Exploiting Structure in Representation of Named Entities using Active Learning

Nikita Bhutani
University of Michigan, Ann Arbor
nbhutani@umich.edu

Kun Qian
IBM Research - Almaden
qian.kun@ibm.com

Yunyao Li
IBM Research - Almaden
yunyaoli@us.ibm.com

H. V. Jagadish
University of Michigan, Ann Arbor
jag@umich.edu

Mauricio A. Hernandez
IBM Research - Almaden
mahernan@us.ibm.com

Mitesh Vasa
IBM Research - Almaden
mitesh.vasa@us.ibm.com

Abstract

Fundamental to several knowledge-centric applications is the need to identify named entities from their textual mentions. However, entities lack a unique representation and their mentions can differ greatly. These variations arise in complex ways that cannot be captured using textual similarity metrics. However, entities have underlying structures, typically shared by entities of the same entity type, that can help reason over their name variations. Discovering, learning and manipulating these structures typically requires high manual effort in the form of large amounts of labeled training data and handwritten transformation programs. In this work, we propose an active-learning based framework that drastically reduces the labeled data required to learn the structures of entities. We show that programs for mapping entity mentions to their structures can be automatically generated using human-comprehensible labels. Our experiments show that our framework consistently outperforms both handwritten programs and supervised learning models. We also demonstrate the utility of our framework in relation extraction and entity resolution tasks.

1 Introduction

Named entities are atomic objects of reference and reasoning in many cognitive applications and knowledge-centric services like deep question answering, text summarization and analytics. A real-world entity may have a great variety of representations (Galárraga et al., 2014; Nakashole et al., 2011). For example, University of California, Santa Cruz could have different string representations or *name variations*: UCSC, UC Santa Cruz, UC–Santa Cruz. Determining if two representations refer to the same entity is an important primitive in entity resolution and entity linking algorithms that drive these knowledge-centric applications (Shen et al., 2015; Arasu and Kaushik, 2009).

Unifying entity representations has been widely studied in record linkage (Christen, 2012), deduplication (Elmagarmid et al., 2007) and reference matching (McCallum et al., 2000). Typically, duplicates are identified using the attributes and/or the contextual information of entities (Zhang et al., 2010; Han et al., 2011; Shen et al., 2015). The string representation or *mention* of an entity forms key evidence: similar mentions likely refer to the same entity. Although deemed as crucial (Dredze et al., 2010), variations in mentions are typically handled using textual similarity like edit distance and cosine similarity (Zheng et al., 2010; Lehmann et al., 2010; Liu et al., 2013), which can be misleading.

Example. Consider the mentions: (a) General Electric Corporation, (b) General Electric China Corporation, (c) GE Corp. Mentions (a) and (b) are textually similar, differing in just one token. However, they refer to different entities, owing to the location detail 'China'. Conversely, textually dissimilar mentions (a) and (c) refer to the same entity.

An entity mention is not merely a sequence of characters (Arasu and Kaushik, 2009). It instead has an internal structure, specific to the type of entity. For example in Table 1, the company mentions have a $\langle name \rangle$, optionally followed by $\langle loc \rangle$ and $\langle suffix \rangle$. Such structural interpretation can help design similarity

Proceedings of the 27th International Conference on Computational Linguistics, pages 687–699
Santa Fe, New Mexico, USA, August 20-26, 2018.

Unlabeled Mention	Structured Representation	Labeled Mention
IBM United Kingdom Ltd.	⟨name⟩⟨loc⟩⟨suffix⟩	IBM→⟨name⟩,United Kingdom→⟨loc⟩, Ltd.→⟨suffix⟩
Alibaba USA	⟨name⟩⟨loc⟩	Alibaba→⟨name⟩, USA→⟨loc⟩
Barclays	⟨name⟩	Barclays→⟨name⟩
Hewlett-Packard Co.	⟨name⟩⟨suffix⟩	Hewlett-Packard→⟨name⟩, Co.→⟨suffix⟩
University of California, Irvine	⟨name⟩,⟨loc⟩	University of California→⟨name⟩, Irvine→⟨loc⟩
UM-Ann Arbor	⟨name⟩-⟨loc⟩	UM→⟨name⟩, Ann Arbor→⟨loc⟩
Stanford University	⟨name⟩	Stanford University→⟨name⟩

Table 1: Example Structured Representations of Company and University Mentions

functions to capture the nuances in name variations of an entity that string similarity functions cannot. For instance, GE can be explained as a mention of General Electric Corporation using transformations like abbreviate $\langle name \rangle$ and drop $\langle suffix \rangle$. These transformations coupled with string similarity functions can augment existing entity linking algorithms (Qian et al., 2017).

The name variations and, consequently, the transformations are highly domain-dependent (Arasu et al., 2008). Designing similarity functions for an entity type that can reason over the structure of entities requires: (a) a comprehensive list of the structured representations (e.g. column 2 in Table 1), and (b) programs that can map mentions to these structures (e.g. column 3 in Table 1). Traditionally, a domain expert would scan a list of mentions of a target entity type to identify the different structured representations and handwrite programs. This requires specialized skills, and is error-prone and expensive, taking up to several person months to tune the programs for a single application (Campos et al., 2015).

To alleviate the high manual effort, some of the prior works (Arasu and Kaushik, 2009) use declarative, programmable frameworks that allow an expert to directly manipulate the mentions. The expert can provide a program as a set of grammar rules to generate the structured representations. While this equips the expert to manipulate the structure of a mention, it is only a partial solution. The rules are not generic, requiring the expert to specify how each mention is parsed to its structure. This is wasteful because a structure, shared by several mentions, can be captured with a generic rule. Another limitation of previous approaches is that they do not handle structural ambiguities. For instance, there can be multiple structures of Apple Inc. such as $\langle name \rangle \langle loc \rangle$ and $\langle name \rangle \langle suffix \rangle$, only one of which is correct.

Since several mentions of an entity type tend to have similar structures, it is possible to learn the various structures from a subset of mentions. Given a good query strategy, active learning offers a promising approach to efficiently select a small set of such mentions. Moreover, the expert need not provide programs that parse the selected mentions and generalize to unseen mentions. These programs can be induced from the labels for the structures of the mentions. Embodying these ideas, we propose LUSTRE, an active-learning based framework that learns structured representations for an entity type from human-comprehensible labels for a small set of mentions. It automatically synthesizes *generalizable* programs from the labels to map new mentions of the entity type to the learned structured representations. In addition, it allows the expert to incorporate domain knowledge and additional feedback to handle structural ambiguities. Our framework significantly reduces the manual effort in labeling mentions and writing programs for the structured representations. We also demonstrate how these structured representations help define similarity functions that benefit entity resolution, and string transformation functions that benefit relation extraction. The intellectual contributions of this work are as follows:

- *Structured Representations.* We present a framework to reason about name variations of entities based on their structured representations.
- *Active-learning.* We present an active-learning approach and a unified query strategy to learn structured representations for a target entity type with minimal human input.
- *Program Synthesis.* We propose to automatically synthesize *generalizable* programs from human-understandable labels to map mentions to their structured representations.
- *Experimental Evaluation.* Our experiments show that our framework achieves an average of 92% precision and 86% recall in predicting structured representations for several entity types, outperforming competing approaches that require high manual effort. We demonstrate the usefulness of the structured representations in two important tasks: entity resolution and relation extraction.

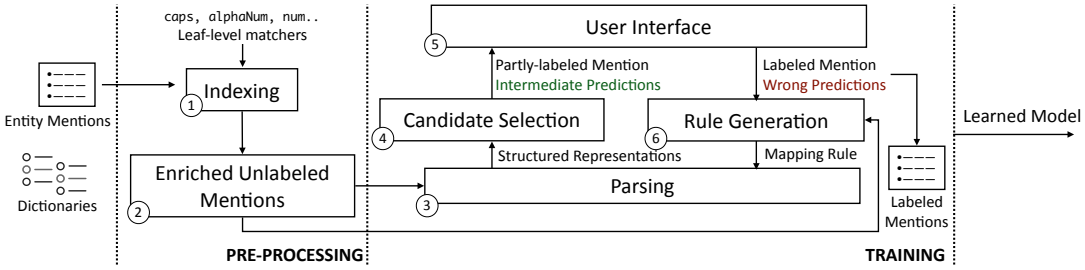

Figure 1: LUSTRE Architecture

2 Problem Formalization and Notations

Given a set of unlabeled mentions (i.e., raw strings) of a target entity type ϵ, our goal is to actively learn a high-quality model M_ϵ (with a small number of user labels) that can map a new mention of type ϵ to its structured representation. Key technical challenges in learning such a model include: (1) designing an effective query strategy to find representative mentions that are structurally similar to several other mentions and to find mentions with diverse structures; (2) automatically inferring mapping programs from the user labels to reduce the human effort, as in (Campos et al., 2015) and (Arasu and Kaushik, 2009); (3) handling ambiguities when an unlabeled mention has multiple candidate structured representations.

Mentions of a target entity type have internal *structured representations* consisting of atomic *semantic units*. Given the labels for the semantic units for a small set of mentions, we want to learn a model M of *mapping rules*. Each mapping rule converts a raw string mention into a structured representation.

Definition 1 (Structured Representation)**.** *A structured representation $\mathcal{S}$ of an entity mention is a sequence of atomic semantic units that compose its structure.*

Textually dissimilar mentions can have the same structure (e.g. Apple Inc. and Hewlett-Packard Co.).

Definition 2 (Semantic Unit)**.** *A semantic unit is a tuple $u = \langle l : p \rangle$ where l is a label for the unit and p is a pattern matching function (e.g., a regular expression), referred to as a **matcher**.*

A matcher describes how a substring in a mention matches a semantic unit. For instance, a matcher ([A-Z][A-Za-z]*[-']?)+ captures $\langle name \rangle$ in mentions Apple Inc. and GE Corp. Name variations of an entity can be explained as transformations on its semantic units (e.g. dropping suffix in Apple Inc. generates Apple).

Definition 3 (Mapping Rule)**.** *The mapping rule $r_\mathcal{S}$ for a structured representation $\mathcal{S}$ consists of matchers which decide how an unlabeled mention is mapped to $\mathcal{S}$, resulting in a **labeled entity mention**.*

Specifically, a mapping rule $r_\mathcal{S}$ is a sequence of matchers in the semantic units in $\mathcal{S}$, denoted by $r_\mathcal{S} = \{p_i \mid \langle l_i, p_i \rangle \in \mathcal{S}\}$. This mapping rule constitutes a program that maps unseen mentions to $\mathcal{S}$. Together the various mapping rules constitute a model M_ϵ for the entity type ϵ.

3 The LUSTRE System

We propose LUSTRE that addresses the aforementioned challenges in learning a model of mapping rules for an entity type by:

- adopting a unified query strategy that combines uncertainty sampling (i.e., selecting a mention whose current structured representation is unknown or uncertain) and density-weighted sampling (i.e., selecting a mention whose structured representation is representative of many unlabeled mentions).
- seeking human-comprehensible labels for the semantic units in the structure of an unlabeled mention, and deducing a mapping rule by combining the matchers for the semantic units.
- handling structural ambiguities of an unlabeled mention by ranking the candidate mapping rules based on their *reliability* and additional user feedback.

Figure 1 depicts the workflow of LUSTRE and Algorithm 1 shows our learning algorithm. It takes as input unlabeled mentions $\mathcal{U}_\epsilon$ and optionally dictionaries $\mathcal{D}_\epsilon$ of a target entity type ϵ. These dictionaries and a set of pre-defined matchers form the building blocks for the mapping rules. Before training, LUSTRE evaluates all the matchers against $\mathcal{U}_\epsilon$ to inform the query strategy and rule generation (Sec 3.1).

Algorithm 1 LUSTRE learning algorithm

input: $\mathcal{U}_\epsilon$ = a pool of unlabeled mentions $\{m\}$
output: $\mathcal{M}_\epsilon$ = a model of mapping rules $\{r_\mathcal{S}\}$

1: **function** TRAIN($\mathcal{U}_\epsilon$)
2: $\mathcal{L}_\epsilon$ = a set of labeled mentions $\{\langle m, l\rangle\}$
3: $\mathcal{F}$ = feedback $\{\langle m, l, f\rangle \mid m \in \mathcal{U}, f \in \{0, 1\}\}$
4: $\mathcal{M} = \varnothing$
5: **for** t = 1,2... **do**
6: $\mathcal{M}$ = UPDATE($\mathcal{M}, \mathcal{L}, \mathcal{F}$)
7: select $m^* \in \mathcal{U}$, mention with highest utility
8: select $\mathcal{F}$ with least confident predictions
9: query label l^* for mention m^*
10: query binary feedback on $\mathcal{F}$
11: $\mathcal{L} = \mathcal{L} \cup \langle m^*, l^*\rangle, \mathcal{U} = \mathcal{U} \backslash \{m^*\}$
12: **return** $\mathcal{M}$
13: **function** UPDATE($\mathcal{M}, \mathcal{L}, \mathcal{F}$)
14: **for** $\langle m, l\rangle \in \mathcal{L}$ **do**
15: $r_\mathcal{S} = generate\_rule(l)$
16: **if** $r_\mathcal{S} \notin \mathcal{M}$ **then**
17: $\mathcal{P}_{r_\mathcal{S}} = reliability(r_\mathcal{S})$
18: $\mathcal{M} = \mathcal{M} \cup \langle r_\mathcal{S}, \mathcal{P}_{r_\mathcal{S}}\rangle$
19: **for** $\langle r_\mathcal{S}, \mathcal{P}_{r_\mathcal{S}}\rangle \in \mathcal{M}$ **do**
20: $\mathcal{P}_{r_\mathcal{S}} = estimate(\mathcal{P}_{r_\mathcal{S}}, \mathcal{F})$
21: **return** $\mathcal{M}$

During training, in each iteration, LUSTRE selects a candidate mention for the user to label (Sec 3.2). Given the user-provided labels for semantic units of the selected mention, it derives a generic mapping rule for the structure of the mention (Sec 3.3). It then updates the model M with the new rule, and predicts the structures of unlabeled mentions (Sec 3.4). It presents a sample of these predictions to the user and integrates the user feedback in M (Sec 3.5). This iterative process continues until all the unlabeled mentions can be mapped to some structure or the user is satisfied with M.

3.1 Indexing

User-provided dictionaries and pre-defined regular expressions constitute the vocabulary of matchers for the mapping rules. The dictionaries help capture key domain-specific terminology but need not correspond to semantic units.

Example. *A suffix dictionary may contain 'Inc.', 'Corp', 'LLC', 'Pvt. Ltd.'* □

Additionally, we use type-independent regex matchers shown in Table 2. In a pre-processing step (Line 1 of Algorithm 1), we evaluate the unlabeled mentions against the matchers to save computational overhead. We refer to these *enriched unlabeled mentions* during training.

Example. *Given dictionaries for country and suffix and a mention IBM UK Ltd., the matchers yield matches UK→*$\langle country\rangle$*, Ltd.→*$\langle suffix\rangle$ *IBM→* $[\langle caps\rangle, \langle alphaNum\rangle]$*, UK→* $[\langle caps\rangle]$. □

Multiple matchers can match the same token (e.g. $\langle country\rangle$ and $\langle caps\rangle$ match 'UK'). We assume that more specific matchers offer higher precision than generic matchers. We further formalize this intuition in Section 3.3. We rank matchers in the following order based on their selectivity: $\mathcal{D}_\epsilon > caps > alphaNum > num > special > wild$[1].

Name	Regex
caps	[A-Z][A-Za-z]*[.-']?
alphaNum	[A-Za-z0-9]+[.-']?
num	[0-9]+
special	[^A-Za-z0-9]+
wild	.+

Table 2: Predefined Matchers

3.2 Candidate Selection

The query strategy to determine what constitutes an informative mention is the central challenge in our active-learning setting (Line 7 of Algorithm 1). We consider a mention *informative* if its represents the structure of several other unlabeled mentions and its current structure is unknown or uncertain. We adopt a unified approach, combining density-weighted sampling (Settles and Craven, 2008) and uncertainty sampling (Culotta and McCallum, 2005) as it is robust to outliers and input distribution.

For each unlabeled mention m_i, we compute a correlation score c_i (based on its structural similarity to other mentions) and an uncertainty score f_i (based on its predicted structure). We then compute a utility score u_i for m_i, combining its correlation and uncertainty scores, and select a mention m^* with the highest utility score for labeling.

To compute correlation score c_i, we need a reliable metric to measure the similarity of the structures of two mentions. Since surface-string similarity metrics won't suffice, we estimate structural similarity of two mentions as a function of the matchers that constitute their structures. Specifically, we compute structural similarity $c(i, j)$ of a pair of mentions (i, j) as the edit distance of their structures, $\mathcal{S}_i$ and $\mathcal{S}_j$.

[1]The order of preference can be inferred in a pre-processing step in which the system counts the selectivity of each matcher against the input mentions

Example. *'IBM Ltd.' and 'Apple Inc.' have the same structure* $\langle caps \rangle \langle suffix \rangle$*, and therefore, have an edit distance of 0. Each has edit distance 1 to 'Microsoft Asia Inc.' with structure* $\langle caps \rangle \langle loc \rangle \langle suffix \rangle$*.* □

Given the pair-wise structural similarity metric, the correlation score c_i of a mention m_i is its average structural similarity to other unlabeled mentions.

$$c_i = \frac{1}{|U|} \sum_{j \in U} c(i,j) \qquad f_i = \begin{cases} 1 & \text{if no rule maps } m_i \\ 1 - \mathcal{P}_{\hat{r}_\mathcal{S}} & \text{otherwise} \end{cases} \qquad u_i = c_i * f_i$$

$$c(i,j) = 1 - \frac{\text{edit distance}(\mathcal{S}_i, \mathcal{S}_j)}{\text{max edit distance}} \qquad \mathcal{P}_{\hat{r}_\mathcal{S}} = \operatorname*{argmax}_r \mathcal{P}_{r_\mathcal{S}} \qquad m^* = \operatorname*{argmax}_{m_i} u_i$$

To estimate the uncertainty score f_i of a mention, we use $\mathcal{P}_{r_\mathcal{S}}$, the reliability of the mapping rules that can parse the mention. If no mapping rule can parse a mention m_i, its structure is unknown i.e. f_i is simply 1. Otherwise, it is the uncertainty of the most reliable rule $\hat{r}_\mathcal{S}$ known to parse the mention. We discuss how the reliability $\mathcal{P}_{r_\mathcal{S}}$ of mapping rules are estimated in Sec. 3.4.

3.3 Rule Generation

Once the user labels a selected mention, LUSTRE next has to synthesize a generic program i.e. a mapping rule for the structure of the mention (Line 15 of Algorithm 1). Deriving a mapping rule is non-trivial as semantic units in the structure can potentially span multiple tokens and matchers. As a result, there are many ways to combine matchers and derive the rule.

Example. *A user can label 'General Motors' as semantic unit* $\langle name \rangle$ *in 'General Motors Co.' Tokens 'General' and 'Motors' map to matchers* $caps$ *and* $alphaNum$*. The most reliable interpretation for* $\langle name \rangle$ *is* $\langle name ::caps\{1,2\} \rangle$*, a combination of two adjacent matchers* $\langle caps \rangle$*.* □

LUSTRE derives a reliable rule as the sequence of most selective matchers, where *selectivity* is the expected number of matches of a matcher over the set of unlabeled mentions $\mathcal{U}$ (Li et al., 2008).

$$sel(p_i) = \mathrm{E}[match(p_i, m \in \mathcal{U})]$$

with $match(p_i, m)$ being number of matches of p_i over mention m.

3.4 Parsing

When a new mapping rule is learned, we want to estimate its reliability in predicting structures of mentions, and update the model M (Line 17-18 of Algorithm 1). These reliability scores are used for estimating utility scores at candidate selection and for resolving ambiguities when multiple rules can parse a mention (with preference given to the most reliable rule). Following the intuition that generic rules are less reliable, we estimate reliability of a rule based on its expected numbers of matches in the unlabeled mentions. Specifically, reliability $\mathcal{P}_{r_\mathcal{S}}$ is a function of the selectivity of the matchers in $r_\mathcal{S}$.

$$p^* = \operatorname*{argmin}_i \{sel(p_i) \mid \langle l_i, p_i \rangle \in \mathcal{S}\}$$

$$\mathcal{P}_{r_\mathcal{S}} = 1 - sel(p^*)$$

3.5 User Interface

LUSTRE uses an easy-to-use interface (Qian et al., 2018) to seek labels for a selected mention and additional feedback on intermediate predictions (Line 9 of Algorithm 1). To reduce the labeling effort, tokens in the mention that match an entry in the dictionary are pre-labeled with the name of the dictionary (e.g. 'IBM Corp' is presented with 'Corp' labeled as suffix). The user can keep these and/or provide new labels (e.g. user can label 'IBM' as name).

In addition, the interface presents the predictions of structures for a sample of unlabeled mentions. It selects 10 least confident predictions based on uncertainty scores (Section 3.2). The user can simply mark a prediction incorrect, without providing labels for its correct structure. LUSTRE effectively integrates this feedback to avoid over-estimating the reliability of learned rules, thereby improving the quality of model M. Specifically, it updates the reliability of a mapping rule $\mathcal{P}^i_{r_\mathcal{S}}$ as a function of number of incorrect predictions $\overline{m_r}$ for the rule in the set of 10 predictions presented to the user (Line 20 of Algorithm 1).

$$\mathcal{P}^j_{r_\mathcal{S}} = \mathcal{P}^i_{r_\mathcal{S}} * (1 - \alpha * \frac{|\overline{m_r}|}{10})$$

with α being the decay constant. We found such estimation, though simple, was effective in estimating the reliability of mapping rules. Training more powerful measures such as a discriminative model would require larger user feedback than is available in this setting.

4 Experiments

We assess the effectiveness of our learning algorithm, and the quality and usefulness of learned structures.[2]

4.1 Experimental Setup

Datasets. We conduct experiments on four entity types of varied complexity and ambiguity.

- Person: Focusing on subtype *Individual*, we randomly select 200 unique mentions for training and another 200 mentions for testing from the ACE 2005 dataset (Walker et al., 2006). For out-of-domain tests, we randomly select 200 person mentions from Freebase (Bollacker et al., 2008).
- Company: Focusing on subtype *Commercial*, we randomly select 200 mentions for training and use the remaining 100 mentions for testing from the ACE dataset. For out-of-domain test, we randomly select 200 company mentions from Freebase.
- Tournament: We use a 50/50 split for the 100 unique mentions of tournaments in Freebase.
- Academic Title: We use a 50/50 split for the 350 unique mentions of academic titles in Freebase.

For each mention, we ask two experts to manually annotate every token with a semantic label to produce the ground truth. The average inter-annotator agreement was 0.89 for Cohen's κ.

Baselines. We compare LUSTRE with the following methods.

- STG: A commercial system (Campos et al., 2015) which requires an expert (with domain knowledge and programming skills) to analyze the structures of an entity type and handwrite mapping programs.
- Linear-chain CRF[3]: A linear-chain CRF model that predicts a sequence of labels for the tokens in a mention. We use matches from the *Indexing* stage as features. We use two different training settings: CRF trained on the entire training set (to compare with best model), and CRF^{L} trained on the subset of training set selected via the query strategy in LUSTRE (to compare with same user effort).
- LUSTRE^{T}: LUSTRE with a native tf-idf based query strategy.

Evaluation Metric. We consider a prediction correct if the model and expert agree on the semantic labels for each token in the mention. We measure *precision* as the fraction of predictions that are correct and *recall* as the fraction of correct structures that are predicted. We define a new metric, which we call the α value, to estimate the role of manual effort on performance of various methods (definition is given below). Intuitively, higher the α value, higher the effectiveness of a method in learning from a user label.

$$\alpha(X,t) = \frac{\text{F-score of method } X \text{ on entity type } t}{\text{number of user labels requested by } X}, \text{where X} \in \{\text{LUSTRE, CRF, CRF}^{\text{L}}\}.$$

4.2 Quality Analysis

The performance results of the different methods are summarized in Table 3. For *Person* mentions, all methods achieve reasonably good results (F-scores are all above 0.85). This is not surprising as person mentions typically have simple structures with few semantic units. STG achieves lowest performance, indicating that manually-crafted programs are not as robust as learned models. CRF models, especially CRF^{L} trained on mentions selected using LUSTRE, show evident improvement over STG. LUSTRE achieves comparable performance. However, it outperforms all other methods on out-of-domain test data, suggesting it can capture the structures in mentions regardless of their data source.

Company mentions are more complex than *Person* mentions. They can have several semantic units such as core name, location, suffix and subsidiary, which can appear in different orders, be separated by special symbols etc. Consequently, capturing structures in company mentions is more difficult, as is reflected in the performance across methods. Learning a reliable model/program would require higher

[2]We will release the code and data from this work. Code is proprietary but can be licensed.

[3]http://mallet.cs.umass.edu/sequences.php

manual effort in STG and more training data in CRF. In contrast, LUSTRE, with small human input, achieves high precision and recall for both in-domain and out-of-domain data.

Tournament and *Academic Title* mentions have even more complex structures that exhibit more variations. Even for these complex types, LUSTRE outperforms CRF. We do not report results of STG for these two entity types because these were not considered when STG was developed. We do not provide an out-of-domain evaluation of the two entity types because only the Freebase data included mentions of these entity types.

In summary, LUSTRE outperforms STG and CRF-based methods in terms of overall F-score with the exception of *Person* where CRF^{L} has a small improvement over LUSTRE. Furthermore, its unified query strategy is more effective than a tf-idf based strategy, as is reflected in the performance of LUSTRE^{T}.

We found two main sources of errors made by LUSTRE. First, akin to all learning methods (such as CRF), it has low recall when the training data is not representative. Second, ranking mapping rules sometimes cannot effectively resolve structural ambiguities. For example, 'The Stanford University Professorship in Nephrology' can be interpreted as $\langle honorary\ \ prefix\rangle\langle title\rangle\langle specialty\rangle$ or $\langle institute\rangle\langle title\rangle\langle specialty\rangle$. The former is incorrect, but being more commonly observed in the training data, is ranked higher.

		In-Domain			Out-of-Domain		
Type	Algorithm	P	R	F_1	P	R	F_1
Person	STG	0.92	0.92	0.92	0.85	0.85	0.85
	CRF	0.97	0.97	0.97	0.90	0.90	0.90
	CRF^{L}	**0.99**	**0.99**	**0.99**	0.91	0.91	0.91
	LUSTRE^{T}	0.98	0.95	0.96	**0.92**	0.90	0.91
	LUSTRE	**0.99**	0.97	0.98	**0.92**	**0.95**	**0.93**
Company	STG	0.83	0.83	0.83	0.79	0.79	0.79
	CRF	0.87	**0.87**	0.87	0.85	**0.85**	0.85
	CRF^{L}	0.81	0.81	0.81	0.73	0.73	0.73
	LUSTRE^{T}	0.84	0.77	0.80	0.78	0.60	0.68
	LUSTRE	**0.95**	0.86	**0.90**	**0.91**	**0.85**	**0.88**
Tournament	CRF	0.70	0.70	0.70	-	-	-
	CRF^{L}	0.68	0.68	0.68	-	-	-
	LUSTRE^{T}	0.96	0.68	0.79	-	-	-
	LUSTRE	**0.96**	**0.90**	**0.93**	-	-	-
Academic Title	CRF	0.69	**0.69**	0.69	-	-	-
	CRF^{L}	0.67	0.67	0.67	-	-	-
	LUSTRE^{T}	0.36	0.23	0.28	-	-	-
	LUSTRE	**0.79**	0.65	**0.72**	-	-	-

Table 3: Performances of LUSTRE, STG and CRF

4.3 Effectiveness

To assess how the quality of structured representations evolves in the learning process, we examine the precision and recall of LUSTRE after each iteration (as shown in Figure 2). We found that the precision remains nearly constant. There are a limited number of structures for the same types of entities. Every time LUSTRE learns a precise rule that improves coverage. There are slight drops in the precision due to the long tail of non-representative cases. The recall generally increases depending on whether the system prefers to gather additional evidence or to discover new structured representations.

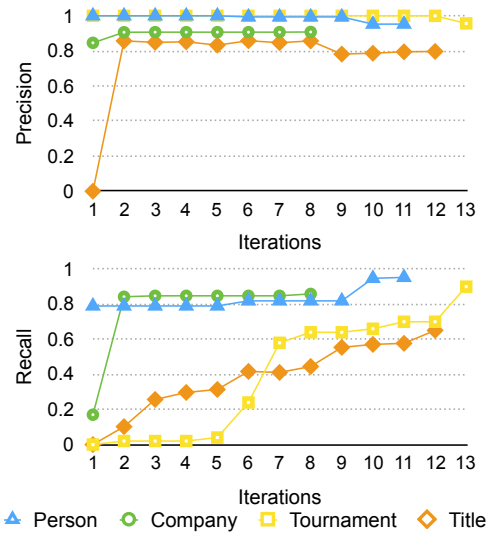

Figure 2: Performance of LUSTRE over iterations

We also examine the number of rules learned, number of incorrect intermediate predictions, and percentage of input training data covered after each learning iteration in LUSTRE (as shown in Figure 3). We found that it took 8-13 iterations to learn almost all different structures of an entity type. The fraction of training data that could be parsed using the mapping rules also generally increased, indicating that our query strategy selected structurally diverse mentions for labeling. Only a few (<5) predictions were

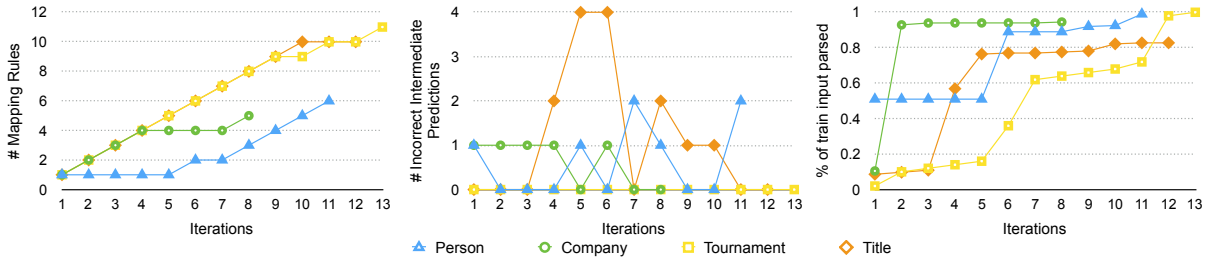

Figure 3: Number of mapping rules, number of incorrect predictions, fraction of training data parsed after each iteration

marked incorrect in each iteration. This suggests that LUSTRE, with a small set of labeled mentions and a little user feedback, learned reliable rules and ranked them correctly.

Next, we compare the nature and amount of human effort across different methods. STG requires high effort and high skill. Even for a simple entity type *Person*, it takes one skilled developer a few months to manually inspect the different structural patterns and write programs. Learned models such as LUSTRE and CRF reduce the skill level and require only human-readable labels for a mention. However, in contrast to CRF, LUSTRE does not require the user to collect and label a set of representative mentions. It guides the user to interactively and iteratively label a few mentions, with each iteration taking less 7 seconds. The optional feedback also is small (<5 predictions) and low-effort (boolean labels).

Quantitatively, we report the α values of the three learning methods. CRF has the lowest α value because it uses all labeled examples. Interestingly, benefiting from the informative mentions selected by LUSTRE, CRF^L have much higher α values. LUSTRE, however, has the highest α values.

	LUSTRE	CRF	CRF^L
`Person (in-domain)`	0.089	0.005	**0.090**
`Person (out-domain)`	**0.084**	0.005	**0.083**
`Company (in-domain)`	**0.125**	0.004	0.101
`Company (out-domain)`	**0.11**	0.004	0.091
`Tournament`	**0.072**	0.014	0.052
`Title`	**0.060**	0.004	0.055

Table 4: The α values of different methods

4.4 Usefulness

We present an extrinsic evaluation on entity resolution and relation extraction tasks, both of which require reasoning over name variations of entities. We implement a configurable variant generation program that uses structure of an entity to generate its name variations. It allows the user to configure a set of string transformation functions for the semantic units in a structure, which are then used to compose name variations. It supports four transformations: DROP (ignore a token), INITIAL (retain first character of a token), INITIAL$_{dot}$ (retain first character followed by a dot) and MAP (replace with user-provided string).

Entity Resolution

We use ERLearn (Qian et al., 2017), a state-of-the-art system for large-scale entity resolution (ER). ERLearn aims to learn a set of *matching rules*, each consisting of *matching functions* for different entity attributes (e.g., to determine duplicate names 'John Smith' and 'J. Smith'). The matching functions are mostly based on textual similarity of attributes. We replace these with matching functions created using LUSTRE[4]. We focus on two scenarios: (1) `Emp-Social` (matching ∼470k employee records with 50 million social network user profiles) (2) `Crystal` (de-duplicating ∼1.3 million company records). We include new matching functions for *person* names and *company* names for the two scenarios respectively.

Type	Algorithm	# links	Precision	# true links
EMP-SOCIAL	ERLearn-CIKM	1088	**0.93**	∼1015
	ERLearn-LUSTRE	**1178**	0.9	**∼1060**
CRYSTAL	ERLearn-CIKM	145,516	0.9	∼130,964
	ERLearn-LUSTRE	**147,232**	0.9	**∼132,508**

Table 5: ERLearn-CIKM vs. ERLearn-LUSTRE

As shown in Table 5, the new matching functions help ERLearn identify 4.40% and 1.17% more true links for `Emp-Social` and `Crystal` scenarios, respectively. This improvement is significant for

[4]Due to space constraints, we request readers to refer to the supplementary material for details

Emp-Social given the extreme low matching ratio for this dataset. ERLearn-CIKM already identifies a significant subset of the true links in such a sparse space that it is non-trivial for ERLearn-LUSTRE to have found additional 45 true links. In contrast, the ER task is more challenging for the Crystal scenario, with more matching functions (158 vs. 68) (Qian et al., 2017). ERLearn-LUSTRE could still identify additional 1544 true links, which is a significant improvement over 130k true links already identified by ERLearn-CIKM. [5]

Relation Extraction

We use MULTIR (Hoffmann et al., 2011), a state-of-the-art relation extractor trained on NY Times text (Riedel et al., 2010) with weak supervision from Freebase (Bollacker et al., 2008). The weak supervision data is generated by exactly matching the textual mentions to canonicalized entities in Freebase. We instead match to the variations of entities of types *Person* and *Company*, generated using the configurations in Table 6. We follow the approach of (Hoffmann et al., 2011) to generate supervision data, compute features and evaluate aggregate extraction.

For the 2 million entities, we generate 5.3 million variations. There were 24,882 sentences where textual mentions exactly matched a canonical Freebase entity and one of the specified relations existed between the entities. This increased to 34,197 sentences when named variations of entities were included, suggesting name variations are useful for entity recognition. By including the variations for only two entity types, we could generate a training data that improved the overall extractor performance (F-1 score increased by 3% from 0.485 to 0.499). The extractor could further benefit from variations for entities of other types.

Person		
⟨first⟩	INITIAL, INITIAL$_{dot}$	*Dr. R. M. Nelson*
⟨middle⟩	INITIAL, INITIAL$_{dot}$, DROP	*Dr. Russell Nelson*
⟨title⟩	DROP	*Russell Nelson II*
⟨suffix⟩	DROP	*Dr. Russell Nelson*
Company		
⟨name⟩	INITIAL, INITIAL$_{dot}$	*HP USA Inc.*
⟨suffix⟩	DROP	*Hewlett-Packard USA*
⟨location⟩	DROP	*Hewlett-Packard Inc.*

Table 6: Configurations for generating variants

5 Related Work

Exploiting the compositional structure of entities and attributes especially from query streams and text has received much attention in databases and NLP literature. It has mostly been used for understanding NL questions (Berant et al., 2013), noun-phrase queries (Li, 2010) or normalizing time expressions (Lee et al., 2014; Bethard, 2013). Consequently, structuring and linking information on the web (Bollacker et al., 2008; Auer et al., 2007) about entities and their attributes, has seen a rise in interest. With this information being automatically extracted from textual data (Fader et al., 2011; Carlson et al., 2010), reconciling variations in entities and attribute names has become an integral part of the effort. Some recent work (Halevy et al., 2016) has attempted to organize attribute names by learning their compositional structure. On the other hand, some have proposed complex normalization frameworks (D'Souza and Ng, 2015) for specific domains. However, we need methods that can learn structured representations for the large scale of entity types found on the web.

Named entities are not atomic units and often contain other entities (Finkel and Manning, 2009). However, entity resolution has relied largely on surface-level match of entity mentions (Riedel et al., 2010; Hoffmann et al., 2011; Xu et al., 2013). Variations are typically handled using similarity functions such as edit distance, jaccard similarity, which have limited customizability. While learning string transformation rules (Arasu et al., 2009; Singh and Gulwani, 2012) to reconcile variations has been studied in different contexts, it typically relies on a set of input-output examples. It is difficult to obtain such data for entities and their variations. We instead propose a different approach to first learn the internal structure of an entity and then enable configurable transformations on its structure to generate its variations.

We focus on the problem of reducing manual effort in selecting a set of representative examples for learning the regular expression patterns, which is different from the problem of synthesizing regular

[5] More results of the entity resolution experiment can be found in the appendix.

expressions from examples (Bartoli et al., 2012; Li et al., 2008).

6 Conclusion

This paper identifies a novel problem of understanding structured representations of entities for handling their name variations. We propose an active-learning based approach, LUSTRE, to iteratively learn the structured representations of an entity type from a few labeled mentions and a large set of unlabeled mentions. With small manual effort, it can learn these structured representations and automatically generate programs to map mentions to their structured representations. Reasoning over such structured representations is useful for entity resolution and relation extraction that require reasoning over name variations of entities. In the future, we plan to extend our approach to learn structures of nested entities, and use sophisticated variant generation algorithms that could rank the variations based on their reliability.

Acknowledgements

This work was supported in part by IBM under contract 4915012629 and a graduate fellowship, and by a grant from the UM Office of Research.

References

Arvind Arasu and Raghav Kaushik. 2009. A grammar-based entity representation framework for data cleaning. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 233–244. ACM.

Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. 2008. Transformation-based framework for record matching. In *Data Engineering, 2008. ICDE 2008. IEEE 24th International Conference on*, pages 40–49. IEEE.

Arvind Arasu, Surajit Chaudhuri, and Raghav Kaushik. 2009. Learning string transformations from examples. *Proceedings of the VLDB Endowment*, 2:514–525.

Sören Auer, Christian Bizer, Georgi Kobilarov, Jens Lehmann, Richard Cyganiak, and Zachary Ives. 2007. Dbpedia: A nucleus for a web of open data. In *The semantic web*, pages 722–735. Springer.

Alberto Bartoli, Giorgio Davanzo, Andrea De Lorenzo, Marco Mauri, Eric Medvet, and Enrico Sorio. 2012. Automatic generation of regular expressions from examples with genetic programming. In *Proceedings of the 14th annual conference companion on Genetic and evolutionary computation*, pages 1477–1478. ACM.

Jonathan Berant, Andrew Chou, Roy Frostig, and Percy Liang. 2013. Semantic parsing on freebase from question-answer pairs. In *EMNLP*, volume 2, page 6.

Steven Bethard. 2013. A synchronous context free grammar for time normalization. In *EMNLP*, pages 821–826.

Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM.

Adriano Crestani Campos, Yunyao Li, Sriram Raghavan, and Huaiyu Zhu. 2015. Entity variant generation and normalization, June 23. US Patent 9,063,926.

Andrew Carlson, Justin Betteridge, Bryan Kisiel, Burr Settles, Estevam R Hruschka Jr, and Tom M Mitchell. 2010. Toward an architecture for never-ending language learning. In *AAAI*, volume 5, page 3.

Peter Christen. 2012. A survey of indexing techniques for scalable record linkage and deduplication. *IEEE transactions on knowledge and data engineering*, 24(9):1537–1555.

Aron Culotta and Andrew McCallum. 2005. Reducing labeling effort for structured prediction tasks. In *AAAI*, volume 5, pages 746–751.

Mark Dredze, Paul McNamee, Delip Rao, Adam Gerber, and Tim Finin. 2010. Entity disambiguation for knowledge base population. In *COLING*.

Jennifer D'Souza and Vincent Ng. 2015. Sieve-based entity linking for the biomedical domain. In *ACL (2)*, pages 297–302.

Ahmed K Elmagarmid, Panagiotis G Ipeirotis, and Vassilios S Verykios. 2007. Duplicate record detection: A survey. *IEEE Transactions on knowledge and data engineering*, 19(1):1–16.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics.

Jenny Rose Finkel and Christopher D Manning. 2009. Nested named entity recognition. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing: Volume 1-Volume 1*, pages 141–150. Association for Computational Linguistics.

Luis Galárraga, Geremy Heitz, Kevin Murphy, and Fabian M Suchanek. 2014. Canonicalizing open knowledge bases. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 1679–1688. ACM.

Alon Halevy, Natalya Noy, Sunita Sarawagi, Steven Euijong Whang, and Xiao Yu. 2016. Discovering structure in the universe of attribute names. In *Proceedings of the 25th International Conference on World Wide Web*, pages 939–949. International World Wide Web Conferences Steering Committee.

Xianpei Han, Le Sun, and Jun Zhao. 2011. Collective entity linking in web text: a graph-based method. In *Proceedings of the 34th international ACM SIGIR conference on Research and development in Information Retrieval*, pages 765–774. ACM.

Raphael Hoffmann, Congle Zhang, Xiao Ling, Luke Zettlemoyer, and Daniel S Weld. 2011. Knowledge-based weak supervision for information extraction of overlapping relations. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 541–550. Association for Computational Linguistics.

Kenton Lee, Yoav Artzi, Jesse Dodge, and Luke Zettlemoyer. 2014. Context-dependent semantic parsing for time expressions. In *ACL (1)*, pages 1437–1447.

John Lehmann, Sean Monahan, Luke Nezda, Arnold Jung, and Ying Shi. 2010. Lcc approaches to knowledge base population at tac 2010. In *TAC*.

Yunyao Li, Rajasekar Krishnamurthy, Sriram Raghavan, Shivakumar Vaithyanathan, and HV Jagadish. 2008. Regular expression learning for information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 21–30. Association for Computational Linguistics.

Xiao Li. 2010. Understanding the semantic structure of noun phrase queries. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, pages 1337–1345. Association for Computational Linguistics.

Xiaohua Liu, Yitong Li, Haocheng Wu, Ming Zhou, Furu Wei, and Yi Lu. 2013. Entity linking for tweets. In *ACL (1)*, pages 1304–1311.

Andrew McCallum, Kamal Nigam, and Lyle H Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 169–178. ACM.

Ndapandula Nakashole, Martin Theobald, and Gerhard Weikum. 2011. Scalable knowledge harvesting with high precision and high recall. In *Proceedings of the fourth ACM international conference on Web search and data mining*, pages 227–236. ACM.

Kun Qian, Lucian Popa, and Prithviraj Sen. 2017. Active learning for large-scale entity resolution. In *Proceedings of the 2017 ACM on Conference on Information and Knowledge Management*, CIKM '17, pages 1379–1388, New York, NY, USA. ACM.

Kun Qian, Nikita Bhutani, Yunyao Li, H. Jagadish, and Hernandez Mauricio. 2018. Lustre: An interactive system for entity structured representation and variant generation. In *Data Engineering (ICDE), 2018 IEEE 34th International Conference on*. IEEE.

Sebastian Riedel, Limin Yao, and Andrew McCallum. 2010. Modeling relations and their mentions without labeled text. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 148–163. Springer.

Burr Settles and Mark Craven. 2008. An analysis of active learning strategies for sequence labeling tasks. In *Proceedings of the conference on empirical methods in natural language processing*, pages 1070–1079. Association for Computational Linguistics.

Wei Shen, Jianyong Wang, and Jiawei Han. 2015. Entity linking with a knowledge base: Issues, techniques, and solutions. *IEEE Transactions on Knowledge and Data Engineering*, 27(2):443–460.

Rishabh Singh and Sumit Gulwani. 2012. Learning semantic string transformations from examples. *Proceedings of the VLDB Endowment*, 5:740–751.

Christopher Walker, Stephanie Strassel, Julie Medero, and Kazuaki Maeda. 2006. Ace 2005 multilingual training corpus. *Linguistic Data Consortium, Philadelphia*, 57.

Wei Xu, Raphael Hoffmann, Le Zhao, and Ralph Grishman. 2013. Filling knowledge base gaps for distant supervision of relation extraction. In *ACL (2)*, pages 665–670.

Wei Zhang, Chew Lim Tan, Yan Chuan Sim, and Jian Su. 2010. Nus-i2r: Learning a combined system for entity linking. In *TAC*.

Zhicheng Zheng, Fangtao Li, Minlie Huang, and Xiaoyan Zhu. 2010. Learning to link entities with knowledge base. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 483–491. Association for Computational Linguistics.

7 Appendix A. Experimental Setup of Entity Resolution

Our entity resolution experiments use ER-Learn (Qian et al., 2017) on two scenarios `Emp-Social` and `Crystal`. We aim to provide it supplementary matching functions that reason over structured representations of entity names. To learn structured representations for the `Emp-Social` scenario, we randomly sample 1000 person names from the *Emp* records and 1000 person names from the online social network data. Using LUSTRE over the 2000 names, we learn the structures for *Person* with five different semantic units: $\langle first \rangle$, $\langle last \rangle$, $\langle middle \rangle$, $\langle suffix \rangle$, and $\langle title \rangle$. For the `Crystal` scenario, we learn structures for *Company* names using a sample of 2000 names. These structures have semantic units $\langle name \rangle$, $\langle industry \rangle$, $\langle suffix \rangle$, and $\langle location \rangle$.

Person		
⟨first⟩	INITIAL, INITIAL$_{dot}$	*Dr. R. M. Nelson*
⟨last⟩	INITIAL, INITIAL$_{dot}$	*Dr. Russell N.*
⟨middle⟩	INITIAL, INITIAL$_{dot}$, DROP	*Dr. Russell Nelson*
⟨title⟩	DROP	*Russell Nelson II*
⟨suffix⟩	DROP	*Dr. Russell Nelson*
ORDER	⟨title⟩⟨first⟩⟨middle⟩⟨last⟩⟨suffix⟩	
Company		
⟨name⟩	INITIAL, INITIAL$_{dot}$	*GM U.S. Trading Corp.*
⟨industry⟩	DROP	*General Motors U.S. Corp.*
⟨suffix⟩	DROP	*General Motors U.S.*
⟨location⟩	DROP	*General Motors Trading Corp.*
ORDER	⟨name⟩⟨industry⟩⟨suffix⟩⟨location⟩	

Table 7: Configurations with examples

To design matching functions that reason over structured representations to identify duplicates, we refer to the name variations of mentions generated using their representations. Intuitively, duplicate mentions are likely to share many name variations. Given a mention, we first generate its variations using the configurations in Table 7. We keep both the original string and the transformed string for a semantic unit modified using an operator from the configuration. For instance, dropping the title in a person name would generate Dr. Russell Nelson Sr. and Russell Nelson Sr. as variations of Dr. Russell Nelson Sr.. We define matching functions to identify two mentions as duplicates if they have at least r common variations.

matchPerson(Emp.name, social.name, r),
matchCompany(cp1.name, cp2.name, r),
matchCompany(cp1.parent.name, cp2.parent.name, r),

where $r = 1, 2, \ldots, 20$. These matching functions complement the matching functions used in the original study (Qian et al., 2017). We compare the ER rules learned by ERLearn in two different configurations, ERLearn-LUSTRE and ERLearn-CIKM, with and without the new matching functions respectively. The rules learned by ERLearn-LUSTRE had similar constituent matching functions as rules from ERLearn-CIKM. However, we found they additionally included the new matching functions and had relaxed some of the matching functions. For illustration, consider the following rules:

```
match Emp i, Social s By R:
  matchPerson(i.Name, s.name, 9)
  AND i.name.last = s.name.last
  AND lastNameFreqFilter(s.name.last, 50%)
  AND sameCity(i.CITY, s.city)
  AND countryIsInUS(i.COUNTRY, s.country)
```

(a) ERLearn-LUSTRE

```
match Emp i, Social s By R:
  nameMatch(i.Name.firstNameVars,s.name.first)
  AND i.Name.last = s.name.last
  AND lastNameFreqFilter(s.name.last, 60)
  AND upperCase(i.CITY)= upperCase(s.home.city)
  AND countryIsInUSA(i.COUNTRY)
```

(b) ERLearn-CIKM

There are to key differences: (1) the matching function for first names now uses the new matching function *matchPerson*, (2) the threshold value used in *lastNameFreqFilter* decreased from 60% to 50%. In contrast to the original rules, the lower threshold value for last names in the new rule makes it less conservative, potentially increasing the risk of identifying incorrect links. However, the matching function *matchPerson* makes the rule less susceptible to over-generalization. We found that by including matching functions that exploit the structured representations of entities, ERLearn could learn an ER model with appropriate generalization.