: a survey of algorithms and results,” *The Journal of Machine Learning Research*, vol. 4, 2003, pp. 603–632.
6. J. Bongard and H. Lipson, “Active coevolutionary learning of deterministic finite automata,” *The Journal of Machine Learning Research*, vol. 6, 2005, pp. 1651–1678.
7. Y. Li, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Jagadish, “Regular expression learning for information extraction,” in *Proceedings of the Conference on Empirical Methods in Natural Language Processing*. 2008, pp. 21–30.
8. R. Babbar and N. Singh, “Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text,” in *Proceedings of the Fourth ACM Workshop on Analytics for Noisy Unstructured Text Data*, 2010, pp. 43–50.
9. K. Murthy, D. P., and P. Deshpande, “Improving recall of regular expressions for information extraction,” in *Web Information Systems Engineering - WISE 2012*, ser. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2012, vol. 7651, pp. 455–467.
10. A. Cetinkaya, “Regular expression generation through grammatical evolution,” in Proceedings of the *ACM Conference on Genetic and Evolutionary Computation*. 2007, pp.2643–2646.
11. F. Brauer, R. Rieger, A. Mocan, and W. M. Barczynski, “Enabling information extraction by inference of regular expressions from sample entities,” in *Proceedings of the ACM Conference on Information and Knowledge Management*. 2011, pp.1285–1294.
12. D. F. Barrero, M. D. R-Moreno, and D. Camacho, “Adapting searchy to extract data using evolved wrappers,” *Expert Systems with Applications*, vol. 39, no. 3, 2012, pp. 3061–3070.
13. P. Prasse, C. Sawade, N. Landwehr, and T. Scheffer, “Learning to identify regular expressions that describe email campaigns,” in *Proceedings of the International Conference on Machine Learning*, 2012.
14. D. D. A. Bui and Q. Zeng-Treitler, “Learning regular expressions for clinical text classification,” *Journal of the American Medical Informatics Association*, vol. 21, no. 5, 2014, pp. 850–857.
15. R. A. Cochran, L. D’Antoni, B. Livshits, D. Molnar, and M. Veanes, “Program boosting: Program synthesis via crowd-sourcing,” in *Proceedings of the 42nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, 2015, pp. 677–688.
16. P. Norvig, “xkcd 1313: Regex golf,” http://nbviewer.ipython.org/url/norvig.com/ipython/xkcd1313.ipynb, Jan. 2014.
17. A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, “Playing regex golf with genetic programming,” in Proceedings of the *ACM Conference on Genetic and Evolutionary Computation*, 2014, pp. 1063–1070.