Active Learning of Regular Expressions for Entity Extraction

Alberto Bartoli, Andrea De Lorenzo, Eric Medvet, and Fabiano Tarlao

***Abstract*—We consider the automatic synthesis of an entity extractor, in the form of a *regular expression*, from *examples* of the desired extractions in an unstructured text stream. This is a long-standing problem for which many different approaches have been proposed, which all require the preliminary construction of a large dataset fully annotated by the user. In this work we propose an *active learning* approach aimed at minimizing the user annotation effort: the user annotates only *one* desired extraction and then merely answers extraction queries generated by the system. During the learning process, the system digs into the input text for selecting the most appropriate extraction query to be submitted to the user in order to improve the current extractor. We construct candidate solutions with *Genetic Programming* and select queries with a form of querying-by-committee, i.e., based on a measure of disagreement within the best candidate solutions. All the components of our system are carefully tailored to the peculiarities of active learning with Genetic Programming and of entity extraction from unstructured text. We evaluate our proposal in depth, on a number of challenging datasets and based on a realistic estimate of the user effort involved in answering each single query. The results demonstrate high accuracy with significant savings in terms of computational effort, annotated characters and execution time over a state-of-the-art baseline.**

***Index Terms*—Inference mechanisms, Semisupervised learning, Evolutionary computation, Genetic programming, Text processing, Man machine systems, Automatic programming**

I. Introduction

The problem of synthesizing string processing procedures automatically, based solely on examples of the desired behavior, is the focus of several recent proposals [1], [2], [3], [4], [5], [6]. Low-level operations of string transformation, substring extraction, format validation and alike occur routinely at organizations of all sizes and usually have to be addressed by business developers, i.e., by people whose primary job function is not application development [7]. Therefore, tools capable of implementing the desired transformations automatically would be a great help in a myriad of different contexts.

In this work we focus on the problem of extracting from an *unstructured text stream* entities that adhere to a syntactic pattern. In particular, we consider the automatic synthesis of an extractor, internally consisting of a *regular expression*, based on *examples* of the desired extractions—a long-standing problem for which many approaches have been proposed [8], [9], [10], [11], [12], [13], [14].

We propose an approach based on *active learning* [15], [16]. The user initially marks *only one* desired extraction in the input text and then interactively answers *extraction queries* generated by the system, by merely annotating on a graphical interface which portions of a query are, or are not, to be extracted. The system generates tentative extractors automatically, attempting to improve the quality of each extractor upon each answer based on a careful trade-off between overfitting the provided examples and capturing their general pattern.

The resulting framework is highly attractive because it may minimize the user annotation effort required for constructing meaningful examples: it is the system, not the user, that digs into the potentially very large input text for identifying those text snippets whose annotation may indeed be useful; most importantly, the system is potentially in a better position than the user for identifying which extraction queries are most useful to the learning procedure.

On the other hand, actually exploiting the potential of active learning is highly challenging. First, the system must be able to generate high quality solutions with less than few tens of queries, in order to be competitive with the state-of-the-art in *passive learning*, i.e., methods in which all annotations must be constructed in advance by the user [17]. This is an extremely challenging baseline. Second, the number of candidate queries grows quadratically with the size of the dataset and becomes huge very quickly—e.g., even if we assume that the learner cannot generate queries ex novo, in a dataset with just 10^5 characters there are $\approx 10^{10}$ substrings, each of which may be a candidate query. Third, a wealth of active learning proposals choose queries for improving the current solution based on *uncertainty* of that solution [18], [19], [20]. Approaches of this kind are not suitable in our case: a regular expression does not provide any kind of uncertainty about the handling of a given substring (i.e., extracted vs. not extracted).

The core of our proposal is a *solver* module, which learns a regular expression from the currently available annotations. This solver is internally based on *Genetic Programming (GP)* [21] . We chose a GP-based approach due to recent results which have advanced significantly over all earlier approaches, demonstrating the ability of GP to address extraction tasks of practical complexity effectively [17], [22], [23]. The cited proposals, however, assume a scenario radically different from active learning as they require tens of extractions in a *fully annotated* dataset available from the *very beginning* of the learning process. We consider instead a scenario in which the dataset is fully unannotated except for only one desired extraction. The user will specify further desired extractions by answering extraction queries constructed by the system. Our proposed solver is based on the approach in [17], that we carefully and substantially redesigned in order to address

All the authors are with the Department of Engineering and Architecture (DIA), University of Trieste, Italy.
E-mail: bartoli.alberto@units.it.

Manuscript received …

the peculiar needs of active learning. These include (a) the need of constructing tentative solutions even with very few examples of the desired extractions available, e.g., even with just 1 or 2: in a scenario of this sort, the inherent tension between overfitting and generalization cannot be addressed in the same way as in a learner which may exploit tens of examples; (b) the need of coping with an unstructured text stream that is only partially annotated, i.e., in which the boundaries of desired/undesired extractions may be unknown; (c) the need of using an evolutionary search strategy capable of exploiting a dynamically growing set of fitness cases, i.e., one in which candidate solutions which excel in the few more recently acquired fitness cases are indeed able to compete with solutions which have been improving for many generations; and, finally, (d) the need of having short time intervals between queries.

Another key component of our proposal is the *query builder*, which constructs candidate queries and select the most useful query to be submitted to the user. We remark that most active learning approaches need not tackle the query construction task as they assume a one-to-one correspondence between input instances and candidate queries. Frameworks of this sort are not a good fit for information extraction problems like ours: when an input instance is a large document, asking the user to annotate which portions of a document have and have not to be extracted may nullify the crucial objective of minimizing user annotation effort. Input units are thus to be segmented in smaller queryable units with a suitable trade-off between informativeness and annotation efficiency [24]. Our proposed query builder need not assume any native segmentation or separation token within input text streams, thereby making the method truly suitable for unstructured text.

A further important and peculiar aspect of our work is that we devoted special care in modelling the user annotation effort in order to obtain a realistic assessment of the solution accuracy/user effort regions of the design space. In most active learning frameworks queries are assumed to require all the same effort, but it has been shown that such an assumption is often inaccurate and, consequently, may lead to misleading conclusions about the real effectiveness of active learning strategies [25]. Such a consideration is particularly relevant in our scenario: an active learning policy that tends to generate short queries that mostly require a single mouse click to be answered cannot have the same cost as a policy that tends to generate longer queries that mostly need some editing by the user. For these reasons, we executed several sessions with human operators actually annotating substrings for extraction and measured several indexes of their activity. Then, we built a user effort model based on those measurements. The resulting model allows quantifying accurately the annotation cost of each single query, based on the length of the query and the number of mouse operations required for constructing the corresponding answer.

In summary, our contribution is as follows:

- We propose an active learning system that addresses the long-standing problem of automatic construction of regular expressions effectively.
- We analyze the trade-off between user annotation effort and solution quality based on an accurate model of the former, constructed from real measurements.
- We present one of the few successful applications of Genetic Programming in an active learning scenario—Genetic Programming is mostly applied in scenarios in which all the learning data is available before starting the evolutionary search.

II. Related work

A. Active learning

Active learning is a principled, broad framework for automatically choosing, from a collection of unlabelled data, data to be labelled by a human operator. The main focus consists in minimizing user annotation cost and, to this end, the learning process is integrated with the query choice procedure in the attempt of selecting those queries that are most useful at any given learning step. In particular, an active learning system (i) constructs a model based on the labeled data currently available and (ii) selects from a collection of unlabeled data the queries to be submitted to the user. The system iterates the previous steps, thereby refining the learned model on a larger set of labelled data, until a specified termination criterion is satisfied, often expressed in terms of maximum number of queries that can be submitted to the user. An essential component of any active learning framework consists of the query strategy, i.e., step ii. In this respect, the most commonly used approach is based on *uncertainty*, that is, the query submitted to the user corresponds to the unlabeled data for which the current model is the least certain how to label. Another common approach is *query-by-committee*, which requires a committee of competing models all based on the same labelled data. In this case the query corresponds to the data about which the current models most disagree. There have been many proposals for actually implementing these approaches, depending for example on the criteria used for quantifying uncertainty and disagreement, as well as many proposals for different query strategy approaches. A comprehensive survey can be found in [16].

The active learning framework is advocated across a broad range of different application domains and recent examples include, e.g., entity filtering in Twitter-like text streams [26], multiclass classification in text streams [27], query and document ranking [28], keyword search-based data integration [29]. In the next sections we focus on the works closer to ours.

B. Automatic generation of regular expressions

A form of active learning for automatic generation of regular expressions for entity extraction is explored in [30]. The cited work proposes heuristics for synthesizing regular expressions from examples and focuses on improving recall of those expressions while retaining high precision. At each active learning step, generalization heuristics derive several candidate expressions and estimate the precision of each candidate. A batch of 10 queries is then chosen as a sample of the substrings that would be extracted by the most promising candidate. The criterion for selecting the sample is not specified. The user labels each query as either correct or wrong and the procedure continues until exhausting the query budget. Our approach is radically different from several points of view. We generate candidate solutions with a GP-based solver; we do not select queries as a mere sample of the extractions of the best (estimated) candidate, rather, we use a principled approach (a form of querying-by-committee) which attempts to identify the query that is *most useful* and that is so for a *set* of competing candidates; we assess user effort based on the annotation cost involved in each single query rather than by merely counting the number of queries. Furthermore, the cited work submits 10 queries at a time whereas many active learning approaches, including ours, choose to submit instead one single query at each round. Selecting a set of queries that are both highly informative and not redundant requires algorithms explicitly designed for *batch mode* active learning [31], [32]. Merely increasing the number of queries to be submitted at each round may result in higher user effort without significant improvement in solution quality. Finally, our extraction tasks are arguably more challenging than those considered in [30].

In our preliminary work on active learning of regular expressions [33] we outlined a system similar to the one described and assessed in this work. The cited work, though, used a solver simpler than the one described in this work and that was not described in detail: in particular, the present work solver includes, w.r.t. [33], a more sophisticated fitness, a mechanism for enforcing behavioral diversity in the population, a population size which can grow during the evolution, and a different way of initializing individuals from the examples (see Section IV-B). Furthermore, the experimental evaluation was much more limited and based on an annotation effort measured simply with the amount of annotated characters. In this work we provide an assessment that is much deeper and is based on detailed models for the annotation effort constructed from real observations, which allow us to carefully estimate the time required for answering each single query as well as for fully annotating a dataset.

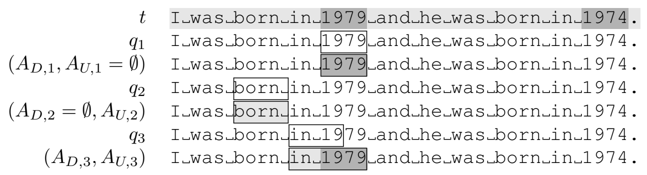
C. Genetic Programming

Active learning with Genetic Programming is proposed in [34]. The cited work considers the problem of *record deduplication*, i.e., identification of different records in a database which actually refer the same entity. The proposed approach evolves a population of binary classifiers which take a pair of records, along with several attributes, and decide whether the pair actually refers the same entity. The constructed solution consists of an ensemble of predefined size, including the classifiers with best performance on user-labelled data. A query consists of a pair of records that the user may classify as being either the same record or two different records. The set of candidate queries Q is constructed before starting the learning procedure as a subset of all record pairs, selected based on an unspecified method. Initially, Q is ranked for estimated similarity between pairs, based on a similarity function not detailed; record pairs that are estimated to be the most similar and the least similar are then queried to the user before starting the evolution. At each generation all candidate queries are tentatively labelled by the committee and those which cause a tie are submitted to the user. A predefined upper limit on the maximum number of queries that can be submitted at each generation is enforced; the criterion for choosing which queries to not submit, in case the budget is exceeded, is not detailed. Mechanisms for updating weights of classifiers in the ensemble and the ranking of candidate queries are an integral part of the design (see the cited paper for details).

Our approach is also a form a querying-by-committee, with several important differences. Each individual in our population is a candidate solution; we randomly choose one single example before starting the evolution; we do not allow query generation at (potentially) each generation; we do not construct a statically defined subset of candidate queries in advance—indeed, as previously observed, the number of candidate queries in our application domains is typically very large; thus, whenever our query trigger suspends our solver, our query builder constructs only a subset of candidate queries chosen dynamically based on the committee composition and behavior. Furthermore, as observed for [30], we do not allow for submitting multiple queries at once and we do not estimate user effort with the mere number of queries. It may be useful to also observe that our baseline is not just a GP-based solution for the same problem: it is the state-of-the-art solution, which happens to be GP-based.

Active learning with Genetic Programming has been applied also for *linkage rules*, i.e., the problem of matching entities from two different data sources which refer the same object [35]—a classification problem that shares several similarities with record deduplication considered in [34]. Query ranking is based on several forms of querying-by-committee: each individual represents a different classifier and the input unit for which disagreement in a set of good-performing individuals (the *committee*) is maximal is submitted to the user. Several different methods for quantifying disagreement and for selecting the composition of the committee are proposed and assessed. Queries are submitted one at a time every 50 generations of the evolutionary search. Solution quality is highly competitive with respect to earlier not GP-based approaches while saving annotation effort (in terms of number of labelled instances). Similar remarks can be made for [36].

Another paradigm called active learning has been proposed for Genetic Programming [37], but with a meaning quite different from the one usually assumed in the literature (as observed by [34]). The cited proposal aimed at sampling a large amount of *already labelled* data in order to increase efficiency of the learning process while not affecting quality of the outcome, whereas the essential focus in the literature consists in efficiently selecting samples of *unlabelled* data for labelling. In this respect, it may be useful to mention active learning of regular expressions as proposed in [9]. In this case, active learning consists of a procedure in which the user selects one example, the system constructs a “contextual extraction rule” which generalizes the example for selecting a subset of the unannotated data and the user is then required to annotate those data. The key point is that the procedure is to be executed *before* actually starting the learning process, that is, the learner



t	`I was born in 1979 and he was born in 1974.`
q_1	`I was born in 1979 and he was born in 1974.`
$(A_{D,1}, A_{U,1} = \emptyset)$	`I was born in 1979 and he was born in 1974.`
q_2	`I was born in 1979 and he was born in 1974.`
$(A_{D,2} = \emptyset, A_{U,2})$	`I was born in 1979 and he was born in 1974.`
q_3	`I was born in 1979 and he was born in 1974.`
$(A_{D,3}, A_{U,3})$	`I was born in 1979 and he was born in 1974.`

Fig. 1: Example of user-system interaction (see the text).

operates on a statically defined training set without any further interaction with the user. As an aside, the cited proposal may learn only a *reduced* form of regular expressions, including only the most basic constructs—e.g., quantifiers for specifying that a preceding token may occur zero or more times are not supported. Furthermore, it learns regular expressions defined over a predefined small dictionary of words and part-of-speech tags. Our proposal is based on [17], which supports most of the constructs available in modern regular expression engines, including quantifiers, non-capturing groups and lookarounds, and learns expressions defined over UNICODE characters.

III. The proposed system

A. Interaction model

We consider a scenario in which the user wants a regular expression tailored to the extraction of entities of interest. To this end, the user operates a system which constructs the regular expression based only on *examples* of the desired extraction behavior. Such examples are *annotations* on a potentially large text t, i.e., substrings of t that are to be extracted or substrings of t that are not to be extracted. The user does not provide the annotations all at once before using the system: it is the system which solicits the user to annotate specific substrings of t, as follows.

At the beginning, the user annotates just one desired extraction in t and starts the system. Then, the system occasionally asks the user to annotate a specific substring q of t (q is a *query*). The answer to a query q consists of a pair (A_D, A_U), where A_D is the set of all desired extractions which overlap q while A_U is the set of all maximal substrings of q which are not desired extractions (either of two sets may be empty, but not both). Figure 1 shows key examples of a query and the corresponding answer: the first row shows the input text t, where darker background indicate the desired extractions. Query q_1 corresponds to a desired extraction, thus it will be $A_{D,1} = \{q_1\}$ and $A_{U,1} = \emptyset$; q_2 corresponds to an undesired extraction, hence $A_{D,1} = \emptyset$, $A_{U,2} = \{q_2\}$. Finally, q_3 overlaps a desired extraction only in part, in this case neither $A_{D,3}$ nor $A_{U,3}$ will be empty: note in the figure that the only element in the former extends outside of the box while the only element in the latter does not. Of course, the case exemplified by q_3 may generalize to cases in which A_D and A_U contain more than one element each.

In other words, in order to accommodate the case when a desired extraction overlaps but does not coincide with the query, we propose an interaction model which allows the user to modify the received query slightly (i.e., by extending its boundaries on one or both sides) and then answer the modified query. In contrast, in most active learning approaches the user is required to provide only the class of queried data and is not allowed to modify those data.

We developed a web-based prototype with a GUI that supports the proposed interaction model efficiently and that in our experience has proven intuitive even to unskilled users. Each query is shown as a highlighted portion of text t and the user is presented with 3 buttons, one for each of the three cases in Figure 1. When the query coincides exactly with a desired extraction (i.e., q_1) or does not contain any desired extraction (q_2), then one single click suffices to answer the query (button "Extract" or "Do not extract", respectively). Otherwise, when the user has to describe a more complex answer (q_3), by clicking an "Edit" button the user may extend the selection boundaries of the query and delimit desired extractions precisely.

B. System architecture

We propose a system composed of three modules which interact according to the following workflow (Figure 2). The system takes a potentially large text t as input, along with the annotation of one desired extraction in t. (1) A *solver* module based on Genetic Programming evolves a population of candidate solutions, i.e., of regular expressions (Section IV-B); the evolution is driven by the currently available annotations on the input text t and can be paused by a *query trigger* module. (2) A *query builder* module selects a substring from t that is not annotated and asks the user to annotate this substring, as illustrated in the previous section. (3) The user answers the annotation query. The system iterates these steps until a specified termination criterion is satisfied (Section V-C). We experimented with different variants for both the query trigger and the query builder (Section IV-C and in Section IV-D, respectively). Both these modules require some knowledge of the internal state of the solver module.

We remark that most applications of active learning need not a query trigger module because they are based on solvers whose synthesis is relatively fast (e.g., a Random Forest or a Support Vector Machine). On the other hand, a full solver execution on all the available data and after each query is clearly impractical in our scenario, because a GP-based execution may take a long time. The policy embedded in the query trigger may influence effectiveness of active learning significantly, because it impacts the trade-offs between user effort and solution quality.

The solver module is based on the proposal in [17]. The cited proposal requires a *fully annotated* dataset *before* starting the learning procedure, i.e., it assumes a *passive learning* scenario. We outline the passive learning solver in Section IV-A for providing the necessary context (space constraints preclude a full description, which can be found on the cited work). Then, we describe our proposed active learning solver in Section IV-B, emphasizing the key differences between constructing regular expressions with passive learning and constructing them with active learning.

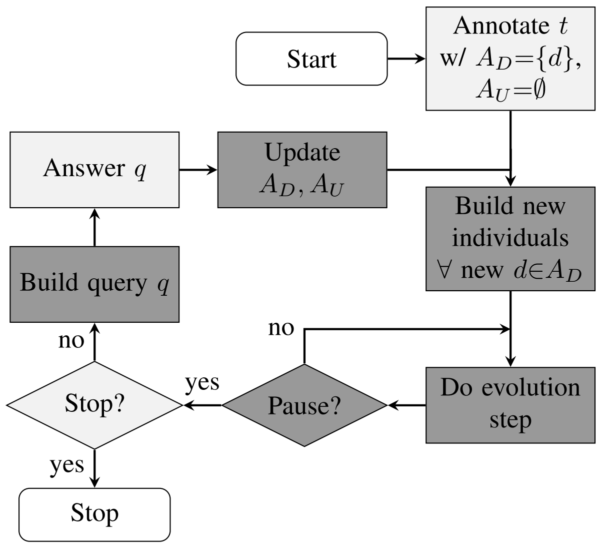

Start
Annotate t w/ A_D=$\{d\}$, A_U=$\emptyset$
Answer q
Update A_D, A_U
Build new individuals $\forall$ new $d \in A_D$
Build query q
no
no
yes
Stop?
Pause?
Do evolution step
yes
Stop

Fig. 2: Workflow of our system. Block background indicates the actor of the corresponding step: light gray for the user, dark gray for the system.

IV. System modules

A. Passive Learning Solver

The passive learning solver, that we call *PL-solver*, constitutes the state-of-the-art for passive learning and is publicly available as webapp (http://regex.inginf.units.it) and in source code form (https://github.com/MaLeLabTs/RegexGenerator).

PL-solver is implemented with tree-based Genetic Programming [21]. Each candidate solution (*individual*) represents a regular expression as an abstract syntax tree. Each node in a tree has a label, which is a string representing basic components of a regular expressions. The regular expression represented by a tree is the string constructed with a depth-first post-order visit. Available labels include most regular expression constructs: character classes (\d, \w), predefined ranges (a-z, A-Z), digits (0, ..., 9), and predefined characters (\., :, ,, ;, _, =, ", ′, \\, /, \?, \!, \}, \{, \(, \), \[, \], <, >, @, #, ␣) constitute the *terminal set*, i.e., labels that can be used for leaf nodes; concatenator (●●), set of possible/not possible matches ([●]), possessive quantifiers (●*+, ●++, ●?+, ●{●,●}+), non-capturing group ((?:●)), and lookarounds ((?<=●), (?<!●), (?=●), (?!●)) constitute the *function set*, i.e., labels that can be used for non-leaf nodes—the character ● represents the string representation of a child. The terminal set includes also labels that are constructed automatically based on the examples of the specific text extraction problem. For instance, in the task of generating a regular expression for extracting FTP addresses, the string ftp could be a useful such block. Recent applications of Genetic Programming have demonstrated the power of this paradigm for evolving candidate solutions represented as a tree conformant to a context-free grammar [38]. Indeed, the representation used by PL-solver is based on a similar idea, tailored to the specific requirements of regular expressions.

PL-solver evolves a *population* of 500 individuals by applying classical genetic operators such as mutation and crossover for 1000 iterations (*generations*): whenever an individual is generated which does not correspond to a valid regular expression, it is discarded and a new one is generated. A multiobjective optimization algorithm drives the evolution of individuals according to their extraction performance computed on the fully annotated dataset (to be maximized) and to their length (to be minimized). The initial population is generated partly at random and partly based on the annotated desired extractions, i.e., for each extraction x the initial population includes 4 different regular expressions constructed with a deterministic heuristics ensuring that all the expressions extract x.

The population is of constant size n_p and its composition follows an *elitism* principle, as follows. At each iteration, n_p new individuals are constructed: 80% with crossover of the current population, 10% with mutation of the current population and 10% constructed at random (selection of individuals for application of genetic operators is done with a tournament of size 7); the resulting $2n_p$ individuals are then ranked and the best n_p individuals will constitute the population of the next generation. We remark that PL-solver does not implement any form of local search for attempting to improve candidate solutions that are particularly promising. Strategies of this sort have proven to be very useful in other application domains [39] and certainly deserve further investigation also in this context.

B. Active Learning Solver

We carefully tailored PL-solver to suit the specific needs of active learning. We describe all these modifications below (we outlined a preliminary design in our earlier work [33]).

PL-solver assumes a fully annotated dataset, thus candidate solutions which extract strings that should not be extracted, or which extract superstrings of desired extractions, may be penalized in fitness evaluations. Implementing this principle in active learning is much more difficult because the dataset is only *partially* annotated and annotations usually account for a small fraction of the full dataset. For this reason, we associate different fitness indexes than in [17] with each candidate solution:

1) F-measure at the level of full desired extractions (rather than precision at the level of individual characters).
2) Sum of false positive rate and false negative rate at the level of individual characters. This index is the same as in [17] with an important difference: a start (end) character of an annotated extraction is counted as extracted correctly only if the actual extraction starts (ends) at that character.
3) Length of the candidate solution.

We rank candidate solutions based on the Pareto frontier they belong to and then establish a total order among solutions in the same Pareto frontier based on a lexicographic ordering among fitness indexes (as in [17]).

PL-solver has available the same amount of annotations during the full learning procedure, while in active learning the amount of annotations increases during execution. It follows that, in active learning, candidate solutions which perform relatively well on the annotations accumulated during the early phases of the search, may not experience an evolutionary pressure sufficient enough to accommodate the annotations acquired more recently. For this reason, we introduced a *behavioral diversity enforcement* during the population construction at each generation. We group all candidate solutions that perform the same set of extractions on the full dataset; we rank all candidate solutions and insert in the new population the best solution from each group; we iterate the choice across all groups, ensuring that at most 20% of the new population belongs to the same group. We remark that we do not insist on enforcing genotypic diversity and that we apply the same criterion at each generation. A different approach which adaptively increases genetic diversity when necessary is proposed in [40] with reference to a differential evolution scenario. Indeed, diversity of population is a crucial element in evolutionary algorithms and a broad range of widely differing approaches are possible (see also the cited work).

PL-solver executes several independent evolutionary searches, each producing a candidate solution. The final solution is the one with best F-measure on the full annotated dataset. Furthermore, the annotated dataset is split in two parts and evolutionary searches use only one of such parts for fitness evaluations. In this work we consider a setting that, ideally, should be interactive or near-interactive: execution of several independent evolutionary searches after each query is likely to result to unacceptably long waiting times. Furthermore, searches should be able to construct meaningful candidate solutions even with very few annotations. For these reasons, we chose to execute a *single* evolutionary search and use all available annotations for fitness evaluations.

PL-solver uses a population of constant size initialized with randomly-generated individuals and with individuals constructed based on the annotated dataset—4 different individuals for each desired extraction, as described above. While such a strategy proved very effective in a passive learning scenario, our earlier experiments with active learning made it clear that this strategy promotes an exploitation/exploration trade-off excessively unbalanced toward the latter. Based on this consideration, the solver in this work promotes more exploitation of the available annotated data: whenever a new desired extraction d is available (i.e., after a query has been answered), the solver constructs 14 different individuals from d, by means of a deterministic heuristics ensuring that all these individuals extract d[1]; all these individuals are then inserted in the population. Population size thus grows with each new annotated extraction and no randomly-generated individual is ever inserted. Population size is still bounded to a maximum of $n_p = 500$ individuals.

PL-solver incorporates a form of *separate and conquer* (S&C) procedure [41], [42], [43] which allows coping with extraction tasks in which learning a single pattern capable of describing all the entities to be extracted may be exceedingly difficult—e.g., dates may be expressed in a myriad of different formats [44]. This procedure consists of an heuristics for discovering automatically whether the extraction task may be solved by a single regular expression or rather a set R of multiple regular expressions, to be eventually joined by an "or" operator, is required. In this work we have to execute a solver even with very few annotated desired extractions available, e.g., only 1 or 2: in these cases determining whether a single pattern or multiple patterns are required is clearly unfeasible, unless one assumes explicit indications of this sort from the user. We thus chose to remove S&C and force the generation of a *single pattern* for capturing all annotations.

[1]The corresponding details, that we omit for space reasons, can be found online: http://machinelearning.inginf.units.it/data-and-tools/active-learning-of-regular-expressions-for-entity-extraction.

C. Query trigger

The query trigger determines when the solver has to be paused for submitting a query to the user. We considered two variants.

The *Const* variant pauses the solver when a predefined number of generations of the solver has been executed: we experimented with 30 and 200 generations. This simple approach has been used in other active learning proposals for Genetic Programming [34], [36], [35].

The *Solved* variant pauses the solver whenever at least one of the two following conditions is met: (i) the best regular expression in the population correctly processes all the currently available annotations, i.e., its first fitness index is exactly 1; (ii) the best regular expression in the population has remained unchanged for a predefined amount of generations n_{solved}. In other words, a new annotation is requested to the user either when no further progress seems to be achievable with the available annotations, or when the problem, as posed by available annotations, may be considered as solved. We experimented with $n_{\text{solved}} = 200$ generations, i.e., one of the values selected for the *Const* variant, in order to assess the accuracy/speed trade-off of the two variants.

D. Query builder

The query builder is the module which actually constructs the query and we considered two variants for this module.

The *SmartRand* variant chooses an unannotated substring of t at random, but we place an upper bound to the maximum length of the query that may be generated. We set the actual bound value to the maximum size of a desired extraction across all our datasets (few hundreds characters). Placing an upper bound causes this query builder to filter out candidate queries which are very long, which significantly advantages this query builder w.r.t. one which selects a truly random substring of t—it is for this reason that we qualify this query builder as *Smart*Rand.

The *Query by restricted committee* (rQbC) variant works as follows. Given a set C of regular expressions (the *committee*), we define as *disagreement* of C on a character c of the input text t the quantity $d_C(c) = 1 - 2\text{abs}\left(\frac{1}{2} - \frac{|C_c|}{|C|}\right)$, where $C_c \subseteq C$ is the subset of regular expressions which extract c. It follows that $d_C(c) = 1$ if half of the committee extracts c (maximum disagreement), while $d_C(c) = 0$ if the entire committee agrees on the processing of c (minimum disagreement). Note that we quantify disagreement based on the class chosen by each candidate solution in C (extracted vs. not extracted) [45] without any reference to forms of confidence value, margin or probability [18], [46]. As we pointed out already in the introduction, such notions are not made available by the solver that we have chosen to use. rQbC constructs the query q according to the following procedure:

1) Select the best 25% of the current population as committee C.
2) Determine the character $c^* \in t$ with maximal disagreement $d_C(c^*)$ in the full input set t.
3) Determine set S as the set composed of all substrings of t which meet the following conditions: they (a) are extracted by at least a regular expression in C, (b) overlap c^*, and (c) do not overlap any available annotation.
4) Compute, for each substring in S, the average disagreement among the characters of the substring.
5) Choose as query q the substring with minimum average disagreement.

rQbC is based on a principle widely used in active learning [20], [16], i.e., on the assumption that the query for which an ensemble of competing hypotheses exhibits maximal disagreement is the most informative for the learning task [47]. Such a principle has been used also in active learning for Genetic Programming [34], [36], [35].

As pointed out in the introduction, though, implementing this principle in our application domain requires solving the preliminary problem of actually constructing candidate queries. In this respect, note that a query builder could merely construct single-character queries and select the one with maximal disagreement in the committee (i.e., terminate the procedure at step 2). We designed instead a query builder which attempts to carefully balance between maximizing potential informativeness of the query for the evolutionary search and minimizing the corresponding user annotation effort. Indeed, the rationale for steps 3a and 3b is that extractions are assumed to be more informative than unextractions; for step 3c is preventing the elicitation of the same information multiple times. Finally, for step 5, the choice of minimum instead of maximum average disagreement is motivated by the need to avoid selecting extremely short substrings as query—the corner case being the substring consisting only of c^*.

To further motivate our query builder design, we observe what follows:

- The proposal in [35] takes into account a measure of diversity between each candidate query and queries already answered. Indeed, such a principle has proven fruitful in several active learning contexts (e.g., [28]) as well as in imbalance classification problems (which resembles our problem in the sense that candidate queries are much more likely to be unextractions than extractions) [48]. Our preliminary exploration of this additional principle, that we do not illustrate for space reasons, has not delivered satisfactory results. We believe the reason consists in the difficulty of finding a diversity measure for substrings which is correlated with diversity between regular expressions—e.g., two substrings could be very different while at the same time they could be captured by the same regular expression or by regular expressions that are very similar.
- A wealth of active learning approaches choose queries based on *uncertainty* of the current solution, especially when the learner is not based on an ensemble of competing hypotheses [18], [19], [20], [26], [49]. Such approaches do not fit our solver: candidate solutions are regular expressions and regular expressions do not provide any confidence level about the handling of a given substring (i.e., extracted vs. not extracted). Similar remarks can be made to *self-learning* approaches, which iteratively enlarge the set of labeled data in a different way than active learning: by classifying unlabeled data with sets of diverse models [50]. Such approaches choose which unlabeled data to use for modifying the learned models based on the confidence of those models, but such information is not available in our regex-learning scenario.

V. Experimental evaluation

The main objective of active learning is minimizing user effort while trying not to penalize the effectiveness of the learned artifact. In most active learning frameworks queries are assumed to require all the same effort. It has been shown, though, that in several real-world problem domains annotation costs may vary considerably across queries, thus failing to take this fact into account may lead to misleading conclusions about the real effectiveness of active learning strategies [25]. For this reason, we built a model for the annotation effort based on real observations and devoted special care in analyzing how annotation effort and solution effectiveness relate.

A. Model for user annotation effort

As described in Section III-B, a query may be answered with a single click (*binary answers*) or may require some mouse actions (*edit answers*). These two kinds of answers clearly involve different amounts of user effort. Furthermore, the effort required for annotating a query constructed by the system is very different from the effort in fully annotating a large text, as required in passive learning [17]. Based on these considerations, we constructed three different models for the annotation cost tuned on real observations, as follows.

We involved a set of 10 users with varying skills and asked them to annotate a broad variety of datasets on a webapp that we developed for this purpose. We recorded user actions in two different scenarios: (i) when answering queries generated by our system, in order to emulate interaction with a system based on active learning; in this case the webapp implements the user interface of our active learning prototype (Section III-B); and, (ii) when fully annotating a training dataset, in order to emulate the annotation required by a passive learning tool; in this case the webapp exhibits the same user interface as the webapp in [17]. We emulated active learning by generating a mix of queries requiring differing annotation efforts, i.e., including a full desired extraction, or without any desired extraction, or including one or more partially overlapping desired extractions. Based on the recorded actions, we fitted several models for the annotation time and chose those that proved to be more accurate.

The resulting models for the annotation effort E (expressed in seconds) of a query q and its answer (A_D, A_U) are as follows:

$$E_{\text{binary}}(q, A_D, A_U) = 0.02\,\ell(A_D \cup A_U) + 2.0$$
$$E_{\text{edit}}(q, A_D, A_U) = 3.4|A_D| + 0.01\,\ell(A_D \cup A_U) + 3.1$$

where $\ell(A_D \cup A_U)$ is the overall length of the annotated substrings in the answer. Both models exhibit a fixed cost, an additional cost that is linear in the length of the answer and, for edit answers only, a further additional cost linear in the number of desired extractions. We remark that the numerical parameters of the two models are significantly different, in particular, concerning the fixed cost. Furthermore, the contribution of the linear components is not negligible: for example, a query answer of $\ell(A_D \cup A_U) = 20$ characters suffices to account for an additional 20% of the fixed cost in the binary answers; and, for edit answers, one single desired extraction suffices to more than double the fixed cost. It follows that our user effort model is arguably more accurate than one which simply counts the number of queries or that does not take into account the actual operations required for answering a given query. Indeed, we quantify the (estimated) annotation effort of each single query differently.

Concerning passive learning, let $\ell(l)$ be the number of characters in the full learning data l:

$$E_{\text{passive}}(l, A_D) = 0.003\,\ell(l) + 3.4|A_D|$$

In this case the user effort is linear in the size of the learning data and includes a linear component in the number $|A_D|$ of desired extractions.

B. Extraction tasks

We considered 20 challenging extraction tasks, the same which were used in [17]. For each extraction task, we randomly selected a subset of the original dataset containing 100 desired extractions. The name of each extraction task can be seen—along with the size of the input text expressed in number of characters—in Table I: the task name is composed of the name of the dataset followed by the name of the entity type to be extracted (which should be self-explanatory). Names ending with a $*$ suffix indicate extraction tasks with *context*. Intuitively, in these tasks a given sequence of characters occurs at the same time as a desired extraction and as a substring which should not be extracted: e.g., substring `11` is a desired extraction when part of a date (as in `10-11-2013`) and should not be extracted when occurring "alone" (as in `there␣are␣11␣dogs`).

Table I also shows the *desired extraction density* ρ, which is the ratio between the number of characters in desired extractions and the overall length of the dataset. The lower ρ, the fewer desired extractions in a given amount of text. From another point of view, the lower ρ, the less likely to "find" a desired extraction by randomly choosing a substring of the dataset.

Extraction task	Chars	ρ
Bibtex/Author*	8528	0.17
Bibtex/Title*	26 388	0.24
Cetinkaya-HTML/href	14 922	0.50
Cetinkaya-HTML/href-Content*	14 922	0.45
Cetinkaya-Text/All-URL	7573	0.50
CongressBill/Date	125 479	0.01
Email-Headers/Email-To-For*	86 657	0.03
Email-Headers/IP	36 925	0.04
Log/IP	5766	0.23
Log/MAC	10 387	0.16
NoProfit-HTML/Email	4651	0.49
ReLIE-Email/Phone-Num.	18 123	0.07
ReLIE-HTML/All-URL	16 655	0.32
ReLIE-HTML/HTTP-URL	18 450	0.30
References/First-Author*	14 676	0.08
Twitter/All-URL	9537	0.21
Twitter/Hashtag+Citation	5308	0.23
Twitter/Username*	5308	0.17
Web-HTML/Heading	37 678	0.49
Web-HTML/Heading-Content*	36 921	0.48

TABLE I: Salient information about the extraction tasks.

C. Experimental procedure

We assessed the *extraction performance* of the regular expression generated for a given amount of *user annotation effort*—more annotation effort results in more information available for learning. We quantify extraction performance with F-measure (Fm), which is the harmonic mean of *precision* (ratio between the number of extracted substrings which should have been extracted and the number of all the extracted substrings) and *recall* (ratio between the number of extracted substrings which should have been extracted and the number of all substrings which should have been extracted). We compute precision and recall on the full dataset—of course, such indexes cannot be computed in a real deployment. We quantify user annotation effort as the time spent by the user while annotating, according to the model presented in Section V-A.

In order to gain more insights into the results and the corresponding trade-offs, we also computed the *execution time* (the elapsed time minus the annotation effort), the number of *annotated characters* (AC) and the *computational effort* (CE). The latter is an hardware-independent index which quantifies the total number of character evaluations during an execution. By character evaluation we mean the processing of a character by a candidate regular expression. The amount of CE spent at the ith generation is given by the number of individuals existing at that generation multiplied by the number of characters involved in each fitness evaluation at that generation. We executed all our experiments on an Intel Xeon E5-2440 2.40 GHz with 32 GB of RAM.

We assessed the 6 design variants of our system (three query triggers and two query builders) as follows. For each task, we chose a random desired extraction as the only starting annotated substring and, for each variant, we performed an execution with a simulated user—that is, a program which correctly answers system queries taking an annotation effort given by the model in Section V-A. We repeated this procedure 30 times for each task and variant, with 5 different starting extractions and 6 different random seeds. We averaged all the results presented below across the 30 repetitions of each

	Variant	Fm	AC	CE [$\times 10^9$]	Time [s]
All tasks	SmartRand/Const30	0.63	1024	0.7	26.4
	SmartRand/Const200	0.69	1030	4.5	168.1
	SmartRand/Solved	0.69	992	4.2	170.6
	rQbC/Const30	0.60	575	1.8	86.5
	rQbC/Const200	0.69	651	10.4	677.3
	rQbC/Solved	0.69	700	7.8	378.1
	Passive	0.81	4710	26.6	1644.9
Tasks w/ $\rho < 0.1$	SmartRand/Const30	0.46	2144	0.7	21.0
	SmartRand/Const200	0.45	2159	4.1	128.7
	SmartRand/Solved	0.47	2027	3.0	98.9
	rQbC/Const30	0.48	587	4.5	230.2
	rQbC/Const200	0.52	750	24.4	1505.0
	rQbC/Solved	0.56	795	19.4	903.0
	Passive	0.66	10 395	40.1	4438.9

TABLE II: Average F-measure and execution time with each variant, averaged across all tasks (top) and across sparse tasks only ($\rho < 0.1$, bottom).

	Variant	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved
F-measure	SmartR./Const30		$-0.06^{\ddagger}$	$-0.06^{\ddagger}$	0.03	$-0.06^{\ddagger}$	$-0.06^{\ddagger}$
	SmartR./Const200	$0.06^{\ddagger}$		0.00	$0.09^{\ddagger}$	0.00	0.00
	SmartR./Solved	$0.06^{\ddagger}$	0.00		$0.09^{\ddagger}$	0.00	0.00
	rQbC/Const30	-0.03	$-0.09^{\ddagger}$	$-0.09^{\ddagger}$		$-0.09^{\ddagger}$	$-0.09^{\ddagger}$
	rQbC/Const200	$0.06^{\ddagger}$	0.00	0.00	$0.09^{\ddagger}$		0.00
	rQbC/Solved	$0.06^{\ddagger}$	0.00	0.00	$0.09^{\ddagger}$	0.00	
AC	SmartR./Const30		-6	32^{*}	$449^{\ddagger}$	$373^{\ddagger}$	$324^{\ddagger}$
	SmartR./Const200	6		38	$455^{\ddagger}$	$379^{\ddagger}$	$330^{\ddagger}$
	SmartR./Solved	-32^{*}	-38		$417^{\ddagger}$	$341^{\ddagger}$	$292^{\ddagger}$
	rQbC/Const30	$-449^{\ddagger}$	$-455^{\ddagger}$	$-417^{\ddagger}$		$-76^{\dagger}$	$-125^{\ddagger}$
	rQbC/Const200	$-373^{\ddagger}$	$-379^{\ddagger}$	$-341^{\ddagger}$	$74^{\dagger}$		$-49^{\ddagger}$
	rQbC/Solved	$-324^{\ddagger}$	$-330^{\ddagger}$	$-292^{\ddagger}$	$125^{\ddagger}$	$49^{\ddagger}$	
Exec. time [s]	SmartR./Const30		$-142^{\ddagger}$	$-144^{\ddagger}$	$-60^{\ddagger}$	$-651^{\ddagger}$	$-352^{\ddagger}$
	SmartR./Const200	$142^{\ddagger}$		$-3^{\ddagger}$	$82^{\ddagger}$	$-509^{\ddagger}$	$-210^{\ddagger}$
	SmartR./Solved	$144^{\ddagger}$	$3^{\ddagger}$		$84^{\ddagger}$	$-507^{\ddagger}$	$-208^{\ddagger}$
	rQbC/Const30	$60^{\ddagger}$	$-82^{\ddagger}$	$-84^{\ddagger}$		$-591^{\ddagger}$	$-292^{\ddagger}$
	rQbC/Const200	$651^{\ddagger}$	$509^{\ddagger}$	$507^{\ddagger}$	$591^{\ddagger}$		$299^{\ddagger}$
	rQbC/Solved	$352^{\ddagger}$	$210^{\ddagger}$	$208^{\ddagger}$	$292^{\ddagger}$	$-299^{\ddagger}$	

TABLE III: Average differences of Fm, AC and CE of pairs of the proposed variants. For each pair, the statistical significance is shown: *: $p < 0.1$, †: $p < 0.05$, ‡: $p < 0.01$, $p \geq 0.1$ without any subscript.

experiment.

We terminated each execution upon the query for which either (i) a predefined amount of user annotation effort E_{tot} has been spent, or (ii) the F-measure on the full dataset (i.e., not only on the annotated portion) was 1. Although in a real deployment the user cannot quantify F-measure on a yet unannotated input text, we chose to include the latter condition in the termination criterion in order to provide a fair assessment of variants which are able to generate perfect solutions before reaching the predefined time budget. We set a tight user effort budget $E_{\text{tot}} = 60\,\text{s}$.

To place results in perspective, we also executed the state-of-the-art passive learning tool [17] on the same tasks (PL-solver, Section IV-A). In order to perform a meaningful comparison, we executed PL-solver with the same user annotation effort budget as the active learning tool. We estimated the annotation effort for building a fully annotated training set with the model based on real observations described in Section V-A. We constructed the training set by annotating a randomly chosen substring of the dataset such that the corresponding annotation effort was roughly equal to 60 s. We repeated the procedure 5 times for each dataset by varying the annotated substring.

VI. Results

A. Extraction performance

Table II (top) shows, for each active learning variant and for PL-solver, all the performance indexes averaged across all extraction tasks. Table III shows the differences among each pair of active learning variants along with the statistical significance of the comparison according to the Wilcoxon signed-rank test[2]. Based on these results, several important considerations can be made.

Concerning query builders, rQbC and SmartRand deliver essentially the same F-measure: roughly 0.69, obtained with Const200 and Solved query triggers. Query builder rQbC elicits a smaller amount of annotated characters from the user, AC being 30%–50% lower than for SmartRand. On the other hand, rQbC results in a much higher computational effort and execution time. From a different point of view, rQbC and SmartRandom deliver solutions of comparable quality based on a different trade-off in terms of annotated characters vs. execution time.

[2]We omit the values of computational effort CE for simplicity; these values are strongly correlated with execution time, though.

Concerning query triggers, both Const200 and Solved outperform Const30 in terms of F-measure, which in turn takes much shorter execution time. In fact, the computational effort used by Const30 between queries turns out to be too small to achieve satisfactory results. Unless stated otherwise, all the following considerations will focus on Const200 and Solved query triggers and will omit Const30. In general, Solved and Const200 are equivalent in terms of F-measure; however, Solved is faster than Const200 and requires smaller CE: this is due to the fact that the former may save some execution time when a regular expression is found which solves the currently available annotations, a condition which is met in particular in the early stages of the learning when few annotations are available.

All the active learning variants delivers F-measure that is quite high but smaller than that delivered by PL-solver. It is remarkable, though, that active learning improves the other indexes by almost one order of magnitude for SmartRand and by almost three times for rQbC. In detail, active learning elicits a much smaller amount of annotated characters from the user (AC with SmartRand and with rQbC is $\approx 22\%$ and $\approx 12\%$, respectively, of AC with PL-solver), requires a much smaller amount of computational effort ($\approx 12\%$ and $\approx 25\%$) and a much smaller execution time ($\approx 7\%$ and $\approx 23\%$). In other words, while active learning does not reach the same

	F-measure							Execution time [s]						
Task	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved	Passive	SmartRand Const30	SmartRand Const200	SmartRand Solved	rQbC Const30	rQbC Const200	rQbC Solved	Passive
Bibtex/Author*	0.62	0.59	0.59	0.49	**0.69**	0.60	0.79	7.9	43.7	44.3	27.6	182.3	216.9	908.6
Bibtex/Title*	0.21	0.50	**0.53**	0.31	0.43	0.44	0.76	39.4	202.2	264.3	64.5	326.2	261.0	2211.4
Cetinkaya-HTML/href	0.50	**0.80**	0.62	0.26	0.73	0.75	0.91	28.1	83.6	121.8	44.7	143.4	178.9	576.8
Cetinkaya-HTML/href-Content*	0.46	**0.58**	0.51	0.41	0.43	0.51	0.76	33.4	214.8	181.1	30.5	165.0	132.0	637.7
Cetinkaya-Text/All-URL	0.96	**0.95**	0.94	0.83	0.85	**0.95**	0.99	9.4	87.6	21.4	33.0	180.1	49.8	381.4
CongressBill/Date	0.15	0.14	0.18	0.28	0.31	**0.32**	0.21	43.0	262.8	190.1	660.9	4974.79	2223.6	413.3
Email-Headers/Email-To-For*	0.15	0.10	0.11	0.22	**0.27**	0.26	0.58	23.4	156.8	176.4	253.5	1327.4	1559.5	15 785.8
Email-Headers/IP	0.85	0.83	0.85	0.83	0.76	**0.86**	0.81	9.6	59.2	24.7	144.5	790.7	412.6	830.3
Log/IP	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	0.1	0.1	0.1	0.7	4.7	0.2	338.8
Log/MAC	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	1.00	0.3	0.5	0.2	0.4	1.4	0.2	484.5
NoProfit-HTML/Email	0.87	0.88	0.80	0.83	**0.99**	0.86	1.00	13.3	50.2	35.3	26.0	110.39	76.1	345.1
ReLIE-Email/Phone-Num.	0.75	0.72	**0.76**	0.55	0.70	0.72	0.91	21.8	119.7	73.9	53.1	216.7	121.2	726.3
ReLIE-HTML/All-URL	0.65	0.73	0.79	0.52	**0.79**	0.59	0.84	29.4	178.3	93.4	31.7	170.3	156.8	639.6
ReLIE-HTML/HTTP-URL	0.78	**0.79**	**0.79**	0.55	0.69	0.64	0.86	31.1	193.1	107.1	46.9	224.6	86.7	483.4
References/First-Author*	0.39	0.47	0.45	0.53	0.57	**0.65**	0.77	7.4	45.0	29.4	39.1	215.2	198.2	488.6
Twitter/All-URL	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	**1.00**	0.98	0.1	0.1	0.1	0.1	0.1	0.1	363.7
Twitter/Hashtag+Citation	0.86	0.91	0.93	0.85	**0.94**	0.89	0.94	9.9	53.7	15.2	27.1	154.9	28.0	335.0
Twitter/Username*	0.85	**1.00**	0.92	0.98	0.95	0.94	1.00	8.1	34.2	3.7	16.0	62.3	8.0	300.2
Web-HTML/Heading	0.48	**0.59**	0.57	0.39	0.46	0.53	0.82	114.0	704.9	912.4	45.7	439.4	391.1	1345.2
Web-HTML/Heading-Content*	0.14	0.27	**0.42**	0.21	0.24	0.33	0.40	88.9	872.0	1116.5	184.7	3855.5	1460.3	5302.3

TABLE IV: Average F-measure and execution time with each variant on each task. For the F-measure, the highest value among active learning variants is highlighted.

solution accuracy as PL-solver, it does deliver good solution accuracy on challenging tasks based on a radically different design trade-off.

Table IV shows F-measure and execution time at the granularity of individual extraction tasks. It can be seen that for many extraction tasks active learning manages to deliver F-measure that is either very high or very close to PL-solver. Indeed, on 4 on 20 tasks the best active learning variant improves over PL-solver (CongressBill/Date, Email-Headers/IP, Twitter/All-URL, and Web-HTML/Heading-Content*). On the other hand, for some extraction tasks, active learning delivers an F-measure that is probably excessively low for many settings. For these tasks active learning does not offer a useful trade-off and PL-solver is the only viable option.

Figure 3 illustrates the trade-off F-measure vs. execution time (top) and F-measure vs. AC (bottom)—note the logarithmic scale on the horizontal axis. It can be seen that a large number of points representing active learning executions are close to the ideal regions of the performance diagram. Moreover, active learning elicits more than one order of magnitude less annotated characters from the user, spends less than half computational effort and execution time.

Finally, Table V shows the regular expressions obtained with rQbC/Solved in the first repetition for each task (w.r.t. the 30 repetitions we performed in our experimentation). It is fair to say that, in most cases, the shown regular expressions appear to appropriately describe the patterns corresponding to our 20 tasks.

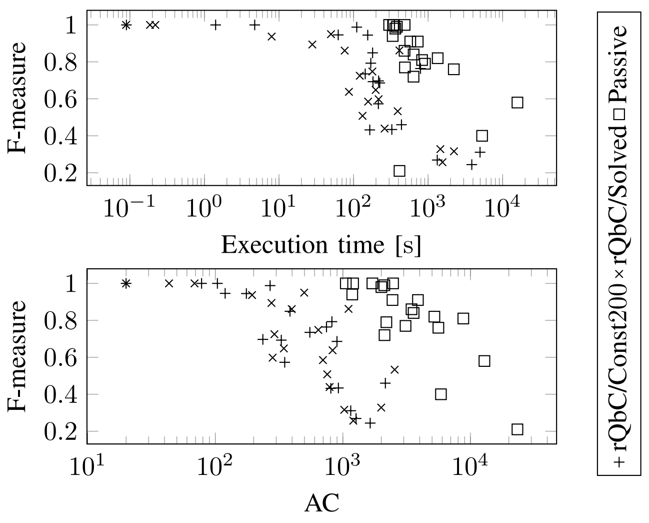


Fig. 3: F-measure vs. execution time (top) or vs. AC (bottom): one point for each task (corresponding to the average index across the repetitions), logarithmic scale on horizontal axis.

B. Querying behavior

In order to gain more insights into the user interactions provoked by the active learning variants, we computed several aggregate indexes from the queries generated during all executions. Table VI reports, for each active learning variant, the proportion of queries with binary answers, the average number $|Q|$ of queries, the average number $|A_D|$ of desired extractions per query, the average number $|\bigcup A_D|$ of desired extractions available at the end of the execution and, finally, the average ratio ρ_q of characters in annotated desired extractions with respect to the total number of annotated characters. For passive learning we provide only indexes $|\bigcup A_D|$, ρ_q, the other indexes being not applicable to such a scenario.

It can be seen that all variants of active learning have available a much smaller amount $|\bigcup A_D|$ of extractions than passive learning. Active learning executions have available

Extraction task	Regular expression
Bibtex/Author*	`(?<=\w{1,5}+␣\w{1,3}+␣)\w++,␣␣\w++(?=[^0]*+)`
Bibtex/Title*	`(?<=\{)[^},]++(?=\},)`
Cetinkaya-HTML/href	`href[^<><␣]*+`
Cetinkaya-HTML/href-Cont.*	`(?<==")[^"]*+`
Cetinkaya-Text/All-URL	`(?:ftp)++[^␣\)3]++`
CongressBill/Date	`\w++␣\w\w,␣\d++`
Email-Headers/Email-To-For*	`(?<=␣)\w++\.\w++\.\w++`
Email-Headers/IP	`\d\d++\.\d++\.[^\];␣]++`
Log/IP	`\w++\.\w++\.\w++\.\w++`
Log/MAC	`\w\w:\w\w:\w\w:\w\w:\w\w:\w\w`
NoProfit-HTML/Email	`[^@]++[^>␣<]++`
References/First-Author*	`(?<=\w{1,3}+\.␣)\w++,␣\w[^:,]*+(?=\w*+)`
ReLIE-Email/Phone-Num.	`\(\w++\)␣\w++\w++`
ReLIE-HTML/All-URL	`(?:http[^␣>]++)++`
ReLIE-HTML/HTTP-URL	`(?:http)[^><;,#␣]*+`
Twitter/All-URL	`\w++:/++\w\.\w++/\w++`
Twitter/Hashtag+Citation	`[@#]\w++`
Twitter/Username*	`(?<=@)\w++`
Web-HTML/Heading	`<[^\]]++[^@]++`
Web-HTML/Heading-Cont.*	`(?<=\w>)(?:[^>@6]++>[^>]++>)++`

TABLE V: Regular expressions obtained with rQbC/Solved: long expressions are split over two lines.

only 5–6 extractions upon *termination* while PL-solver has available 15 extractions for its *full* learning procedure. We believe that this observation is crucial: it seems fair to claim that our active learning setting, in which we start with only one annotated extraction, is extremely challenging; and, that the ability of generating solutions with 0.69 F-measure (on the average) with just 5–6 extractions is indeed remarkable.

Table VI also highlights crucial differences between SmartRand and rQbC: the former tends to consume the annotation effort budget with less queries and tends to generate queries that require more edit operations from the user (smaller values for $|Q|$ and for Binary, respectively). Based on these facts, we may also conclude that SmartRand generates queries that are sufficiently long to include at least part of a desired extraction (indeed, SmartRand exhibits smaller ρ_q than rQbC and comparable $|A_D|$). This observation suggests that SmartRand generates solutions of good quality only because several of our tasks have a large density of desired extractions—sufficiently large to make it very likely that a randomly generated query will partly overlap a desired extraction. This hypothesis is confirmed by the analysis in the next section, in which we focus on those extraction tasks with desired extractions that are a very small portion of the text. Tasks of this sort are indeed practically relevant and highly challenging.

C. *Effectiveness on sparse tasks*

We analyzed more in depth the results on the 5 *sparse* extraction tasks, i.e., those with a density ρ of desired extractions smaller than 0.1 (see Table I): CongressBill-Date, Email-Headers/Email-To-For*, Email-Headers/IP, ReLIE-Email/Phone-Number, and References/First-Author*.

Table II-bottom shows the average F-measure, AC, CE, and execution times, averaged only across those sparse tasks. This data illustrates several key findings. First, there is a much sharper difference between query builders than when considering all tasks. It can be seen that rQbC now clearly outperforms SmartRandom in terms of F-measure. The AC increase is only $\approx 10\%$ for rQbC while it is $\approx 100\%$ for SmartRand. Increase of execution time and CE is instead more pronounced with SmartRand. Second, the F-measure advantage of PL-solver over the best active learning variant (rQbC/Solved) is narrower on sparse tasks than on the full set of tasks: 0.10 vs. 0.12, respectively. Third, the AC and execution time advantage of rQbC/Solved over PL-solver is higher on sparse tasks: ≈ 9600 vs. ≈ 4000 chars and $3500\,\mathrm{s}$ vs. $\approx 1300\,\mathrm{s}$.

In other words, the superiority of rQbC over SmartRand is much more remarkable than it would appear from the average values across all tasks: the former is able to handle sparse extraction tasks effectively, while the latter is not. Furthermore, on sparse tasks, rQbC constitutes an even more viable alternative to passive learning in the trade-off between effectiveness and efficiency.

This finding is further confirmed by Table VI-bottom, which shows the aggregate query indexes computed only over sparse tasks. It can be seen that, for these tasks, the SmartRand query builder simply tends to generate many long queries without any desired extraction (much higher value for the Binary index and slightly smaller values for both $|A_D|$ and $|\bigcup A_D|$). Indeed, the density of extractions in annotated data drops from 0.36 to 0.05. On the other hand, the rQbC query builder tends to generate queries which are qualitatively similar to those over the full set of tasks (equivalent value for the Binary index) and that manage to elicit a slightly larger amount of desired extractions. Most importantly, the density of extractions in annotated data ρ_q exhibits a 75% decrease from 0.47 to 0.14, while the SmartRand exibits a 90% decrease from 0.29 to 0.03.

It is also important pointing out that, for sparse tasks, ρ_q with SmartRand and with passive learning are almost identical (≈ 0.04) while ρ_q with rQbC is much larger (≈ 0.14). These values imply that SmartRand tend to generate learning data which simply reflects the intrinsic density of the desired extractions whereas rQbC is indeed able to dig desired extractions out of the data even when those extractions are sparse. We believe this is one of the most interesting properties of our proposal.

D. *Querying intervals*

An important dimension for an active learning tool is the distribution of time intervals between consecutive queries, whose mean and standard deviation are shown in Table VII.

Concerning reference methods rQbC/Const200 and rQbC/Solved, it can be seen that tasks can be roughly grouped in three categories: one with queries generated every few seconds, another with queries generated every few tens of seconds and a third one (composed of the two most challenging tasks) in which queries are generated every few minutes. While it is not surprising that a near-interactive behavior cannot be supported with all tasks, it is perhaps more surprising that interactive or near-interactive may indeed be

	Variant	Binary	$\|Q\|$	$\|A_D\|$	$\|\bigcup A_D\|$	ρ_q
All tasks	SmartRand/Const30	0.71	10.22	0.47	5.30	0.29
	SmartRand/Const200	0.71	9.95	0.46	5.00	0.29
	SmartRand/Solved	0.71	10.25	0.47	5.24	0.30
	rQbC/Const30	0.89	16.48	0.28	4.90	0.51
	rQbC/Const200	0.89	15.07	0.44	5.04	0.44
	rQbC/Solved	0.85	13.48	0.46	6.30	0.46
	Passive	-	-	-	15.14	0.18
Tasks w/ $\rho < 0.1$	SmartRand/Const30	0.93	9.69	0.42	4.45	0.03
	SmartRand/Const200	0.93	10.19	0.36	4.05	0.03
	SmartRand/Solved	0.92	9.75	0.39	4.21	0.03
	rQbC/Const30	0.89	19.23	0.31	6.23	0.16
	rQbC/Const200	0.89	17.49	0.33	6.17	0.12
	rQbC/Solved	0.81	14.81	0.50	7.96	0.15
	Passive	-	-	-	16.80	0.04

TABLE VI: Aggregate indexes over queries generated during active learning executions, averaged across all tasks (top) and across sparse tasks ($\rho < 0.1$, bottom).

supported with several tasks. Determining what an acceptable inter-query interval is difficult, because such a limit depends on the task difficulty, both real and perceived, as well as on the subjective need of having the task solved and the available alternatives. Overall, it seems fair to claim that an active learning approach with near-interactive behavior for evolutionary solvers is indeed feasible.

Concerning the difference among query triggers, Const200 and Solved are very similar when used with SmartRand (on the average across all tasks, $15.0\,\mathrm{s}$ vs. $14.8\,\mathrm{s}$) while Const200 exhibits a smaller inter-query time interval than Solved when used with rQbC ($42.2\,\mathrm{s}$ vs. $56.7\,\mathrm{s}$, respectively).

We expected inter-query time intervals would be much more uniform with Const200 than with Solved, thereby leading to a better, more predictable behavior from the view of user. Actual data do not result in a clear-cut between the two options, though. Indeed, standard deviation is rather high for all variants which does not lead to a very predictable behavior.

We investigated the feasibility of providing the user with an estimate of the time interval required for generating the next query. Such an indication may mitigate the inconvenience caused by the variability of inter-query time intervals: when the user answers a query, he knows immediately for how long he will have to wait for the next one.

We experimented with a very simple estimator, i.e., we estimate the time interval for generating the next query with the time spent for generating the previous query. As it turns out, such an estimator suffices to provide a very reliable indication, with a Mean Absolute Percentage Error (MAPE) across all of our tasks equal to 9.3% and 8.2%, respectively, for rQbC/Solved and rQbC/Const200. Given such good figures, we chose to not devote further effort in tuning the estimator. Interestingly, the prediction for the Solved query trigger is nearly as good as the prediction for Const200 despite the fact that, with the former, one does not know in advance how many generations will be required before the next query.

E. Impact of solver improvements

We executed a further experimental campaign in order to gain insights into two of our key solver improvements (Section IV-B): new fitness definition and enforcement of behavioral diversity in populations. To this end, we repeated the above suites of experiments in exactly the same configuration, alternately removing each of the two proposed improvements. Table VIII shows the values averaged across all tasks for F-measure, AC, and execution time[3] obtained with: our proposed solver, the proposed solver without enforcement of behavioral diversity, the proposed solver with the fitness used in PL-solver [17] and, finally, with the solver proposed in our earlier active learning work [33].

It can be seen that our new fitness definition and enforcement of behavioral diversity indeed deliver significant improvements in F-measure and thus turn out to be very useful for active learning. One may wonder whether these mechanisms could improve F-measure of PL-solver as well. Although we currently have no elements for answering this question, we speculate that these mechanisms are unlikely to improve the quality of solutions in passive learning because they are meant to address scenarios with very few annotated data and small computational effort, while PL-solver has not been designed for addressing those scenarios .

Concerning AC, we observe that our proposed improvements leave that figure essentially unchanged. On the other hand, Table VIII shows that, not surprisingly, the improved F-measure with unchanged AC is associated with an increased execution time.

VII. Concluding remarks

We have designed, implemented and assessed experimentally an active learning approach for the automatic generation of regular expressions for entity extraction. The approach constructs candidate solutions with Genetic Programming and selects queries with a form of querying-by-committee, i.e., based on a measure of disagreement within the best candidate solutions. All the components are carefully tailored to the peculiarities of active learning with Genetic Programming and of entity extraction from unstructured text. We evaluate our proposal in depth, on a number of challenging datasets and based on a realistic estimate of the annotation effort involved in answering each single query. Our results indicate that the approach is practically feasible, delivering high accuracy in nearly all the extraction tasks analyzed while at the same time ensuring significant savings in terms of computational effort, annotated characters and execution time over a state-of-the-art baseline.

Beyond the specific problem addressed, we also remark that our work is one of the very few successful applications of Genetic Programming based on active learning, that is, in which the labelled data is acquired during the search based on queries constructed by the system rather than being acquired all at once before starting the learning process.

References

[1] R. A. Cochran, L. D'Antoni, B. Livshits, D. Molnar, and M. Veanes, "Program boosting: Program synthesis via crowd-sourcing," in *ACM SIGPLAN Notices*, vol. 50, no. 1. ACM, 2015, pp. 677–688.

[3]In this suite of experiments we did not include the CongressBill/Date task due to its very large execution times.

Task	SmartRand Const30 mean	SmartRand Const30 sd	SmartRand Const200 mean	SmartRand Const200 sd	SmartRand Solved mean	SmartRand Solved sd	rQbC Const30 mean	rQbC Const30 sd	rQbC Const200 mean	rQbC Const200 sd	rQbC Solved mean	rQbC Solved sd
Bibtex/Author*	0.8	0.8	4.6	4.6	4.5	8.9	1.3	3.0	11.1	14.1	20.8	13.7
Bibtex/Title*	3.0	3.0	15.3	17.3	20.2	31.8	3.8	6.9	21.6	18.9	27.7	15.4
Cetinkaya-HTML/href	2.3	2.7	10.5	14.1	10.4	14.6	2.0	4.2	7.9	9.1	14.9	10.4
Cetinkaya-HTML/href-Content*	2.4	2.6	15.9	19.9	13.9	20.4	1.6	3.0	8.4	12.4	13.1	6.8
Cetinkaya-Text/All-URL	1.4	1.3	6.1	6.6	1.6	4.3	1.4	2.2	7.2	8.3	12.1	2.8
CongressBill/Date	6.6	7.1	38.2	31.8	26.4	36.1	42.8	53	343.9	433.4	397.0	159.6
Email-Headers/Email-To-For*	4.3	4.7	23.7	29.7	35.5	56.6	14.3	17.0	72.9	83.5	150.5	104.6
Email-Headers/IP	2.0	2.1	10.7	8.6	5.1	12.6	8.2	12.7	43.8	44.6	65.6	31.8
Log/IP	0.0	0.1	0.0	0.0	0.0	0.0	0.2	0.2	1.0	0.8	0.0	0.1
Log/MAC	0.1	0.1	0.3	0.5	0.0	0.0	0.1	0.1	0.6	0.6	0.0	0.0
NoProfit-HTML/Email	1.1	1.0	4.7	4.3	2.6	4.9	1.3	1.1	7.0	4.8	7.6	4.1
ReLIE-Email/Phone-Num.	1.7	1.9	9.0	14.4	5.5	11.4	2.5	4.7	11.9	15.7	20.8	7.4
ReLIE-HTML/All-URL	2.7	2.4	16.3	16.5	8.3	13	1.8	3.2	11.2	14.5	15.2	9.8
ReLIE-HTML/HTTP-URL	2.8	3.0	16.9	16.2	9.3	15.5	2.9	5.0	17.3	16.3	12.2	6.0
References/First-Author*	0.4	0.3	2.4	1.8	1.6	2.5	1.6	3.7	11.6	14.7	18.2	12.5
Twitter/All-URL	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Twitter/Hashtag+Citation	0.7	0.7	3.6	3.3	1.0	2.3	1.2	1.2	6.5	6.8	7.1	1.8
Twitter/Username*	0.5	0.6	2.6	2.8	0.2	0.7	0.8	0.7	3.8	4.7	3.0	0.5
Web-HTML/Heading	8.4	12.8	58.1	64.2	74.6	129.2	3.8	5.6	38.1	81.5	74.1	36.8
Web-HTML/Heading-Content*	6.1	9.6	61.7	224.7	75.8	186.9	9.8	11.1	219.1	200.4	274.0	114.7

TABLE VII: Mean and standard deviation for the time interval between consecutive queries, in seconds, for each variant/task.

	Variant	Our method	w/o diversity	w/o fitness	w/o improv.
F-measure	SmartRand/Const30	0.66	0.63	0.58	0.60
	SmartRand/Const200	0.72	0.69	0.62	0.65
	SmartRand/Solved	0.71	0.68	0.60	0.64
	rQbC/Const30	0.62	0.62	0.59	0.58
	rQbC/Const200	0.71	0.69	0.61	0.65
	rQbC/Solved	0.71	0.62	0.63	0.69
AC	SmartRand/Const30	916	773	745	686
	SmartRand/Const200	922	723	750	683
	SmartRand/Solved	894	737	740	690
	rQbC/Const30	550	520	687	695
	rQbC/Const200	625	550	708	613
	rQbC/Solved	683	643	707	617
Exec. time [s]	SmartRand/Const30	25.6	22.6	22.8	7.8
	SmartRand/Const200	163.1	122.9	128.0	40.2
	SmartRand/Solved	169.5	83.1	75.6	27.2
	rQbC/Const30	56.3	81.3	51.0	23.2
	rQbC/Const200	451.1	377.1	288.3	127.4
	rQbC/Solved	280.9	291.2	205.6	95.6

TABLE VIII: Average F-measure, AC, and execution time with or without the improvements in the solver.

[2] V. Le and S. Gulwani, "Flashextract: A framework for data extraction by examples," in *ACM SIGPLAN Notices*, vol. 49, no. 6. ACM, 2014, pp. 542–553.

[3] B. Wu, P. Szekely, and C. A. Knoblock, "Minimizing user effort in transforming data by example," in *Proceedings of the 19th international conference on Intelligent User Interfaces*. ACM, 2014, pp. 317–322.

[4] K. Davydov and A. Rostamizadeh, "Smart autofill-harnessing the predictive power of machine learning in google sheets," http://googleresearch.blogspot.it/2014/10/smart-autofill-harnessing-predictive.html, 2014.

[5] S. Gulwani, W. R. Harris, and R. Singh, "Spreadsheet data manipulation using examples," *Communications of the ACM*, vol. 55, no. 8, pp. 97–105, 2012.

[6] P. J. Guo, S. Kandel, J. M. Hellerstein, and J. Heer, "Proactive wrangling: Mixed-initiative end-user programming of data transformation scripts," in *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, 2011, pp. 65–74.

[7] M. Gualtieri, "Empowering the "business developer"," Forrester Research, Tech. Rep., January 2011.

[8] F. Ciravegna *et al.*, "Adaptive information extraction from text by rule induction and generalisation," in *International Joint Conference on Artificial Intelligence*, vol. 17, no. 1. Lawrence Erlbaum Associates LTD, 2001, pp. 1251–1256.

[9] T. Wu and W. M. Pottenger, "A semi-supervised active learning algorithm for information extraction from textual data," *Journal of the American Society for Information Science and Technology*, vol. 56, no. 3, pp. 258–271, 2005.

[10] B. Rozenfeld and R. Feldman, "Self-supervised relation extraction from the web," *Knowledge and Information Systems*, vol. 17, no. 1, pp. 17–33, 2008.

[11] A. Cetinkaya, "Regular expression generation through grammatical evolution," in *Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*. ACM, 2007, pp. 2643–2646.

[12] 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*. Association for Computational Linguistics, 2008, pp. 21–30.

[13] R. Babbar and N. Singh, "Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text," in *Proceedings of the fourth workshop on Analytics for noisy unstructured text data*. ACM, 2010, pp. 43–50.

[14] 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 20th ACM international conference on Information and knowledge management*. ACM, 2011, pp. 1285–1294.

[15] D. Angluin, "Queries and concept learning," *Machine learning*, vol. 2, no. 4, pp. 319–342, 1988.

[16] B. Settles, "Active learning literature survey," *University of Wisconsin, Madison*, vol. 52, no. 55-66, p. 11, 2010.

[17] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, "Inference of regular expressions for text extraction from examples," *Knowledge and Data Engineering, IEEE Transactions on*, vol. 28, no. 5, pp. 1217–1230, 2016.

[18] D. D. Lewis and W. A. Gale, "A sequential algorithm for training text classifiers," in *Proceedings of the 17th annual international ACM SIGIR conference on Research and development in information retrieval*. Springer-Verlag New York, Inc., 1994, pp. 3–12.

[19] T. Scheffer, C. Decomain, and S. Wrobel, "Active hidden markov models for information extraction," in *Advances in Intelligent Data Analysis*. Springer, 2001, pp. 309–318.

[20] F. Olsson, "A literature survey of active machine learning in the context of natural language processing," Swedish Institute of Computer Science, Tech. Rep., 2009.

[21] J. R. Koza, *Genetic programming: on the programming of computers by means of natural selection*. MIT press, 1992, vol. 1.

[22] A. Bartoli, G. Davanzo, A. De Lorenzo, E. Medvet, and E. Sorio,

"Automatic synthesis of regular expressions from examples," *Computer*, no. 12, pp. 72–80, 2014.

[23] A. Bartoli, G. Davanzo, A. De Lorenzo, M. Mauri, E. Medvet, and E. Sorio, "Automatic generation of regular expressions from examples with genetic programming," in *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*. ACM, 2012, pp. 1477–1478.

[24] A. Esuli, D. Marcheggiani, and F. Sebastiani, "Sentence-based active learning strategies for information extraction." in *IIR*, 2010, pp. 41–45.

[25] B. Settles, M. Craven, and L. Friedland, "Active learning with real annotation costs," in *Proceedings of the NIPS workshop on cost-sensitive learning*, 2008, pp. 1–10.

[26] D. Spina, M.-H. Peetz, and M. de Rijke, "Active learning for entity filtering in microblog streams," in *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*. ACM, 2015, pp. 975–978.

[27] Y. Cheng, Z. Chen, L. Liu, J. Wang, A. Agrawal, and A. Choudhary, "Feedback-driven multiclass active learning for data streams," in *Proceedings of the 22nd ACM international conference on Conference on information & knowledge management*. ACM, 2013, pp. 1311–1320.

[28] W. Cai, M. Zhang, and Y. Zhang, "Active learning for ranking with sample density," *Information Retrieval Journal*, vol. 18, no. 2, pp. 123–144, 2015.

[29] Z. Yan, N. Zheng, Z. G. Ives, P. P. Talukdar, and C. Yu, "Active learning in keyword search-based data integration," *The VLDB Journal*, vol. 24, no. 5, pp. 611–631, 2015.

[30] K. Murthy, P. Deepak, and P. M. Deshpande, "Improving recall of regular expressions for information extraction," in *Web Information Systems Engineering-WISE 2012*. Springer, 2012, pp. 455–467.

[31] Y. Guo and D. Schuurmans, "Discriminative batch mode active learning," in *Advances in neural information processing systems*, 2008, pp. 593–600.

[32] Z. Wang and J. Ye, "Querying discriminative and representative samples for batch mode active learning," *ACM Transactions on Knowledge Discovery from Data (TKDD)*, vol. 9, no. 3, p. 17, 2015.

[33] A. Bartoli, A. De Lorenzo, E. Medvet, and F. Tarlao, "Active learning approaches for learning regular expressions with genetic programming," in *31-th ACM Symposium on Applied Computing*, 2016, p. to appear.

[34] J. De Freitas, G. L. Pappa, A. S. Da Silva, M. Gonçalves, E. Moura, A. Veloso, A. H. Laender, M. G. De Carvalho *et al.*, "Active learning genetic programming for record deduplication," in *Evolutionary Computation (CEC), 2010 IEEE Congress on*. IEEE, 2010, pp. 1–8.

[35] R. Isele and C. Bizer, "Active learning of expressive linkage rules using genetic programming," *Web Semantics: Science, Services and Agents on the World Wide Web*, vol. 23, pp. 2–15, 2013.

[36] A.-C. N. Ngomo and K. Lyko, "Eagle: Efficient active learning of link specifications using genetic programming," in *The Semantic Web: Research and Applications*. Springer, 2012, pp. 149–163.

[37] R. Curry, P. Lichodzijewski, and M. I. Heywood, "Scaling genetic programming to large datasets using hierarchical dynamic subset selection," *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, vol. 37, no. 4, pp. 1065–1073, 2007.

[38] J. M. Luna, J. R. Romero, C. Romero, and S. Ventura, "On the use of genetic programming for mining comprehensible rules in subgroup discovery," *Cybernetics, IEEE Transactions on*, vol. 44, no. 12, pp. 2329–2341, 2014.

[39] S. Nguyen, M. Zhang, M. Johnston, and K. C. Tan, "Automatic programming via iterated local search for dynamic job shop scheduling," *Cybernetics, IEEE Transactions on*, vol. 45, no. 1, pp. 1–14, 2015.

[40] M. Yang, C. Li, Z. Cai, and J. Guan, "Differential evolution with auto-enhanced population diversity," *Cybernetics, IEEE Transactions on*, vol. 45, no. 2, pp. 302–315, 2015.

[41] J. Fürnkranz, "Separate-and-conquer rule learning," *Artificial Intelligence Review*, vol. 13, no. 1, pp. 3–54, 1999.

[42] G. L. Pappa and A. A. Freitas, "Evolving rule induction algorithms with multi-objective grammar-based genetic programming," *Knowledge and information systems*, vol. 19, no. 3, pp. 283–309, 2009.

[43] R. C. Barros, M. P. Basgalupp, A. C. de Carvalho, and A. A. Freitas, "A hyper-heuristic evolutionary algorithm for automatically designing decision-tree algorithms," in *Proceedings of the fourteenth international conference on Genetic and evolutionary computation conference*. ACM, 2012, pp. 1237–1244.

[44] 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*. Springer Verlag, 2015.

[45] N. A. H. Mamitsuka, "Query learning strategies using boosting and bagging," in *Machine Learning: Proceedings of the Fifteenth International Conference (ICML'98)*. Morgan Kaufmann Pub, 1998, p. 1.

[46] P. Melville and R. J. Mooney, "Diverse ensembles for active learning," in *Proceedings of the twenty-first international conference on Machine learning*. ACM, 2004, p. 74.

[47] H. S. Seung, M. Opper, and H. Sompolinsky, "Query by committee," in *Proceedings of the fifth annual workshop on Computational learning theory*. ACM, 1992, pp. 287–294.

[48] W. W. Ng, J. Hu, D. S. Yeung, S. Yin, and F. Roli, "Diversified sensitivity-based undersampling for imbalance classification problems," *Cybernetics, IEEE Transactions on*, vol. 45, no. 11, pp. 2402–2412, 2015.

[49] J. Zhang, X. Wu, and V. S. Shengs, "Active learning with imbalanced multiple noisy labeling," *Cybernetics, IEEE Transactions on*, vol. 45, no. 5, pp. 1095–1107, 2015.

[50] I. Triguero, S. García, and F. Herrera, "Seg-ssc: a framework based on synthetic examples generation for self-labeled semi-supervised classification," *Cybernetics, IEEE Transactions on*, vol. 45, no. 4, pp. 622–634, 2015.

Alberto Bartoli received the degree in Electrical Engineering in 1989 cum laude and the PhD degree in Computer Engineering in 1993, both from the University of Pisa, Italy. Since 1998 he is an Associate Professor at the Department of Engineering and Architecture of University of Trieste, Italy, where he is the Director of the Machine Learning Lab. His research interests include machine learning applications, evolutionary computing, and security.

Andrea De Lorenzo received the diploma degree in Computer Engineering cum laude from the University of Trieste, Italy in 2006 and the MS degree in Computer Engineering in 2010. He received the PhD degree in Computer Engineering in 2014, endorsed by the University of Trieste. His research interests include evolutionary computing, computer vision, and machine learning applications.

Eric Medvet received the degree in Electronic Engineering cum laude in 2004 and the PhD degree in Computer Engineering in 2008, both from the University of Trieste, Italy. He is currently an Assistant Professor in Computer Engineering at the Department of Engineering and Architecture of University of Trieste, Italy. His research interests include Genetic Programming, web and mobile security, and machine learning applications.

Fabiano Tarlao received the degree in Electronic Engineering from the University of Trieste, Italy in 2010. He is currently working toward the PhD degree from the Department of Engineering and Architecture at University of Trieste, Italy. His research interests are in the areas of web security, Genetic Programming, and machine learning applications.