EXHIBIT 4

Improving Recall of Regular Expressions for Information Extraction

Karin Murthy, Deepak P., and Prasad M. Deshpande

IBM Research - India, Bangalore, India
{karin.murthy,deepak.s.p,prasdesh}@in.ibm.com

Abstract. Learning or writing regular expressions to identify instances of a specific concept within text documents with a high precision and recall is challenging. It is relatively easy to improve the precision of an initial regular expression by identifying false positives covered and tweaking the expression to avoid the false positives. However, modifying the expression to improve recall is difficult since false negatives can only be identified by manually analyzing all documents, in the absence of any tools to identify the missing instances. We focus on partially automating the discovery of missing instances by soliciting minimal user feedback. We present a technique to identify good generalizations of a regular expression that have improved recall while retaining high precision. We empirically demonstrate the effectiveness of the proposed technique as compared to existing methods and show results for a variety of tasks such as identification of dates, phone numbers, product names, and course numbers on real world datasets.

1 Introduction

Unstructured data such as emails, blogs, web pages, and chat conversations increasingly form an integral source of information for data analytics. An important step to make unstructured data available for analytics is Information Extraction (IE), which extracts structured information (that is, concepts such as phone numbers, email addresses, person names, and SSNs) from text.

Learning-based IE (for example, [15] and [13]) relies on humans to mark enough examples of the concept that has to be extracted; the labelings are then used to build a mathematical model for IE. Rule-base IE (for example, CPSL [1] and SystemT [4]) in contrast relies on humans to construct appropriate machine-understandable rules to extract specific concepts from a text. Regular expressions (regexes) are the dominant building block for a large class of extraction tasks in rule-based IE. To build effective information extraction engines, regexes that cover almost all instances of a specific concept (high recall) and have few false positives (high precision) are required.

Data-driven fine-tuning of regexes to render them more accurate for information extraction tasks is a highly challenging task. An intuitive method of modifying a regex is to specialize or generalize it. Specializing a regex involves restricting its scope so that the modified regex matches only a subset of earlier matches; an example is to modify $\backslash w\{3\}$, the regex that matches any sequence of three alpha-numeric characters to $\backslash d\{3\}$, a regex that matches any sequence of three numeric characters. An example of generalization is the reverse modification.

X.S. Wang et al. (Eds.): WISE 2012, LNCS 7651, pp. 455–467, 2012.
© Springer-Verlag Berlin Heidelberg 2012

If all the matches of a regex are labeled as true matches (those that indeed denote the concept of interest) or false matches, the regex itself can be specialized [11] to cover as many of the true matches as possible, while excluding as many of the false matches as possible. However, the analogous task of generalizing a regex to render it more accurate for IE, has not been well addressed in literature. Generalizing a regex is much harder since the scope of labeling is not limited to instances covered by the existing regex. Instead, to identify false negatives additional labeling of the text corpus is necessary.

Consider the regex, *1-\d{3}-\d{3}-\d{4}*, intended to extract phone numbers. It is possible that the intent was to also extract phone numbers of the form *1-800-COMPANY* (sometimes used for advertisements). However, no labeled instances of this kind were given to the regex learner or the regex author was unaware of such phone numbers. In such a case, it would be beneficial to learn that a slightly generalized expression such as *1-\d{3}-\w{3}-{0,1}\w{4}* covers such phone numbers. In the new regex the last two groups of digits (*\d*) were generalized to also allow letters (*\w*) and the second '-' was made optional.

We focus on regex generalization as a way to improve the recall of a regex and exploit the fact that in general humans can easily decide whether newly covered instances are true or false. In particular, we present a technique to incrementally navigate the exponential space of possible generalizations of an initial regex and identify good quality generalizations while soliciting minimal user feedback on correctness of newly discovered matches.

2 Related Work

Learning a regular expression from scratch using a list of true matches has been explored well [5–7]. It has also been found to have applications in learning other patterns like DTDs from XML data [3, 8].

Data-driven modification of a seed pattern to arrive at more accurate patterns has also been subject of some research. Outside the realm of regexes, an early work [12] proposes techniques to generalize a pattern in a data-driven manner; this, however, assumes the availability of a comprehensive set of true and false matches which is harnessed by the search procedure to fine tune the generalization. In particular, the information whether any matching instance is a true match or not, is deemed to be available beforehand. [2] proposes a clustering-based method for a similar generalization task for regexes under such an all-labels-available-beforehand assumption. Our problem addresses the more realistic scenario where only a few labels, those of instances that match the seed regex, are available. Adapting the techniques in [2, 12] to solicit labels on the fly would demand significant manual labeling effort, whereas the manual labeling effort is typically the point of optimization. We show in Section 5.5 that the technique we propose incurs up to orders of magnitude less labeling effort than methods that rely on all-labels-available-beforehand assumptions

If the seed regex is already generic enough, the technique proposed in [11] may be employed to find more accurate specializations of the regex. It works by using a labeling of all instances covered by the seed regex, to arrive at specializations that cover as many of the true matches and as few of the false matches as possible. Our work is complementary in that our goal is to discover generalizations.

[16] presents an active-learning approach to identify instances of concepts that span multiple tokens in the text. This approach makes use of lexical features, and produces a sequence of part-of-speech (POS) tags that are likely to *contain* the concept of interest. Such sequences of POS tags, unlike regexes, cannot be readily used in rule-based IE systems. But the enhanced generalized regex that we identify can be used as input to the work in [16] to discover such POS sequences that could contain the concept.

3 Generalizing a Regex

3.1 The Space of Possible Generalizations

Regexes have two main building blocks: characters (or classes of characters) and quantifiers. For example, the character class [a-zA-Z] matches any alphabet character. Quantifiers specify the number of matches expected for a set of characters. For example, \d{5,10} denotes that between 5 to 10 contiguous digits are expected.

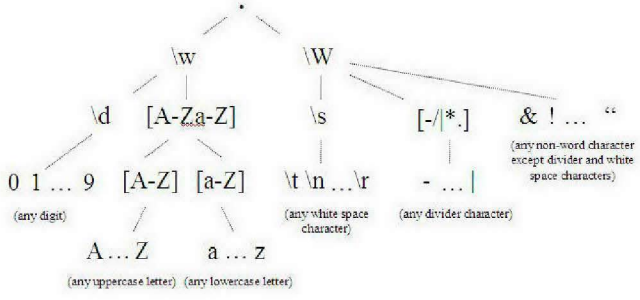


Fig. 1. Character Class Hierarchy

To generalize a regex, we generalize each component of the regex separately and incrementally. Each character class is generalized using the character class hierarchy shown in Figure 1 by incrementally going up the hierarchy until the *root* node (that is, the *dot* that matches any character) is reached. We generalize quantifiers by separately decreasing (and increasing, respectively) the lower and the upper bound of each quantifier. Capturing groups that denote a set of strings (e.g., (India|United States)) are generalized only if an appropriate dictionary or concept hierarchy covering the strings is available.

The example regex $\backslash d\{1,3\}$ has three positions: the character class digit, the lower, and the upper bound of the quantifier. $\backslash w\{0,3\}$ is derived by generalizing the first as well as the second position once each (and thus, is intuitively represented by the vector $[1,1,0]$). Figure 2 shows the space of all possible generalizations for the regex $\backslash d\{1,3\}$, up to a quantifier upper bound of 4.

Definition 1. *Let r_1, r_2, and r_3 be three regexes (with the same number of positions to generalize). r_2 is a direct generalization of r_1 if r_1 can be transformed into r_2 by generalizing a single position in r_1 exactly once. We call r_3 a generalization of r_1 if there exists a sequence of direct generalizations $[r_1, g_1, g_2, \ldots, g_n, r_3]$.*

Anthropic_0000619852

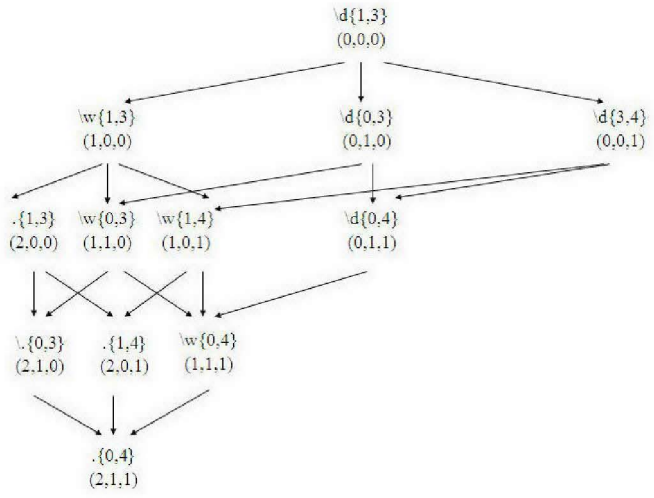


Fig. 2. Generalization Space for \d{1,3}

\w{1,3}, \d{0,3}, and \d{1,4} are direct generalizations of \d{1,3} produced by a single generalization each. \w{0,6} is a generalization of \w{1,4} with the intervening sequence $\backslash w\{0,4\}$, $\backslash w\{0,5\}$ of direct generalizations.

Definition 2. *Let r be an initial regex. The generalization space contains all regexes r_i where r_i is a generalization of r. The is-a-direct-generalization-of relation defines a partial order over all the regexes in the generalization space.*

For a regex with k positions, each node in the generalization space can have at most k direct generalizations. When considered up to a depth of n, the number of possible generalizations is up to $\mathcal{O}(k^n)$. It is infeasible to exhaustively search this exponential space. However, we show how to efficiently explore promising regions of the generalization space by employing periodic user feedback.

3.2 Quality of a Generalization

We would like to generalize an initial regex r such that the generalized regex r' has higher recall (that is, it covers more possible variations of the given concept) while still ensuring a user-specified precision for the generalized regex (that is, covering more variations does not significantly increase the amount of false matches). Coverage denotes the number of true matches covered by a regex.

We assess the quality of the new matches (that is, *Matches(r')−Matches(r)*) by soliciting user feedback on whether they are true or false matches. Let the precision (fraction of true matches) for the new matches, M_{new}, be p'; then the coverage and precision of r' is:

$$cov(r') = p * |M| + p' * |M_{new}| \qquad (1)$$

$$prec(r') = cov(r')/(|M| + |M_{new}|) \qquad (2)$$

Anthropic_0000619853

where p and M denote the precision and the number of matches for the initial regex r. However, as the goal is to minimize manual labeling effort, we do not want to label all matches in M_{new}; instead we would like to *estimate* the quality of r' even before any instances in M_{new} are labeled.

Often, we may be able to roughly estimate an upper bound on the number of possible correct matches of the concept, in a document collection. For example, a corporate email is likely to on average contain two telephone numbers, as a sender may include his/her office phone number(s), and periodically the text may contain information on how to reach a third person. Thus, we may estimate that a collection of n corporate emails contains approximately $2n$ correct matches for the phone number concept. We use such an estimate (called tcm, for total correct matches) to estimate the precision of r'.

The intuition is that even if M_{new} is much larger than tcm, the coverage will not exceed tcm. Based on this we define the upper bound on the precision of r' as:

$$ub_prec(r') = \frac{min\{p * |M| + |M_{new}|, tcm\}}{|M| + |M_{new}|}$$

We further refine the estimate using the intuition that the generalization of the i^{th} position for the j^{th} time is likely to introduce new matches of similar quality regardless of the number of generalizations performed on other positions. For a given regex, we average all historical precisions of the generalization of the i^{th} position for the j^{th} time (which we denote by $hp(i, j)$) and use it to estimate the precision of a direct generalization as follows:

$$est_prec(r') = \frac{min\{p * |M| + hp(.,.) * |M_{new}|, tcm\}}{|M| + |M_{new}|}$$

$hp(.,.)$ is the appropriate historical precision of the direct generalization. We now outline the problem of finding good quality generalizations.

Definition 3. *Given an initial regex r and a precision threshold τ, find a set of generalizations $\mathcal{R}$ such that $\forall r' \in \mathcal{R}$, $cov(r') > cov(r)$ and $prec(r') > \tau$.*

Since a generalization r' of r subsumes the matches of r, we retain only r' in $\mathcal{R}$. We now discuss an approach to find such generalizations based on the introduced concepts and definitions.

4 Interactive Regex Generalization Algorithm

Overview: Algorithm 1 describes our best-first-style search [14] that starts by initializing the candidate set with a user-specified regex. At any point, the algorithm picks the best candidate regex according to the estimated precision and calculates the candidate's actual precision by soliciting user feedback on previously unseen matches. If the calculated precision exceeds the user-specified precision threshold τ, it moves the candidate to the result set and adds its direct generalizations whose estimated precision exceeds τ to the candidate set. Upon addition of a regex to the result set, its ancestors in the

Alg. 1. ***Interactive Regex Generalization Overview***

Input. Initial regex r, Document set S, Minimal precision τ
Output. Set of generalized regexes $\mathcal{R}$

$Solutions\ \mathcal{R} = \{r\}; Candidates\ \mathcal{C} = \{c | c \in children(r) \wedge\ est_prec(c) \geq \tau\ \}$
$while(\mathcal{C} \neq \phi)$
  $pick\ most\ promising\ (highest\ estimated\ precision)\ candidate\ c\ from\ \mathcal{C}$
  $aks\ user\ to\ label\ new\ matches\ of\ c\ and\ compute\ precision$
  $if(prec(c) > \tau \wedge cov(c) > cov(parent(c)))$
    $\mathcal{R} = \mathcal{R} \cup \{c\}$
    $remove\ obsolete\ solutions\ from\ \mathcal{R}$
    $\mathcal{C} = \mathcal{C} \cup \{c' | c' \in children(c) \wedge est_prec(c') \geq \tau \wedge no\ other\ condition\ violated\ \}$

generalization space are purged from the result set. The search progresses as long as the candidate set is non-empty or until termination by the user, upon which the result set is output.

Input Parameters: The user is expected to supply a regex that satisfies her precision requirements. If a precise regex is unavailable, the user may first specialize it (see Section 2) to achieve the desired precision. Apart from the precision threshold τ, the user is also expected to provide a corpus of documents (that is, *training data*) to evaluate regexes during the search.

Navigating the Search Space: Since our generalization space forms a lattice, we are likely to reach the same generalized regex through different paths. To avoid repetetive work in search, we ensure that no regex is examined twice, by efficiently keeping track of which regexes should no longer be explored. The intricate details are omitted due to space constraints.

Picking the Most Promising Candidate: If there are multiple candidates with the same estimated precision, we break ties based on the following criteria:

1. Highest number of new matches (encourages more user feedback early on)
2. Highest number of matches of its parent (through which we reached it)
3. Smallest number of changes (prioritizes breadth first search)

Soliciting User Feedback: To compute the precision of a chosen regex, we solicit user feedback for newly covered matches. We present those instances including a snippet of surrounding text to the user and ask her to label them true or false. In case of many extra matches, we solicit user feedback only on a sample. We empirically show in Section 5 that a sample size of 10 is sufficient to achieve good results.

Maintaining the Candidate List: Once we explore a candidate, we add each direct generalization c' to the candidate set if all of the following holds:

1. c' has an estimated precision of at least τ
2. c' has never been added to the candidate set
3. no parent of c' has been evaluated with a precision below τ
4. no expression in $\mathcal{C}$ is a generalization of c'

Maintaining the Result List: We add each generalized regex whose precision is at least τ and which covers at least one new true instance with respect to its parent, to the result set. While adding a new regex to the result, we purge all regexes in any path from the root to the newly inserted regex and all other regexes that cover only a subset of matches of the new regex.

5 Experiments

5.1 Experimental Setup, Datasets and Evaluation Criteria

All experiments were run on a computer with a 1.07 GHz dual core CPU and 3 GB RAM. If not otherwise mentioned, we ran all experiments with τ = 95% and assuming tcm to be two times the coverage of the input regex. If the extra matches for a regex exceeded 10, we choose a random sample of 10 instances for user feedback. We stopped each experiment once the user had labeled 100 examples or the run-time (excluding user interaction) exceeded five minutes.

Our datasets comprise random samples from the Enron Email Corpus [9] and data under categories comp.windows.x (Windows) and misc.forsale (ForSale) in the 20 Newsgroups data set [10]. To test our approach for tasks on web pages, we use a dataset of product webpages retrieved from the IBM Intranet and course webpages from the WebKB dataset[1]. We generated separate training and test data sets.

In addition to quantifying the labeling effort incurred during the search, we also evaluated the quality of the generalized regexes. Labeling the test data to include all variations of the concept is very labor-intensive (see Section 5.5). Thus, we use the fractional increase in recall to evaluate the generalized regexes. Let M_t and M_t' be the true matches of r and r' ($|M_t'| \geq |M_t|$ where r' is a generalization of r) and M be the actual true matches (which is unknown), then the fractional increase in recall $\mathcal{R}$ is:

$$\Delta\mathcal{R}(r \rightarrow r') = \frac{(M_t'/M) - (M_t/M)}{(M_t/M)} = \frac{M_t' - M_t}{M_t}$$

5.2 Regex Generalization Evaluation

We evaluate our approach on various datasets for four tasks, pertaining to extraction of *dates*, *phone numbers*, *product names*, and *course numbers*, respectively.

Evaluation Outline: Each generalization task is first evaluated for the *training* phase where an input regex is generalized by periodically soliciting user feedback on regex matches on the training dataset. In the *testing* phase the quality of the regexes is evaluated on a different *test* dataset that comes from the same distribution as the training dataset. This two-phase approach is used in evaluating most supervised-learning approaches. *The quality of our approach for a particular task is inversely related to the manual labeling effort in the training phase and directly related to the fractional increase in recall measured on the test dataset.*

[1] `http://www.cs.cmu.edu/~webkb/`

Table 1. Results for Initial Regex r_1: $\backslash b\backslash d\{2\}/\backslash d\{2\}/\backslash d\{4\}\backslash b$

Training Data (#docs)		Test Data (#docs)			
Generalized Expressions	Labeled Instances	r_1 Matches	New Matches	Precision of New Matches	Fractional Increase in Recall (%)
Enron1 (10k)		EnronTest1 (0.1k)			
$R_1 = \backslash b\backslash d\{1,2\}/\backslash d\{2,2\}/\backslash d\{4,4\}\backslash b$	10	16	21	1.0	131
$\mathbf{R_2 = \backslash b\backslash d\{1,2\}/\backslash d\{1,2\}/\backslash d\{4,4\}\backslash b}$	20		96	1.0	600
Enron2 (1k)		EnronTest1 (0.1k)			
$R_1 = \backslash b\backslash d\{2,2\}/\backslash d\{1,2\}/\backslash d\{4,4\}\backslash b$	10	16	21	1.0	131
$R_2 = \backslash b\backslash d\{1,2\}/\backslash d\{1,2\}/\backslash d\{4,4\}\backslash b$	20		96	1.0	600
$\mathbf{R_3 = \backslash b\backslash d\{1,2\}[-/.]\backslash d\{1,2\}[-/.]\backslash d\{4,4\}\backslash b}$	38		96	1.0	600
Enron3 (1k)		EnronTest1 (0.1k)			
$R_1 = \backslash b\backslash d\{1,2\}/\backslash d\{2,2\}/\backslash d\{4,4\}\backslash b$	10	26	21	1.0	81
$R_2 = \backslash b\backslash d\{1,2\}/\backslash d\{1,2\}/\backslash d\{4,4\}\backslash b$	20		21	1.0	81
$\mathbf{R_3 = \backslash b\backslash d\{1,2\}[-/.]\backslash d\{1,2\}[-/.]\backslash d\{4,4\}\backslash b}$	50		96	1.0	369
$\mathbf{R_4 = \backslash b\backslash d\{1,2\}/\backslash d\{1,2\}/\backslash d\{2,4\}\backslash b}$	60		120	1.0	462

For each task starting with an input regex r_i, we prepare tables summarizing results for generalizing the regex. We report the successively found solutions R_i and the number of examples labeled by the user until a solution was found.

Illustrative Example: We illustrate how to interpret the results taking the first result rows in Table 1 as an example. Each training run starts with a regex r_i. We first focus on the multi-column *Training Data*; the regex generalization approach started with r_1 on dataset *Enron1*, and the first row shows that the approach solicited user feedback on *10* instances before it added the generalization R_1 to the result set. According to the second row, R_2 was added to the result set after feedback on *10* more instances (that is, a total of *20* instances). The fact that R_1 is not shown in bold, denotes that a subsequent more general solution (in this case, R_2) purged R_1 from the result set.

The multi-column *Test Data* evaluates the performance of the three regexes, r_1, R_1 and R_2, on the test set *EnronTest1* containing 100 documents. The number of instances in the test set matching r_1 is 16, as shown in the first column in the *Test Data* multi-column. The second column and all further columns needs to be read in conjunction with the corresponding R_is from the *Training Data* multi-column. R_1 resulted in 21 extra matches on *EnronTest1* beyond the 16 matches of r_1. The set of 21 extra matches has a precision of 1.0 resulting in a 131% increase in recall, over r_1. Similarly, R_2 resulted in 96 additional matches with a precision of 1.0 resulting in a 600% increase in recall.

Date Task: We performed two runs with two separate initial regexes:

$r_1 = \backslash b\backslash d\{2\}/\backslash d\{2\}/\backslash d\{4\}\backslash b$

$r_2 = \backslash b\backslash d\{2,2\}\backslash W\backslash(_{\text{January}}|\ldots|_{\text{December}})\backslash W\backslash d\{4\}\backslash b$

r_1 finds dates of the form $dd/dd/dddd$ where d is any digit. It does not cover single digit days and months, two-digit years, or divider characters other then /. Table 1 shows that r_1 is easily generalized to cover single-digit days in most datasets with a rather small labeling effort of 10 instances, whereas single-digit months are covered with another 10 labelings, on each dataset. The recorded fractional increase in recall across various datasets for the final (bold) regexes averages 500%. r_2 expects alphabetical representations of months and allows for more generic delimiters. Similar to r_1, r_2 was generalized to include single digit days after just 10 labeled instances, with the final

regexes showing a similar fractional increase in recall. Details excluded due to space limitations.

Table 2. Results for Initial Regex r_3: $\backslash b\ 1 - \backslash d\{3\} - \backslash d\{3\} - \backslash d\{4\}\backslash b$

Training Data		Test Data			
Generalized Expressions	Labeled Instances	r_3 Matches	New Matches	Precision of New Matches	Fractional Increase in Recall (%)
Enron1 (10k)		EnronTest2 (32k)			
$R_1 = \backslash b\ 1 - \backslash d\{3,3\} - \backslash d\{3,3\} - \backslash w\{4,4\}\backslash b$	4	374	3	1.0	1
$R_2 = \backslash b\ 1 - \backslash d\{3,3\} - \backslash w\{3,3\} - \backslash w\{4,4\}\backslash b$	6		12	1.0	3
$R_3 = \backslash b\ 1 - \backslash d\{3,3\} - \backslash w\{3,3\} - \{0,1\}\backslash w\{4,4\}\backslash b$	16		57	1.0	15
$R_4 = \backslash b\ 1 - \backslash d\{3,3\} - \backslash w\{3,3\} - \{0,1\}\backslash w\{4,5\}\backslash b$	18		60	1.0	16
$R_5 = \backslash b\ 1 - \backslash d\{3,3\} - .\{3,3\} - \{0,1\}\backslash w\{4,5\}\backslash b$	25		75	1.0	20
$R_6 = \backslash b\ 1 - \backslash d\{3,3\} - \backslash w\{2,3\} - \{0,1\}\backslash w\{4,5\}\backslash b$	25		74	0.99	20
$R_7 = \backslash b\ 1 - \backslash d\{3,3\} - \backslash w\{3,4\} - \{0,1\}\backslash w\{4,4\}\backslash b$	31		59	0.98	15
$\mathbf{R_8 = \backslash b\ 1 - \backslash d\{3,3\} - .\{3,4\} - \{0,1\}\backslash w\{4,4\}\backslash b}$	31		75	0.99	20
$R_9 = \backslash b\backslash d - \backslash d\{3,3\} - .\{3,3\} - \{0,1\}\backslash w\{4,5\}\backslash b$	34		76	0.98	20
$\mathbf{R_{10} = \backslash b\backslash d - \backslash d\{3,3\} - \backslash w\{2,3\} - \{0,1\}\backslash w\{4,5\}\backslash b}$	50		76	0.86	17
Enron3 (1k)		EnronTest2 (32k)			
$R_1 = \backslash b\ 1 - \backslash d\{3\} - \backslash w\{3\} - \backslash d\{4\}\backslash b$	2	374	3	1.0	1
$R_2 = \backslash b\backslash d\{1,2\} - \backslash d\{3\} - \backslash w\{3\} - \backslash d\{4\}\backslash b$	3		22	0.64	5
$\mathbf{R_3 = \backslash b\ \backslash d\{1,2\}\backslash W\backslash d\{3\} - \backslash w\{3\} - \backslash d\{4\}\backslash b}$	4		25	0.68	5
ForSale (1k)		ForSaleTest (1k)			
$R_1 = \backslash b\ 1\backslash W\backslash d\{3\} - \backslash d\{3\} - \backslash d\{4\}\backslash b$	3	17	3	1.0	18
$R_2 = \backslash b\ 1\backslash W\backslash d\{3\}\backslash W\backslash d\{3\} - \backslash d\{4\}\backslash b$	6		5	1.0	29
$R_3 = \backslash b\ 1\backslash W\backslash d\{3\}\backslash W\backslash d\{3\}[-/|.]\backslash d\{4\}\backslash b$	8		7	1.0	41
$R_4 = \backslash b\ 1\backslash W\backslash d\{3\}\backslash W\backslash d\{3\}\backslash W\backslash d\{4\}\backslash b$	11		10	1.0	59
$R_5 = \backslash b\ 1\backslash W\backslash d\{3\}\backslash W\backslash d\{3\}\backslash W\backslash w\{4\}\backslash b$	12		11	1.0	65
$\mathbf{R_6 = \backslash b\ 1\backslash W\{1,2\}\backslash d\{3\}\backslash W\{1,2\}\backslash d\{3\}\backslash W\backslash w\{4\}\backslash b}$	24		13	1.0	76
Windows (1k)		WindowsTest (1k)			
$R_1 = \backslash b\ 1 - \backslash d\{3\} - \backslash w\{3\} - \backslash w\{4\}\backslash b$	1	5	1	1.0	20
$\mathbf{R_2 = \backslash b\ 1 - \backslash d\{3\} - \backslash w\{3\} - \{0,1\}\backslash w\{4\}\backslash b}$	2		1	1.0	20

Phone Number Task: We used the following initial regex:

$r_3 = \backslash b\ 1 - \backslash d\{3\} - \backslash d\{3\} - \backslash d\{4\}\backslash b$

r_3 covers most US phone numbers, while missing out on international phone numbers and descriptive phone numbers such as 1-800-COMPANY. Despite the fact that descriptive phone numbers are rarely used in Enron mails (there are only 55 instances in the 32,000 mails) as well as the 20 Newsgroup data, our approach is able to generalize r_3 to cover such phone numbers. Labeling less than 50 instances (not full documents) is sufficient to discover a generalized regex that covers most descriptive phone numbers while barely compromising the precision of the original expression.

Table 2 shows how the initial regex is incrementally generalized to cover additional phone numbers. For example, for Enron1 the following phone numbers are included step-by-step: 1-800-982-BEVO (R_1), 1-800-The-Card (R_2), 1-800-FLOWERS (R_3), 1-800-NEWPOWER (R_4), and 1-800-97-ENRON (R_6). For Enron3, we additionally generate a regex that covers international phone numbers (e.g. 44-207-783-4040). Note that this requires three changes to the initial expression ($1 \rightarrow \backslash d \rightarrow \backslash d\{1,2\}$) where none of the individual changes produces any new matches. For the ForSale dataset, we generate a new regex that covers phone numbers with a different structural form (e.g. 1-(419)-756-2950). The variety of variants that are covered by our generalizations illustrate the effectiveness of our approach.

Product Name Task: We describe the extraction of IBM product names from a collection of IBM intranet web pages. Our initial regex r_4 is best explained by describing its three constituent parts:

(IBM): This first part enforces that the product name starts with *IBM*.

Anthropic_0000619858

Table 3. Results on Product Name and Course Number Tasks

Task	Regex	Labeled Instances	Precision of New Matches	Fractional Recall Increase (%)
Product Names	$r_4 = \backslash b(IBM)([\ ][A\text{-}Z][a\text{-}z]+)?[\ ][0\text{-}9]\{1,2\}(\backslash.[0\text{-}9])?\backslash b$	0	-	-
	$R_1 = \backslash b(IBM)([\ ][A\text{-}Z][a\text{-}z]+)\{0,2\}[\ ][0\text{-}9]\{1,2\}(\backslash.[0\text{-}9])?\backslash b$	10	1.0	297
	$\mathbf{R_2 = \backslash b(IBM)([\ ][A\text{-}Z][a\text{-}z]+)\{0,2\}[\ ][0\text{-}9]\{1,2\}(\backslash.[0\text{-}9])?\backslash b}$	11	1.0	300
Course Number	$r_5 = \backslash bCS\backslash d\{3,3\}$ (e.g., CS215)	0	-	-
	$R_1 = \backslash bCS?\backslash d\{3,3\}$ (e.g., C123)	10	1.0	23
	$R_2 = \backslash bCS?\backslash d\{2,3\}$ (e.g., CS51)	20	1.0	31
	$\mathbf{R_3 = \backslash bCS?\backslash d\{2,4\}}$ (e.g., CS1270)	45	1.0	34

$([\ ][A\text{-}Z][a\text{-}z]+)?$: This second part denotes a capitalized word (e.g., *Java*, *Lotus*) that is allowed to occur at most once ($?$ in regexes denote $\{0,1\}$).

$[0\text{-}9]\{1,2\}(\backslash.[0\text{-}9])?$: The last part captures version numbers that can contain an optional sub-version, and can capture strings such as 10 and 1.7.

r_4 is formed by putting these three expressions together with intervening whitespace. It matches product names such as *IBM Java 1.7*. Table 3 shows that the fractional increase in recall improves by 300% by querying the user for as little as 11 samples. In the process, the regex is generalized to include two optional capitalized words instead of at most one (as in r_4), thus enabling matching of product names such as *IBM Directory Server 5.1*.

Course Number Task: We provide an initial regex r_5 for the course number task that matches course numbers of the form $CS215$, accounting for a large fraction of the course numbers in the *WebKB* dataset. Our approach generalized r_5 to include course numbers of the form *CS1270* and *C512*, as shown in Table 3. The fractional increase in recall is low since most course numbers were already covered by the initial regex r_5.

5.3 Scalability

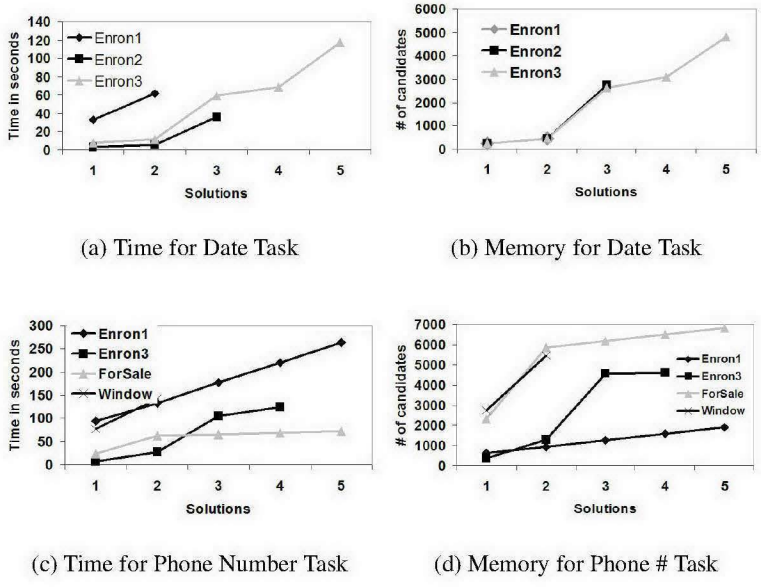
We now analyze the time taken and the memory used when generalizing r_1 and r_3 (see Section 5.2) to find the first through fifth solution. We observed similar trends for other tasks. The time is reported in seconds and includes only the runtime of our algorithm and not the time spent on user interaction (see Figures 3(a) and 3(c)). The space requirement is dominated by the size of the candidate list as well as the number of future candidates kept to ensure that no candidate is explored twice. Thus, the memory usage is reported as the maximal number of candidates and future candidates kept at any point in time to find the respective solution. Figures 3(b) and 3(d) show that the maximum number of items kept in memory is only of the order of thousands.

5.4 Effect of Changing Input Parameters

The precision threshold (τ) simply excludes regexes below a certain precision from the result. In the following we assume $\tau = 0.95$ and study the effect of varying the estimated number of correct matches. We describe our analysis for r_1 on Enron2; similar results were observed for other regexes and data sets.

The parameter tcm influences the amount of exploration of the search space: larger values of tcm induce a larger search. For example, for tcm at 110% of the coverage of

the initial regex, only R_1 is found by the search. When increased to 125%, R_2 is still not found but the search discovers $\backslash b\backslash d\{1,2\}/\backslash d\{2,2\}/\backslash d\{4\}\backslash b$; this regex is later subsumed by R_2 for larger tcm values. Any tcm value over 140% produces R_1, R_2, and R_3.



(a) Time for Date Task (b) Memory for Date Task

(c) Time for Phone Number Task (d) Memory for Phone # Task

Fig. 3. Time and Memory Usage

It is generally safe to overestimate tcm but a very high tcm may lead to an increase in runtime. Though we found 200% to be empirically effective across many settings, a user may want to first run experiments with lower values of tcm and increase it incrementally if no good results are found within the first few minutes.

Table 4. Comparison of Labeling Efforts

Data Set (#docs)	Regex Generalizations	True instances of new kind	Instances labeled in our approach	Estimated documents to label for learning methods
Enron1 (10k)	$r_1 \rightarrow R_2$	1711	20	6
	$r_3 \rightarrow R_8$	26	31	385
	$r_3 \rightarrow R_{10}$	25	50	400
Enron2 (1k)	$r_1 \rightarrow R_3$	59	38	17
Enron3 (1k)	$r_1 \rightarrow R_3$	55	50	19
	$r_1 \rightarrow R_4$	420	60	2
	$r_3 \rightarrow R_3$	3	4	333
ForSale (1k)	$r_3 \rightarrow R_6$	13	24	77
Windows (1k)	$r_3 \rightarrow R_2$	1	2	1000

5.5 Comparison with Related Work

Approaches for learning regexes from scratch (see Section 2) need at least one example of each kind of instance. The labeling effort to cover less common variants is typically

significant. For example, the 10k sample of Enron emails contains only 26 instances of the new kind of phone numbers covered by our generalization from r_3 to R_8. In order to label at least one instance, a human may need to label 385 (that is, $10,000/26$) emails under random sampling. In contrast, labeling 31 instances (not complete emails) was sufficient to derive R_8 using our approach. Table 4 compares the labeling effort in our approach against the effort of labeling at least one instance of the new kind.

We also compare our results against the clustering-based approach (CBA) [2] which works by generalizing an initial regex guided by the set of all true matches. We provide all true examples procured during our interactive generalization process as correct examples to CBA. As expected, we observe that CBA works well if enough labeled instances of each kind exist. CBA generalizes $r1$ (on Enron1) to *three expressions* that together cover exactly the same instances as R_2 (see Table 1) though it does not discover R_2 itself. Thus, even while taking advantage of our sampling, CBA generates result sets that contain many expressions that together are only as powerful as a single regex in our approach. CBA fails to work in cases where only a few true matches exist and is unable to generalize r_3 for any of the datasets.

6 Conclusion and Future Work

We have described a technique to find generalizations of a regular expression that improve its recall without much affect on precision. We have demonstrated empirically that it significantly reduces the labeling effort involved in arriving at regexes that also cover uncommon forms of a concept. We were able to identify good generalizations of an expression within a few minutes and with labeling only tens of instances as opposed to examining hundreds of documents.

As described in Section 2, our work is orthogonal to the work presented in [11] that specializes a regex. It will be interesting to explore how to combine these two approaches into an integrated method that uses both generalizations and specializations. For example, in the search process we may initially over-generalize and then specialize to reduce the false positives.

References

1. Appelt, D.E.: Introduction to information extraction. AI Commun. 12(3), 161–172 (1999)
2. Babbar, R., Singh, N.: Clustering based approach to learning regular expressions over large alphabet for noisy unstructured text. In: AND Workshop, pp. 43–50 (2010)
3. Bex, G.J., Neven, F., Schwentick, T., Tuyls, K.: Inference of concise DTDs from XML data. In: VLDB, pp. 115–126 (2006)
4. Chiticariu, L., Krishnamurthy, R., Li, Y., Raghavan, S., Reiss, F., Vaithyanathan, S.: SystemT: An algebraic approach to declarative information extraction. In: ACL, pp. 128–137 (2010)
5. Ciravegna, F.: Adaptive information extraction from text by rule induction and generalisation. In: IJCAI, pp. 1251–1256 (2001)
6. Denis, F.: Learning regular languages from simple positive examples. Machine Learning 44(1/2), 37–66 (2001)
7. Fernau, H.: Algorithms for learning regular expressions from positive data. Inf. Comput. 207(4), 521–541 (2009)

8. Garofalakis, M.N., Gionis, A., Rastogi, R., Seshadri, S., Shim, K.: XTRACT: A system for extracting document type descriptors from XML documents. In: SIGMOD, pp. 165–176 (2000)
9. Klimt, B., Yang, Y.: Introducing the Enron corpus. In: CEAS (2004)
10. Lang, K.: 20 Newsgroups (1997), http://people.csail.mit.edu/jrennie/20Newsgroups
11. Li, Y., Krishnamurthy, R., Raghavan, S., Vaithyanathan, S., Jagadish, H.V.: Regular expression learning for information extraction. In: EMNLP, pp. 21–30 (2008)
12. Mitchell, T.M.: Generalization as search. Artif. Intell. 18(2), 203–226 (1982)
13. Nie, Z., Wen, J., Zhang, B.: 2D conditional random fields for web information extraction. In: ICML, pp. 1044–1051 (2005)
14. Pearl, J.: Heuristics: intelligent search strategies for computer problem solving. Addison-Wesley Longman Publishing Co., Inc., Boston (1984)
15. Sarawagi, S., Cohen, W.W.: Semi-markov conditional random fields for information extraction. In: NIPS, pp. 1185–1192 (2004)
16. Wu, T., Pottenger, W.M.: A semi-supervised active learning algorithm for information extraction from textual data: Research articles. JASIST 56(3), 258–271 (2005)