Provenance-based Dictionary Refinement in Information Extraction

Sudeepa Roy[*]
University of Washington
sudeepa@cs.washington.edu

Laura Chiticariu
IBM Research-Almaden
chiti@us.ibm.com

Vitaly Feldman
IBM Research-Almaden
vitaly@post.harvard.edu

Frederick R. Reiss
IBM Research-Almaden
frreiss@us.ibm.com

Huaiyu Zhu
IBM Research-Almaden
huaiyu@us.ibm.com

ABSTRACT

Dictionaries of terms and phrases (e.g. common person or organization names) are integral to information extraction systems that extract structured information from unstructured text. Using noisy or unrefined dictionaries may lead to many incorrect results even when highly precise and sophisticated extraction rules are used. In general, the results of the system are dependent on dictionary entries in arbitrary complex ways, and removal of a set of entries can remove both correct and incorrect results. Further, any such refinement critically requires laborious *manual labeling* of the results.

In this paper, we study the *dictionary refinement problem* and address the above challenges. Using *provenance* of the outputs in terms of the dictionary entries, we formalize an optimization problem of maximizing the quality of the system with respect to the refined dictionaries, study complexity of this problem, and give efficient algorithms. We also propose solutions to address incomplete labeling of the results where we estimate the missing labels assuming a statistical model. We conclude with a detailed experimental evaluation using several real-world extractors and competition datasets to validate our solutions. Beyond information extraction, our provenance-based techniques and solutions may find applications in view-maintenance in general relational settings.

Categories and Subject Descriptors

H.0 [**Information Systems**]: General; I.7.0 [**Computing Methodologies**]: Document and Text Processing—*General*

Keywords

Information Extraction, F-score, Provenance, Optimization

1. INTRODUCTION

Information Extraction, the problem of extracting structured information from unstructured text, is an essential component of many important applications including business intelligence, social media analytics, semantic search and regulatory compliance. The success of these applications is tightly connected with the quality of the extracted results, as incorrect or missing results may often render the application useless. Most information extraction systems use a set of *rules*, a number of *dictionaries* of terms and phrases (also known as *gazetteers* or *lists*) along with other basic features like *syntactic features* (*e.g.* regular expressions) and *morphological features* (*e.g.* part of speech) to identify common patterns in text; these are subsequently used to extract *entities* (*e.g.* Person, Organization, Location) and *relations* between entities (*e.g.* Person's birth date or phone number) from the text.

Developing and maintaining high-quality entity or relation extractors is an extremely laborious process. Typically, a developer starts by collecting an initial set of features and developing an initial set of rules. She then executes the extractor on a document collection, labels some of the results as *correct* (or, a *true positive*) and *incorrect* (or, a *false positive*), examines the causes of incorrect results, and refines the extractor in order to remove these incorrect results. This process is repeated until the developer is satisfied with the quality of the extractor. In this paper, we investigate in detail one of the most important parts of this process, that of *refining dictionaries* by suggesting to the human supervisor of the system a list of entries whose removal from the dictionaries will remove many false positives in the output without removing too many true positives. The supervisor may then decide to either: (1) delete these dictionary entries altogether, or (2) list them as ambiguous entries and move them to other dictionaries, or (3) update the rules in order to handle these entries appropriately.

Dictionaries are integral to information extraction systems [10, 9, 22]. For example, a Person extractor would make use of dictionaries of complete names (*e.g.* names of famous persons), as well as dictionaries of common first names and last names, and dictionaries of common titles (*e.g.* "Mr.", "Esq."). More sophisticated Person extractors may use a larger collection of more fine-grained dictionaries to improve accuracy (*e.g.* a dictionary of common first names that are also the names of major U.S. corporations). Dictionaries are even more relevant in the context of today's informal data sources (*e.g.* social media), which do not obey the constraints of formal language: syntactic features such as capitalization and punctuation are rarely present, and classic part of speech analyzers trained on formal text such as TreeBank have low accuracy [27].

Although many dictionaries are readily available from public or commercial sources, (*e.g.* the U.S. Census Bureau [1] provides extensive lists of first and last names, while [2] provides lists of locations containing millions of entries with detailed geographical information), inherent ambiguity in language prevents such exhaus-

[*]This work was done while the author was at the University of Pennsylvania and IBM Research-Almaden.

tive dictionaries from being consumed directly, as they would lead to many false positives. Instead, these dictionaries require curation before they can be used effectively in an extractor, and the extent of the curation depends on the application, domain and language. For example, it has been observed that an extensive dictionary of location names is useful, whereas an extensive dictionary of person/organization names is not that effective [30, 21]. Dictionary entries are often collected from noisy sources [24, 31], or, generated automatically by a previous step of information extraction [33, 22]. One may frequently encounter in dictionaries of person names ambiguous entries such as "San", "Francisco", "Hong", "Kong", "April", "Costa" that mostly produce false positives. These entries must be either removed or treated differently in order to improve the quality of the system.

In addition to refining noisy dictionaries, it is important to derive dictionaries with high precision tailored to a particular domain of interest by refining more general dictionaries. For instance, there may be inherent overlap between dictionaries for different entities ("Chelsea" is a common first name, but also the name of a famous football club; "Victoria" or "Washington" can be a location or a person). While including "Chelsea" in a dictionary of names may work well in general, when the system is customized to process a corpus of sports news articles, we need to add a rule that can disambiguate "Chelsea". For example, the supervisor may decide to remove "Chelsea" from the dictionary of first names, and at the same time add "Chelsea" to a new dictionary of ambiguous names, along with a new rule that marks occurrences of ambiguous names as candidate persons only if another strong contextual (e.g., a title) clue is present in the nearby text. However, prior to adding the new rule, one must first determine the ambiguous entries in the dictionary. In practice, the decision whether to refine dictionaries or to add more sophisticated rules to directly handle all possible scenarios is usually carefully drawn on a case by case basis, by considering not only the overall improvement in accuracy, but also the ease of maintainability of the resulting extractor. Maintaining complex rules in general is a labor intensive process requiring expertise in the system's rule language. Building high precision dictionaries, especially for domain-specific applications, provides a low-overhead option requiring little or no knowledge of the system, and helps create, maintain and update extractor rules to further improve the quality of the system [26, 9, 17, 28].

To refine a dictionary, the system's supervisor would wish to examine a list of entries whose removal would lead to a significant improvement of the system by removing many false positives. This raises two technical challenges. *First*, a realistic extractor usually uses a number of dictionaries that are combined via complex rules. An output is determined by alternative or joint uses of multiple dictionary entries, and their arbitrary combinations. Therefore, when an entry is removed, some of the correct and incorrect results it has produced may get deleted and some may be retained depending on the other entries being removed. We need to consider the actual connection between dictionary entries and output results while refining the dictionaries. *Second*, *labeling* the outputs of extractors (as true and false positives) is a laborious task requiring substantial human effort. In addition, dictionaries frequently contain thousands of entries, therefore information about individual entries is sparse even when a large amount of labeled data is available. Nevertheless, one should ensure that the refined dictionaries perform well with respect to unlabeled data.

Our contribution. We systematically study the dictionary refinement problem in this paper. Note that in a Person extractor, if the entry "Chelsea" produces 15 true positives and 15 false positives, while "April" produces only 1 true positive and 5 false positives, it is not obvious which of these two entries should be removed first. Therefore, to balance the requirements of extraction *precision* (minimize false positives) and *recall* (avoid discarding correct answers), we maximize the standard objective of **F-score** (the harmonic mean of precision and recall) [38]. We study this maximization problem under two natural constraints that a human supervisor is likely to use: a limit on the number of dictionary entries to remove (*size constraints*), and the maximum allowable decrease in recall (*recall constraints*). The complex connection between dictionary entries and output results is captured using *provenance* of the results in terms of the entries as a boolean expression (which we call *provenance polynomials*) [19]. These polynomials are used to find the results that are deleted when a set of entries is removed.

To handle incomplete labeling of the results, we divide the dictionary refinement problem into two sub-problems: (a) *Label estimation* estimates the "fractional" labels of the unlabeled results assuming a statistical model for the labels. (b) *Refinement optimization* takes the (exact or estimated) labels of the results as input, and outputs a set of dictionary entries to remove in order to maximize the resulting F-score under size or recall constraint.

1. For **label estimation**, we give a method based on the well-known *Expectation - Maximization* (EM) algorithm [14]. Under our statistical model, we show that there is a closed-form expression for the update rules in EM for our application which can be efficiently evaluated.

2. To understand the complexity of the **refinement optimization** problem, we start with an important special case, called *single argument rules* (or, *simple rules*), where a *unique* dictionary entry is responsible for producing each output result. Then we move on to the case of general *multiple arguments rules* (or, *complex rules*). Besides serving as a stepping stone for complex rules, the case of simple rules has several practical applications including the initial refinement of a noisy dictionary generated from various sources, and the curation of specialized high-precision dictionaries (such as lists of organizations in healthcare and banking).

   The complex objective function of maximizing the F-score makes the refinement optimization step non-trivial even for simple rules, and even when the entire set of results is labeled. However, we give a poly-time optimal algorithm for simple rules under size constraints. The problem becomes NP-hard under recall constraints even for simple rules. Nevertheless, we show that an efficient near-optimal algorithm exists. In addition, we show that the estimated labels reduce to empirical estimates for simple rules, and therefore the label estimation step can be skipped in this case. For complex rules, the optimization problem is NP-hard even under size constraints and we propose efficient heuristics.

3. We conduct an extensive set of **experiments** on several real-world information extraction rule-sets and competition datasets that validate the effectiveness of our techniques. The complexity of the extractor rules varies from one rule (for the simple rule case) to a highly refined extractor using over 400 rules (for the complex rule case). This demonstrates that our techniques are useful for both doing the initial refinement of a noisy dictionary and also to further improve the quality of an extractor that uses a sophisticated rule set.

Beyond information extraction, our approach has applicability in **view maintenance in general relational settings**. Our model and theoretical results can be used to refine erroneous source tuples in order to reduce the number of false positives in the output view without removing too many true positives (see Section 2).

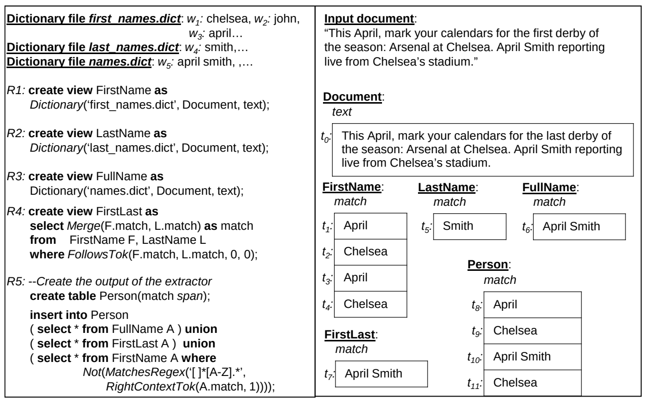
Roadmap. We start with related work and preliminaries (Sections 2 and 3), and formalize the dictionary refinement problem in Section 4. Sections 5 and 6 present our technical contributions on estimation of labels for unlabeled results, and respectively, refinement optimization for simple and complex rules. We discuss our experimental results in Section 7 and conclude with directions for future work in Section 8.

2. RELATED WORK

Previous work on entity extraction from lists on the web (*e.g.* [15]), open information extraction (*e.g.* [4, 39]) and dictionary learning (*e.g.* [33]) addresses the problem of creating new dictionaries of terms from unstructured or semi-structured data sources by exploiting contextual patterns or the structure of web pages. Our work is complementary to these approaches, and can be used to further refine such automatically generated dictionaries. Our work is also orthogonal to *Named Entity Disambiguation* techniques (*e.g.* [20]), which, given an external catalog of entities, aim to associate an extracted named entity with an actual entity in the catalog. The goal of our techniques for complex rules is to find out entries that produce many false positives during the extraction process, so that the human supervisor can either remove them or update the rules. Our techniques are applicable not only to Named Entity Recognition tasks, but also other common tasks such as relationship or sentiment extraction; furthermore, they can be applied when the application cannot make use of an external catalog, for example due to license limitations or when speed of execution is critical.

Approaches for refining rule-based information extraction programs have been recently proposed in [34, 7, 26]. Shen et al. [34] propose an approach for refining rules by posing a series of template questions to the user, where each question asks for additional information about a specific (predefined) feature of the desired extracted data, whereas Chai et al. [7] allow users to update any (incorrect) intermediate result derived by the system and proposes techniques for incorporating these updates during program execution. In contrast, we develop techniques to automatically compute a (small) set of dictionary entries, therefore allowing the user to focus on a (small) set of base tuples whose removal results in highest quality improvements for the extractor. Liu et al. [26] proposed a provenance-based framework for refining information extraction rules. They showed how to use provenance to compute *high-level changes*, a specific intermediate result whose removal from the output of an operator causes the removal of a false positive from the result, and how multiple high-level changes can be realized via a *low-level change*: a concrete change to the operator that removes one or more intermediate results from the output of the operator. In this paper, we do rigorous theoretical analysis of one specific and important type of low-level change, that of refining dictionaries used in an extractor, especially in the presence of incomplete labeled data, which is an aspect not considered in [26].

Also related are recent studies on *causality* [29] and *deletion propagation* [6, 23] in general relational setting. In [29], the input consists of source tuples, an output boolean vector and a ground truth boolean vector, and the goal is to compute the *responsibility* of every source tuple using modifications such that the output is identical to the ground truth. In other words, the modifications should remove all incorrect results, while keeping all correct results. On the other hand, [6, 23] study the problem of deleting an incorrect answer tuple of an SPJU query while minimizing the *view-side-effect*, i.e. the number of other answer tuples deleted. We consider extractors consisting essentially of SPJU queries, where dictionary entries correspond to source tuples and extraction results correspond to outputs. As such, our work can be seen as an alternative objective for these related problems that tries to balance between the number of incorrect tuples retained and the number of correct tuples deleted. To the best of our knowledge, ours is the first formal study of dictionary refinement for information extraction with a well-defined objective of maximizing the F-score.



Dictionary file *first_names.dict*: w_1: chelsea, w_2: john, w_3: april…
Dictionary file *last_names.dict*: w_4: smith,…
Dictionary file *names.dict*: w_5: april smith, ,…

```
R1: create view FirstName as
     Dictionary('first_names.dict', Document, text);

R2: create view LastName as
     Dictionary('last_names.dict', Document, text);

R3: create view FullName as
     Dictionary('names.dict', Document, text);

R4: create view FirstLast as
     select Merge(F.match, L.match) as match
     from   FirstName F, LastName L
     where FollowsTok(F.match, L.match, 0, 0);

R5: --Create the output of the extractor
     create table Person(match span);

     insert into Person
     ( select * from FullName A ) union
     ( select * from FirstLast A )  union
     ( select * from FirstName A where
                Not(MatchesRegex('[ ]*[A-Z].*',
                     RightContextTok(A.match, 1))));
```

Input document:
"This April, mark your calendars for the first derby of the season: Arsenal at Chelsea. April Smith reporting live from Chelsea's stadium."

Document:

	text
t_0:	This April, mark your calendars for the last derby of the season: Arsenal at Chelsea. April Smith reporting live from Chelsea's stadium.

FirstName:

	match
t_1:	April
t_2:	Chelsea
t_3:	April
t_4:	Chelsea

LastName:

	match
t_5:	Smith

FullName:

	match
t_6:	April Smith

FirstLast:

	match
t_7:	April Smith

Person:

	match
t_8:	April
t_9:	Chelsea
t_{10}:	April Smith
t_{11}:	Chelsea

Figure 1: Example extraction program, input document D, and view instances created by the extraction program on D.

3. PRELIMINARIES

We illustrate the main components of a rule-based information extraction system in Figure 1 using a simplified Person extractor. An information extraction system typically consists of a set of *rules* (R_1 to R_5), and a set of *dictionaries* ("first_names.dict", "last_names.dict" and "names.dict"). Given one or a set of *input documents* (often called a *corpus*), such a system extracts the entities or relations specified in the *extractor* (view instance *Person* in Figure 1). In practice, the extractors are much more complex; one of the extractors used in our experiments uses more than 400 rules.

Rule Language. We use the *select-project-join-union* (SPJU) subset of SQL enriched with a number of basic extraction primitives to express information extraction rules[1]. This subset expresses a core set of functionality underlying most rule languages in common use today [5, 12, 25, 32, 35], and therefore our work can be easily applied to other rule languages. Specifically, we augment SQL with the following basic information extraction primitives. To model data values extracted from the input document we add a new atomic data type called *span*. A *span* is an ordered pair $\langle begin, end \rangle$ that identifies the region of an input document between the *begin* and *end* offsets. The input document is modeled as a table called Document with a single attribute of type *span* named *text*. We also add several predicates, scalar functions, and table functions to SQL's standard set of built-in functions as discussed below.

Example Rules. The program in Figure 1 consists of five rules labeled R_1 through R_5. Rules R_1 to R_4 define logical views, while R_5 materializes a table of extraction results. Rules R_1, R_2 and R_3 illustrate one of our text extraction additions to SQL: the *Dictionary* table function, which identifies all occurrences of a given set of terms specified as entries in a dictionary file. The dictionary files used in R_1 to R_3 contain a list of common first names, a list of common last names, and respectively a list of common full names. The three rules define three single-column views *FirstName*, *LastName* and *FullName* containing one intermediate tuple for each dictionary match in the document. We use w_i to denote dictionary entries

[1]The complex extractor used in our experiments also contains set difference operator (SQL minus) which we discuss in Section 7.1.

in order to distinguish them from their (possibly multiple) matches in a document (*e.g.* w_1 generates the intermediate tuples t_2 and t_4).

Rule R_4 identifies pairs of first and last names that are 0 tokens apart in the input document. The view definition uses two of the scalar functions that we add to SQL: *FollowsTok* and *Merge*. The *FollowsTok* function is used as a join predicate: it takes two spans as arguments, along with a minimum and maximum character distance, and returns *true* if the spans are within the specified distance of each other in the text. The *Merge* function takes as input a pair of spans and returns the shortest span that completely covers both input spans. (*Merge* is sensitive to the order of its input spans: if the second span appears before the first, the result of is undefined.) The *select* clause of R_4 uses *Merge* to define a span that extends from the beginning of each first name to the end of the last name.

Finally, rule R_5 materializes the table Person, which constitutes the output of our extractor. It uses a *union* clause to union together candidate name spans identified by rules R_3 and R_4, as well as candidates identified by R_1 that are not immediately followed by a capitalized word. Note the *where* clause of the last union operand of R_5 which uses two other additions to SQL: the *RightContextTok* function returns the span of a given length measured in tokens to the right of the input span, while the *MatchesRegex* predicate returns true if the text of the input span matches the given regular expression. In this case, the regular expression `‘[ ]*[A-Z].*’` identifies a sequence of zero or more whitespaces, followed by an uppercase letter, followed by zero or more occurrences of any character.

Provenance of results in terms of dictionary entries. Following the existing work on data provenance, we assume a canonical algebraic representation of extraction rules as trees of operators. For the *select-from-where-union* subset of the language, the canonical representation is essentially the same as the representation of corresponding SQL statements in terms of relational operators σ, π, $\bowtie$ and $\cup$. When an extraction function such as *Dictionary* appears in the *from* clause of a *select* statement, we translate these table functions to operators by the same names.

The canonical rule representation enables us to express the *provenance* (or *lineage*) of extraction results. Provenance for database queries has been intensely studied in recent years (see, *e.g.* [8]). In this work, we use the notion of *how-provenance* to represent how an extraction result has been generated by the rules. Specifically, using the semiring framework of [19] we can encode the how-provenance of an output tuple t, denoted as $\texttt{Prv}(t)$, that describes how that output tuple has been derived from source or intermediate tuples using joins (multiplication $\cdot$) and union (addition $+$) operations; *e.g.* $\texttt{Prv}(t_{10}) = t_6 + t_7 = t_6 + t_3 \cdot t_5$, since $\texttt{Prv}(t_7) = t_3 \cdot t_5$. We call $\texttt{Prv}(t)$ the as *provenance polynomial* of t. We regard individual dictionary entries as the source of intermediate tuples generated by the *Dictionary* operator and express the provenance of an output tuple directly in terms of the entries that are responsible for generating that tuple. Since $\texttt{Prv}(t_3) = w_3$, $\texttt{Prv}(t_5) = w_4$ and $\texttt{Prv}(t_6) = w_5$, we have that $\texttt{Prv}(t_{10}) = w_5 + w_3 \cdot w_4$.

Simple and Complex Rules. To understand the complexity of the dictionary refinement problem, we classify the input extractors as (1) *single argument rules* (or, *simple rules*), and (2) *multiple arguments rules* (or, *complex rules*). Simple rules contain a single *extract dictionary* statement involving single or multiple dictionaries, whereas the extractor for complex rules can consist of a number of arbitrary rules involving one or more dictionaries (like the example in Figure 1). An example of a simple rule is

```
CREATE VIEW Person AS
EXTRACT DICTIONARY 'name_USA.dict' and
'name_German.dict' ON D.text AS name
FROM DOCUMENT D; OUTPUT VIEW Person;
```

Here a *unique* dictionary entry is responsible for producing each output result and the provenance polynomial of a result tuple t is simply of the form $\texttt{Prv}(t) = w$. Besides being a fundamental special case for complex rules, the case of simple rules is of independent interest. Our solution for simple rules may be useful in initially refining manually or automatically generated dictionaries that can be used as basic features not only in rule-based, but also statistical (machine learning) information extraction systems. Other applications include the curation of specialized high-precision dictionaries (such as lists of organizations in healthcare and banking) and to create and refine complex extractor rules.

Precision, Recall and F-score. We now explain the standard measures for evaluating an extractor's quality, given a labeled dataset containing expected results for the extraction. With respect to the labeled dataset, an extracted result is either *a true positive* (correct) or a *false positive* (incorrect). The *precision*, or accuracy of the extractor is defined as the fraction of true positives among the total number of extracted results. An extractor with high precision outputs very few false positives. An expected mention that is not identified by the extractor is called a *missing result* (false negative). The *recall*, or coverage of the extractor is defined as the fraction of true positives among the total number of expected results. An extractor with high recall misses very few expected results. Finally, the standard *F-score*, also known as F_1*-score* or *F-measure* [38], combines precision and recall into a single measure computed as the harmonic mean of precision and recall ($2PR/(P+R)$).

4. DICTIONARY REFINEMENT PROBLEM

Let E be an extractor and let A be the set of all dictionary entries used by E (union of all dictionary). Given a set of documents $\mathbb{D}$, E produces a set of results, some of which are true positives (`Good`) and some are false positives (`Bad`). When a set of entries $S \subseteq A$ is removed from A, it results in another extractor E', which on the same set of documents $\mathbb{D}$ will produce a subset of the earlier results. Let $\bar{S} = A \setminus S$. Then the precision $P_{\bar{S}}$ and recall $R_{\bar{S}}$ of E' are:

$$P_{\bar{S}} = \frac{\text{No. of } \texttt{Good} \text{ results with } \bar{S}}{\text{No. of } \texttt{Good} \text{ results + No. of } \texttt{Bad} \text{ results (with } \bar{S})} \quad (1)$$

$$R_{\bar{S}} = \frac{\text{No. of } \texttt{Good} \text{ results with } \bar{S}}{\text{No. of } \texttt{Good} \text{ results with } A} \quad (2)$$

It is not hard to see that the recall of E' will be at most that of E (i.e. 1), whereas the precision and therefore the F-score of E' can be more or less than that of E depending on the set S. For instance, in the output Person table of Figure 1, all results except "April Smith" are `Bad`, therefore the initial precision, recall and F-scores are $\frac{1}{4}, 1$ and $\frac{2}{5}$ respectively. If the entry w_1 = "Chelsea" is removed from the dictionary *first_names.dict*, the recall will remain the same, whereas the precision and F-score will improve to $\frac{1}{2}$ and $\frac{2}{3}$ respectively. The goal of the *dictionary refinement problem* is to compute a subset S whose removal results in an extractor E' having the maximum value of the F-score on $\mathbb{D}$.

4.1 Estimating Labels of Results for Incomplete Labeling

Even computing the F-score of an extractor requires knowing labels (`Good`/`Bad`) on the entire set of results, while often only a small fraction of the results is labeled. One possible approach is to ignore the unlabeled results altogether and try to maximize the F-score using the labeled results alone. But this may lead to overfitting and the solution may not work well for the entire result set. Estimating the labels empirically is a natural choice for simple rules (where each result is uniquely determined by an entry). However, the number of possible results may be very large for complex rules, and very few or zero labels are likely to be available for each result. This makes empirical estimates expensive as well as difficult. In Section 5, we propose a statistical model and give algorithms to estimate missing labels in the dataset.

4.2 Formal Problem Definition

We are given b dictionaries $A_1, \ldots, A_b$ and let n denote the total number of entries in $A = \cup_{\ell=1}^{b} A_\ell$. A result τ is produced by matches of one or more dictionary entries combined by the rules in the extractor; all such dictionary entries w are said to be in *provenance* of τ. How the entries produce τ is captured by the provenance polynomial $\texttt{Prv}(\tau)$ of τ (see Section 3); for all such entries w we say that $w \in \texttt{Prv}(\tau)$. Here every entry in A is treated as a unique boolean variable, and $\texttt{Prv}(\tau)$ is a boolean expression on these variables.

For simple rules, every result τ produced by an entry w has $\texttt{Prv}(\tau) = w$; when w is deleted exactly those results get deleted. However, for complex rules, some results τ such that $w \in \texttt{Prv}(\tau)$ can disappear when w is deleted, while some results may survive. In Figure 1, when w_3 = "april" is deleted, the result t_8 = "April" gets deleted, but t_{10} = "April Smith" survives (although $w_3 \in \texttt{Prv}(t_{10}) = w_5 + w_3 \cdot w_4$). This illustrates the following observation:

OBSERVATION 1. *When $S \subseteq A$ is removed, a result τ disappears from the result set if and only if its provenance polynomial* $\texttt{Prv}(\tau)$ *evaluates to* FALSE *by an assignment of* FALSE *(resp.* TRUE*) value to the variables corresponding to the entries in S (resp. $A \setminus S$).*

Let $\texttt{surv}(S)$ denote the set of results that survive when a set of entries S is deleted. For example, given three results τ_1, τ_2, τ_3 with provenance polynomials $uv, u+v, uw+uv$ respectively, when $S = \{u\}$ is deleted, the set $\texttt{surv}(S)$ will only contain τ_2. Let $\phi(\tau)$ denote the *label* for a result τ. When the entire result set is labeled, $\phi(\tau) = 1$ if τ is `Good`, and, = 0 if τ is `Bad`. Then rewriting (1) and (2), the "residual" precision ($P_{\bar{S}}$), recall ($R_{\bar{S}}$) and their harmonic mean F-score ($F_{\bar{S}}$) when S is deleted respectively are

$$P_{\bar{S}} = \frac{\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{|\texttt{surv}(S)|}, \quad R_{\bar{S}} = \frac{\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{\sum_\tau \phi(\tau)},$$
$$F_{\bar{S}} = \frac{2\sum_{\tau \in \texttt{surv}(S)} \phi(\tau)}{|\texttt{surv}(S)| + \sum_\tau \phi(\tau)} \qquad (3)$$

For incomplete labeling, we extend the above definitions by allowing *fractional labels* $\phi(\tau)$, which intuitively denote the confidence that the label of the result is `Good`.

The refinement of an extractor is commonly done with human supervision. The supervisor may prefer to examine a small number of entries at a time, or may want to ensure that not too many original true positives are removed by the refinement. Therefore, we maximize the F-score under (1) **Size constraint:** Given an integer $k \leq n$, find a subset S, that maximizes $F_{\bar{S}}$, where $|S| \leq k$; and (2) **Recall constraint:** Given a fraction $\rho \leq 1$, find a subset S, that maximizes $F_{\bar{S}}$, where the residual recall $R_{\bar{S}} \geq \rho$. We study these optimization problems in Section 6. Note that both numerator and the denominator of $F_{\bar{S}}$ depend on the set S being removed, which makes the optimization and analysis non-trivial even for simple rules.

5. ESTIMATING LABELS

In this section we discuss our statistical data model and algorithms to estimate the missing labels.

5.1 The Statistical Model

A possible approach to estimate the missing labels is to group together the results with equal provenance (equivalent boolean expressions), and then estimate the labels empirically. The problem with this approach is that it is expensive, and it is likely that very few (if any) labels will be available for each result since the possible number of such expressions may be very large. At the same time it is quite likely that the correctness of individual entries will be similar across results with different provenance expressions. For example, the candidate last name "'Island"[2] can produce more than one `Bad` results like "Victoria Island" and "Baffin Island", and "Island" is a `Bad` *match*[3] as a last name in both results.

We represent this intuition by defining the model in the following way. We assume that each entry w has a fixed (and unknown) *entry-precision* p_w. For any given result τ such that $w \in \texttt{Prv}(\tau)$, the *match* of w for τ is correct with probability p_w and incorrect with probability $1 - p_w$ independent of the other results and other entries in $\texttt{Prv}(\tau)$. Further, we assume that the rules are correct, *i.e.* the label of τ is `Good` if and only if $\texttt{Prv}(\tau)$ evaluates to 1 given the correctness of the matches of individual entries (`Good` $\equiv 1$ and `Bad` $\equiv 0$). Next we discuss how we estimate the labels from the estimated entry-precisions, followed by our EM-based algorithm to estimate the entry-precisions from the available labeled data.

5.2 Estimating Labels from Entry-Precisions

Under our statistical model, the label $\phi(\tau)$ of a result τ can be estimated by evaluating the probability of its provenance $\texttt{Prv}(\tau)$ given p_w-s for all entry $w \in \texttt{Prv}(\tau)$. Computing the probability of any boolean expression ϕ given the probabilities of its constituent variables is in general *#P-hard* [37], and, the classes of relational algebra queries for which the probability can be efficiently computed have been extensively studied (*e.g.*, see [13]). However, the provenance of the results involve a small number of variables (typically ≤ 10). So we compute $\phi(\tau)$ given p_w-s by an exhaustive enumeration.

5.3 Estimating Entry-Precisions by EM

Here we estimate the values of entry-precisions p_w given a set of results τ along with their provenance $\texttt{Prv}(\tau)$ and labels. We use the *Expectation-Maximization (EM)* algorithm to solve this problem. The EM algorithm [14] is a widely-used technique for the maximum likelihood estimation of parameters of a probabilistic model with hidden variables. This algorithm estimates the parameters iteratively either for a given number of steps or until some convergence criteria are met.

First, we introduce some notation to present the update rules of EM in terms of our problem. We index the entries arbitrarily as $w_1, \cdots, w_n$. Each entry w_i has an entry-precision $p_i = p_{w_i}$. There are N labeled results $\tau_1, \cdots, \tau_N$. We assume that $\tau_1, \cdots, \tau_N$ also denote the labels of the results $\phi(\tau_1), \cdots, \phi(\tau_N)$, so each τ_i is boolean, where $\tau_i = 1$ (resp. 0) if the label is `Good` (resp. `Bad`). If $w_i \in \texttt{Prv}(\tau_j)$, we say that $\tau_j \in \texttt{Succ}(w_i)$. For simplicity of notation, we assume that exactly b dictionary entries are involved in the expression $\phi_j = \texttt{Prv}(\tau_j)$ for each result τ_j; our implementation works for general cases. Hence each ϕ_j takes b inputs $y_{j1}, \cdots, y_{jb}$ and produces τ_j. Each $y_{j\ell}$ is boolean, where $y_{j\ell} = 1$ (resp. 0) if the match of dictionary entry corresponding to $y_{j\ell}$ is correct (resp. incorrect) while producing the label for τ_j. The entry corresponding to $y_{j\ell}$ will be denoted by $\texttt{Prv}_{j\ell} \in \{w_1, \cdots, w_n\}$.

To illustrate the notation, consider the *"firstname-lastname"* rule R_4 in Figure 1: the result is a person if it is a match from the dictionary *first_names.dict*, followed by a match from *last_name.dict*. In this example, $b = 2$ and for every result τ_j, $\tau_j = \phi_j(y_{j1}, y_{j2}) =$

[2]"Island" is a last name (ref. *http://names.whitepages.com/last /island*, *http://www.surnamedb.com/Surname/Island*) in USA.

[3]A match is the tuple in the intermediate relation extracted by the Dictionary operator.

$y_{j1}y_{j2}$. For a `Good` result "April Smith", $\tau_j = 1, y_{j1} = 1$ (for "April"), and $y_{j2} = 1$ (for "Smith"), $\texttt{Prv}_{j1}$ = "April" and $\texttt{Prv}_{j2}$ = "Smith". For a `Bad` result "Victoria Island", $\tau_j = 0, y_{j1} = 1$ (for "Victoria"), and $y_{j2} = 0$ (for "Island").

Parameters and hidden variables for EM. For our problem, the vector of labels of the results $\vec{x} = \langle \tau_1, \cdots, \tau_N \rangle$ are the *observed variables*, the vector of vectors for the correctness of matches of individual entries for these results $\vec{\vec{y}} = \langle y_{j\ell} \rangle_{j \in [1,N], \ell \in [1,b]}$ are the *hidden variables*, and the vector of entry-precisions $\vec{\theta} = \{p_1, \cdots, p_n\}$ is the vector of unknown *parameters*.

Update rules for EM. Let $\vec{\theta}^t$ be the parameter vector at iteration t. The log-likelihood of the observed variables is $q(\vec{x}; \vec{\theta}) = \log P(\vec{x}|\vec{\theta}) = \sum_{j=1}^{N} \log P(\tau_i|\vec{\theta})$. The complete information for the problem includes the observed variables $\vec{x} = \langle \tau_1, \cdots, \tau_N \rangle$ as well as the hidden variables $\vec{\vec{y}} = \langle y_{j\ell} \rangle_{j \in [1,N], \ell \in [1,b]}$. The expected log-likelihood of the complete information given the observed variables $\vec{x}$ and current parameter vector $\vec{\theta}^t$ is $\mathbb{E}[q(\vec{x}, \vec{\vec{y}}; \vec{\theta})|\vec{x}, \vec{\theta}^t] = K$ (say). In the **E-step**, we compute the value of K in terms of p_i, $i \in [1,n]$, by using the current value of the parameters $\vec{\theta}^t$ In the **M-step**, we maximize K w.r.t. parameter vector $\vec{\theta}$, by differentiating K w.r.t. each p_i and equating $\delta K / \delta p_i$ to zero, to get the next guess of the parameters $\vec{\theta}^{t+1}$. We show that this gives a closed-form expression for each p_i which can be efficiently computed in each iteration of EM. The complete derivation is given in Appendix B.

Estimation for simple rules. Here we prove a simplifying observation which will be used in the next section:

OBSERVATION 2. *For simple rules, the estimated entry-precision using EM reduces to its empirical entry-precision. Further, EM converges in a single step.*

For simple rules, $\texttt{Prv}(\tau) = w$, where the result τ is a result of an entry w. (*i.e.* $b = 1$). Fix an arbitrary result τ_j, $j \in [1,N]$, and *the unique* entry w_i such that $\texttt{Prv}_{j,1} = w_i$. Then, at any time step t and for any values of the parameters $\vec{\theta}^t$, $c_{w_i,\tau_j,t} = \mathbb{E}[y_{j1}|\tau_j, \vec{\theta}^t] = \tau_j$. In other words, when the label of a result is given, whether or not the corresponding entry is a correct match for this result can be exactly inferred from the label of the result. Hence, the entry-precision p_i for any entry w_i is $p_i = \frac{C_1}{C_1+C_2} = \frac{\sum c_{w_i,\tau_j,t}}{\sum c_{w_i,\tau_j,t} + \sum(1-c_{w_i,\tau_j,t})}$ (the sums are over all τ_j such that $\tau_j \in \texttt{Succ}(w_i)) = \frac{\sum \tau_j}{\sum \tau_j + \sum(1-\tau_j)}$ which equals the fraction of `Good` results produces by w_i, *i.e.* the empirical entry-precision of w_i. Further, the EM algorithm will converge in a single step since this estimate is independent of the time-step t, and therefore label-estimation can be omitted altogether.

6. REFINEMENT OPTIMIZATION

Next we discuss the problem of maximizing the residual F-score for simple and complex rules (resp. Sections 6.1 and 6.2). This step takes the provenance polynomial and the (possibly fractional) label of all results as input, and outputs a set of entries to be deleted.

6.1 Simple Rules

The following observation simplifies the expressions of precision, recall, and F-score for simple rules.

OBSERVATION 3. *For simple rules, a result $\tau \in \texttt{surv}(S)$ if and only if the entry $w \notin S$ where $w = \texttt{Prv}(\tau)$.*

Let us denote the *frequency* of an entry w (*i.e.*, the number of results τ such that $\texttt{Prv}(\tau) = w$) by f_w. Then $|\texttt{surv}(S)| = \sum_{w \notin S} f_w$, and using Observation 2, $\sum_{\tau \in \texttt{surv}(S)} \phi(\tau) = \sum_{w \notin S} \sum_{\tau: w = \texttt{Prv}(\tau)} \phi(\tau) = \sum_{w \notin S} p_w f_w$. Here p_w is the (estimated) entry-precision of w.[4] Therefore, the expressions for $P_{\bar{S}}, R_{\bar{S}}, F_{\bar{S}}$ given in (3) reduce to:

Algorithm 1 Algorithm for size constraints (given k and Δ)

```
 1: – Let $\theta_{low} = F_A$ and $\theta_{high} = 1$
 2: while $\theta_{high} - \theta_{low} > \Delta$ do
 3:     Let $\theta = (\theta_{high} + \theta_{low})/2$ be the current guess.
 4:     Sort the entries $w$ in descending order of $f_w(\theta - 2p_w)$.
 5:     Let $S$ be the top $\ell \leq k$ entries in the sorted order such that
        $f_w(\theta - 2p_w) \geq 0$ for all $w \in S$.
 6:     if $\sum_{w \in S}[f_w(\theta - 2p_w)] \geq \sum_{w \in A} f_w(\theta - (2 - \theta)p_w)$ then
 7:         $\theta$ is feasible, set $\theta_{low} = F_{\bar{S}}$ and continue.
 8:     else
 9:         $\theta$ is not feasible, set $\theta_{high} = \theta$ and continue.
10:     end if
11: end while
12: Output the set $S$ used to define the most recent $\theta_{low}$.
```

$$P_{\bar{S}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \notin S} f_w}, \; R_{\bar{S}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w}, \; F_{\bar{S}} = 2\frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w}$$

In Section 6.1.1, we give an optimal and efficient algorithm to maximize the F-score under size constraints. For recall constraints, in Section 6.1.2 we show that the exact optimization is NP-hard. However, we give a simple and efficient algorithm that is provably nearly optimal and works well in our tests.

6.1.1 Size Constraints

Our goal is to maximize $F_{\bar{S}} = 2\frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w}$, where $|S| \leq k$. The main idea of our algorithm is that finding out whether there exists a dictionary with F-score of at least θ is a significantly simpler problem which overcomes the non-linearity of the objective function [16]. Accordingly, our algorithm *guesses* a value θ and then checks if θ is a feasible F-score for some S. The value of θ that maximizes $F_{\bar{S}}$ is then found via a binary search.

Checking if θ is a feasible F-score. We need to check whether there is a set S of entries such that $F_{\bar{S}} = 2\frac{\sum_{w \in A} p_w f_w - \sum_{w \in S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \in A} f_w - \sum_{w \in S} f_w} \geq \theta$, *i.e.*, we need to find out whether there exists S such that

$$\sum_{w \in S} f_w(\theta - 2p_w) \geq \sum_{w \in A} f_w(\theta - (2 - \theta)p_w)$$

The right hand side of the inequality is independent of S, so it suffices to select the top-most (at most) k entries with non-negative value of $f_w(\theta - 2p_w)$ and check if the sum is at least $\sum_{w \in A} f_w(\theta - (2 - \theta)p_w)$. Clearly we want a subset S such that $F_{\bar{S}} \geq F_A$. Hence the guess θ is varied between F_A and 1. Algorithm 1 presents this idea in terms of an accuracy parameter Δ; the value of Δ for the optimal F-score will be discussed later.

Running time. There is an $O(n)$-time algorithm for the feasibility step 6: (i) Use the standard linear time selection algorithm to find the k-th highest entry, say u, according to $f_w(\theta - 2p_w)$, (ii) do a linear scan to choose the entries w such that $f_w(\theta - 2p_w) > f_u(\theta - 2p_u)$, and choose entries such that $f_w(\theta - 2p_w) = f_u(\theta - 2p_u)$ to get k entries in total, (iii) discard the selected entries with negative values of $f_w(\theta - 2p_w)$ and output the remaining $\leq k$ entries. However, we can have simpler implementations of the verification step - using a min-heap gives $O(n + k \log n)$ time, and a simple sorting gives $O(n \log n)$ time. Since values of the guesses are between 0 and 1, and the algorithm stops when the upper and lower bounds are

[4]For unlabeled results τ-s such that $\texttt{Prv}(\tau) = w$, estimated label $\phi(\tau) = p_w$. For the labeled results, the fraction having label 1 (`Good`) is p_w.

less than Δ away, at most $\log(1/\Delta)$ steps will be required. Hence there is an implementation that runs in time $O(n\log(1/\Delta))$.

Value of Δ for optimal F-score. Let B be the number of bits to represent each p_w (fraction), f_w (integer) in the input. Consider any $F_{\overline{S}} = 2\frac{\sum_{w\notin S} p_w f_w}{\sum_{w\in A} p_w f_w + \sum_{w\notin S} f_w}$. B-bit numbers between 0 and 1 can represent values $t.2^{-B}$, for $0 \le t \le 2^B - 1$. Multiply both numerator and denominator of $F_{\overline{S}}$ by 2^B to get integer values in the numerator and denominator. Each of $p_w f_w$ and f_w in the denominator is $\le 2^{2B}$ after this multiplication, and there are at most $2n$ of them. The denominator of the fraction representing difference between two unequal F-score values is $\le n2^{2B+1}$, whereas the numerator of the fraction is ≥ 1. Hence the difference is $\ge \frac{1}{(n2^{2B+1})^2}$, and setting $\Delta = \frac{1}{(n2^{2B+1})^2}$ suffices. This leads to the following theorem:

THEOREM 1. *For simple rules, there is a poly-time optimal algorithm to maximize the F-score under size constraint. The algorithm runs in time $O(n \cdot (\log n + B))$, where B is the number of bits used to represent each p_w and f_w given to the algorithm.*

A natural question that may arise is whether selecting entries greedily (select the next entry that gives the maximum improvement in F-score and repeat for k steps) also gives an optimal solution. We give an example in Appendix A to show that greedy is not optimal.

6.1.2 Recall Constraints

First, we sketch the proof of NP-hardness. Then we give the near-optimal algorithm.

NP-hardness for exact optimization. We give a reduction from the *subset-sum problem* which is known to be NP-hard [18]. In the subset-sum problem the input is a set of positive integers $I = \{x_1, \cdots, x_n\}$[5], and an integer C, and the goal is to decide if there is a subset $S \subseteq I$ such that $\sum_{x_i \in S} x_i = C$.

Our reduction creates an instance of the refinement problem in which every number in the subset-sum instance corresponds to an entry with fixed and low entry-precision and frequency proportional to the number. In addition, we create a single word with high entry-precision. This word ensures that the highest F-score is achieved when the total frequency of the removed low precision words is the highest. Therefore, the maximum F-score can be achieved only when the recall budget is used exactly. The complete reduction is given in Appendix A.

Nearly Optimal Algorithm. We now describe a simple and efficient algorithm that gives a nearly optimal solution when used on a large corpus where frequencies of individual entries are small. Our algorithm sorts the entries in increasing order of entry-precisions p_w, and selects entries according to this order until the recall budget is exhausted or there is no improvement of F-score by selecting the next entry (the pseudocode of the algorithm is straightforward and is omitted). Clearly the algorithm runs in time $O(n \log n)$. Formally, we prove the following theorem that gives a lower bound on the F-score of the solution produced by our algorithm.

THEOREM 2. *Let $w_1, \cdots, w_n$ be the entries sorted by the corresponding entry-precisions $p_1 \le \cdots \le p_n$ (here $p_i = p_{w_i}$). Let S^* be the set of entries whose removal gives the optimal F-score and $R_{\bar{S}^*} \ge \rho$. Let $r^* = \sum_{i\in\bar{S}^*} p_i f_i$ and let ℓ be the largest index for which $\sum_{i>\ell} p_i f_i \ge r^*$. Then the set S returned by our algorithm satisfies*

$$F_{\bar{S}} \ge \frac{2\sum_{i\in\bar{S}^*} p_i f_i}{\sum_{i\in\bar{S}^*} f_i + \sum_i p_i f_i + f_{max}/p_{\ell+1}}.$$

[5]The subset-sum problem is NP-hard even for positive integers.

PROOF. Let $p_i = p_{w_i}$ and $f_i = f_{w_i}$, where $p_1 \le \cdots \le p_n$. Let $f_{max} = \max\{f_1, \ldots, f_n\}$, $S^i = \{w_j : 1 \le j \le i\}$ and $\overline{S} = A \setminus S$. Let $r^\ell = \sum_{i\in\overline{S^\ell}} p_i f_i = \sum_{i>\ell} p_i f_i$. By definition, $r^* + f_{max} \ge r^\ell \ge r^*$. Note that recall $R_{\overline{S^\ell}} = r^\ell / \sum_i p_i f_i \ge r^* / \sum_i p_i f_i = R_{\overline{S^*}} \ge \rho$. Due to the monotonicity check, the algorithm returns a solution with F-score $\ge F_{\overline{S^\ell}}$. Hence we give a lower bound on $P_{\overline{S^\ell}}$.

Observe that: (1) $\sum_{i\in\overline{S^\ell}\setminus\overline{S^*}} p_i f_i - \sum_{i\in\overline{S^*}\setminus\overline{S^\ell}} p_i f_i = \sum_{i\in\overline{S^\ell}} p_i f_i - \sum_{i\in\overline{S^*}} p_i f_i \le f_{max}$, and, (2) $\sum_{i\in\overline{S^\ell}\setminus\overline{S^*}} f_i \le \frac{\sum_{i\in\overline{S^\ell}\setminus\overline{S^*}} p_i f_i}{p_{\ell+1}} \le \frac{(\sum_{i\in\overline{S^*}\setminus\overline{S^\ell}} p_i f_i + f_{max})}{p_{\ell+1}} \le \sum_{i\in\overline{S^*}\setminus\overline{S^\ell}} f_i + \frac{f_{max}}{p_{\ell+1}}$. From (1), (2): $\sum_{i\in\overline{S^\ell}} f_i \le \sum_{i\in\overline{S^*}} f_i + \frac{f_{max}}{p_{\ell+1}}$. Hence $P_{\overline{S^\ell}} = \frac{r^\ell}{\sum_{i\in\overline{S^\ell}} f_i} \ge \frac{r^*}{\sum_{i\in\overline{S^*}} f_i + f_{max}/p_{\ell+1}}$, and, $F_{\overline{S^\ell}} = \frac{2}{1/R_{\overline{S^\ell}} + 1/P_{\overline{S^\ell}}} \ge \frac{2}{1/R_{\bar{S}^*} + 1/P_{\overline{S^\ell}}} \ge \frac{2\sum_{i\in\overline{S^*}} p_i f_i}{\sum_{i\in\overline{S^*}} f_i + \sum_i p_i f_i + \frac{f_{max}}{p_{\ell+1}}}$ □

Note that the lower bound guaranteed by the algorithm differs from the optimal F-score $F_{\bar{S}^*}$ only by the addition of the error term $\frac{f_{max}}{p_{\ell+1}}$ to the denominator. Individual frequencies are likely to be small when the given corpus and the dictionary are large. At the same time ℓ and hence $p_{\ell+1}$ are determined solely by the recall budget. Therefore the error term $\frac{f_{max}}{p_{\ell+1}}$ is likely to be much smaller than the denominator for a large dictionary. Our experiments support this argument. Another surprising property of this algorithm is that while it is not necessarily optimal in general, without the recall budget (*i.e.* $\rho = 0$), it finds the globally optimal solution; we defer the proof to the full version of the paper due to lack of space. Naturally, the slightly more involved Algorithm 1 with $k = n$ also gives the optimal solution for $\rho = 0$.

6.2 Complex Rules

The optimization problems become substantially harder for complex rules. In Section 6.2.1, we show that even for a complex rule as simple as "firstname-lastname" (rule R_4 in Figure 1) the optimization problem for size constraint is NP-hard. Further, the problem is NP-hard even when the independence and rule correctness assumptions we make in our statistical model hold. Note that this problem was shown to be poly-time solvable for simple rules. The case of recall constraint has already been shown to be NP-hard even for simple rules. Then in Section 6.2.2 we discuss efficient algorithms that we evaluate experimentally.

6.2.1 NP-hardness for Size Constraint

We give a reduction from the *k'-densest subgraph problem in bipartite graphs* [11]. Here the inputs are a bipartite graph $H(U,V,E)$ with n' vertices and m' edges, and an integer $k' < n'$. The goal is to select a subset of vertices $W \subseteq U \cup V$, $|W| = k'$, such that the subgraph induced on W has the maximum number of edges. We will denote the set of edges in the *induced subgraph* on W (every edge in the subgraph has both its endpoints in W) by $E(W)$.

For simplicity, first we prove a weaker claim: removing a subset S with *exactly* k (as opposed to *at most* k) vertices is NP-hard. Intuitively, the vertices correspond to entries and the edges correspond to results. We show that if the induced subgraph on a subset of vertices of size k' has a large number of edges, then removing entries in the *complement* of this subset gives high F-score.

Given an instance of the k'-densest subgraph problem, we create an instance of the dictionary refinement problem as follows. The vertices in U and V respectively correspond to the entries in the firstname and lastname dictionaries. Every edge $(u,v) \in E$ corresponds to a unique provenance expression $\phi_{u,v} = uv$, where the entries u and v are chosen from these two dictionaries respectively. For each $(u,v) \in E$, there is one result with label 1 (`Good`), and one with label 0 (Bad). The parameter k in the dictionary refinement problem is $k = n' - k'$. It is not hard to see that there is a subset $W \subseteq U \cup V$, such that $|W| = k'$ and $E(W) \geq q$ if and only if there is a subset S for the dictionary refinement problem such that $|S| = k$ and $F_{\bar{S}} \geq \frac{2}{\frac{m'}{q}+2}$ (proof is deferred to the full version).

The residual precision in the above reduction is a constant for all choices of S, and therefore, the F-score is a monotone function of the residual recall. Hence the above reduction does not work for the relaxed constraint $|S| \leq k$ (the residual recall is always maximized at $S = \emptyset$, i.e. when $k = 0$, independent of the k'-densest subgraph solution). Below we sketch the reduction for $|S| \leq k$, the complete reduction is deferred to the full version due to space constraint.

Outline of reduction for $|S| \leq k$. To strengthen the above reduction for $|S| \leq k$, we retain the graph with a good and a bad result for every edge as before. In addition, we add $s = mn$ Good results that are unrelated to anything; they will make the differences in the recall between solutions tiny (while preserving monotonicity in the size of $E(W)$). For every original entry $u \in U \cup V$, we add s Bad results (u, u^i) $1 \leq i \leq s$ connected to it. The entries $\bigcup_{u \in U \cup V} \{u_i : 1 \leq i \leq s\}$ are called auxiliary-bad entries. This way the precision will be roughly equal to $\frac{1}{n-k}$ (since these results dominate the total count) and hence solutions with smaller number of entries removed will have noticeably lower precision. Removing any of the auxiliary bad entries will have a tiny effect, so any optimal solution will always remove the entries corresponding to graph vertices $U \cup V$. Finally, it is easy to observe that labels for results that we create in this reduction can be obtained by assuming independent entry-precisions for all the dictionary entries created in the reduction. This proves the following theorem:

THEOREM 3. *For complex rules, maximizing the F-score under size constraint is NP-hard even for the firstname-lastname rule (*i.e., *the rule R_4 in Figure 1), and even when the entry-precisions satisfy independence.*

6.2.2 Refinement Algorithms

Since the dictionary refinement problem is NP-hard for complex rules under both size and recall constraints, we propose and evaluate efficient algorithms that work well in practice. These algorithms take the (actual or estimated) label and the provenance of each result as input, and produce a subset of entries to remove across all dictionaries.

Some natural and simple approaches that we examine for *size constraint* are (1) choosing (at most) top-k entries in decreasing order of *the fraction of* Bad *results* or *the number of* Bad *results* they produce, ignoring the actual provenance polynomials, and (2) greedily choosing (at most) k entries that give the maximum improvement in the F-score. Note that the implementation of these algorithms involves repeated computation of the surv(S) sets after each entry is included to S by evaluating the provenance polynomials of all results.

In addition, we improve the solution obtained by each of these algorithms by performing a *hill climbing* algorithm from the solution returned by the algorithm to find a local maximum. Here two solutions (set of entries to be deleted) S_1, S_2 are considered to be neighbors if S_2 can be obtained from S_1 by replacing an entry $w_1 \in S_1$ with a new entry $w_2 \notin S_1$. We iteratively choose the neighbor that gives the maximum improvement ΔF in the F-score. The algorithm stops if there is no more change in the set S of entries being removed, or a specified maximum number of iterations allowed is reached. The pseudocode for the generic hill-climbing algorithm is given in Algorithm 2.

For *recall constraint*, we use the same approach. However, while

Algorithm 2 Generic hill-climbing Algorithm for complex rules, size constraint (given maximum number of iterations *MAX*)

```
 1: – Find an initial solution (an array) $S$.
 2: while $S$ changes and the no. of iterations is $\leq MAX$ do
 3:   for each position $i$ in $S$ do
 4:     Let $w = S[i]$, and let $f$ be the current F-score
 5:     for each entry $w' \notin S$ do
 6:       Compute the new fscore $f_{w,w'}$ using $S \setminus \{w\} \cup \{w'\}$.
 7:     end for
 8:     Let $w_m = argmax_{w'} f_{w,w'}$.
 9:     If $f_{w,w*} > f$, $S[i] = w_m$. Otherwise $S$ is unchanged.
10:   end for
11: end while
12: Output the set $S$.
```

doing the local search using hill climbing, we maximize $\Delta F/\Delta R$ at each step (ΔR denotes the change in recall by deleting one additional entry), since we are trying to maximize F while still satisfying $R \geq \rho$ for a given recall budget ρ. These algorithms are empirically evaluated in Section 7.2.2.

7. EXPERIMENTS

Now we present our experimental results. We discuss the datasets and extractors used in our experiments in Section 7.1. In Section 7.2, we evaluate the refinement algorithms on fully labeled datasets and compare their performance and efficiency; this supplements the theoretical results derived earlier. We evaluate our label estimation approach on partially labeled datasets in Section 7.3.

Experimental Settings. We used SystemT v1.0.0 [25, 32], enhanced with a framework to support provenance of the results [26]. The rules are specified in AQL (Annotation Query Language), which is the rule language of SystemT. All experiments were implemented in Java with JDK 6.0, and run on a 64-bit Windows 7 machine with Intel(R) Core^{TM} i7-2600 processor (4 GB RAM, 3.40 GHz).

7.1 Evaluation Datasets and Extractors

We used two labeled datasets in our experiments: CoNLL2003 (*CONLL* in short, with 945 documents) and ACE2005 (*ACE* in short, with 342 documents)[6]; both these datasets are frequently used in official Named Entity Recognition competitions [3, 36].

Each dataset contains *occurrences* identified by "(document id, begin offset, end offset)", and tagged with the corresponding entity name (Person, Location, Organization, etc.). Given an extractor, a result is Good (true positive) if it belongs to the set of occurrences for the corresponding entity, otherwise it is Bad (false positive).

We primarily considered two extractors to evaluate the cases of *simple rules* (provenance polynomials comprises a single variable) and *complex rules* (arbitrary provenance polynomials): (1) For simple rules a Person extractor is used, which comprises a single AQL rule (see Section 3) and a single dictionary "name.dict" containing contains both first and last names (12992 entries in total); (2) For complex rules, we use an Organization extractor which is part of a generic Named Entity Extractor for Person, Location and Organization ([9]). This extractor uses a variety of (over 400) rules and more than 120 dictionaries curated from public data sources such as the US Census Bureau (Census, 2007) and Wikipedia. There are dic-

[6]We do not consider larger labeled datasets because these are usually hard to come by in practice. However, we do show that the quality of the results output by the algorithm improves as more labeled data is available

tionaries for common first and last names from different countries, salutation and suffixes for person names, different locations (cities, states, countries, etc.), industries and organizations (medical, government, education, etc.), months and days and so on. In particular to extract Organization entities, it generates candidate mentions using dictionaries of full organization names, rules that combine syntactic features (capitalized tokens) with dictionaries of clues including industry suffixes (*e.g.*,"Technologies") or company suffixes ("Co.", "Ltd."), as well as complex filtering rules to disambiguate across different types of named entities.

Handling set-difference operation in the extractor. In practical extractors, including the complex extractor used in our experiments, *set-difference* operators (like SQL MINUS) are typically unavoidable. For instance, the extractor may output a person name if it appears in a dictionary of common names, and it does not appear in the dictionary of common organization names. In general, the extractor may output a tuple t if it appears in an intermediate view R_1 (as another tuple t_1) and does not appear in another view R_2. However, note that if t is output, there is no tuple t_2 in R_2, so it is not possible to write $\texttt{Prv}(t)$ as $\texttt{Prv}(t_1) \cdot \neg\texttt{Prv}(t_2)$. In this case we store $\texttt{Prv}(t) = \texttt{Prv}(t_1)$ [26]. This approach is always sound (removing t_1 does remove t), and it is not hard to see that it is complete as well as long as there is no nested set-difference operations (in that case both $\texttt{Prv}(t_1), \texttt{Prv}(t_2)$ are monotone and the only way to remove a result t by deleting entries is by removing t_1). For nested difference operations, t_2 may appear (and as a result t disappears) if some entries are removed. However, to handle this we need to repeatedly run the complex extractor in our algorithms which is not a scalable approach. Therefore, we use $\texttt{Prv}(t) = \texttt{Prv}(t_1)$ (which are monotone boolean expressions) and our algorithms operate with these provenance polynomials as inputs.

Here we present results for the simple person extractor on the CONLL dataset (simple rule case), and the organization extractor from the sophisticated annotator mentioned above on ACE (complex rule case); results with other combinations of extractors and datasets are similar and are omitted for space limitations.

7.2 Dictionary Refinement with Full Labeling

Here we evaluate the algorithms proposed in Section 6 for simple and complex rules, both in terms of the resulting F-score after refinement and running time of the algorithms.

7.2.1 Refinement Algorithms for Simple Rules

For both size and recall constraints, we compare our algorithms (Sections 6.1.1 and 6.1.2) with the greedy algorithms proposed for complex rules (Section 6.2.2) and two other natural approaches. The algorithms compared for **size constraint** are (see Figure 2 (i)): (1) *K-Optimal*: Algorithm 1 with $\Delta = 0.001$, (2) *K-Greedy*: the greedy algorithm for complex rules – the entry giving the maximum increment in F-score (*i.e.* ΔF) is selected next, (3) *K-BadFraction*: chooses the top-k entries in the decreasing order of the *fraction* of `Bad` results, (4) *K-BadCount*: chooses the top-k entries in the decreasing order of the *number* of `Bad` results.

The algorithms compared for **recall constraint** are (see Figure 2 (ii)): (1) *RecBudget-Near-Opt*: the near-optimal algorithm that selects entries in the decreasing order of the fraction of `Bad` results (see Section 6.1.2), (2) *RecBudget-Greedy-FR*: the greedy algorithm for complex rules that maximizes the incremental improvement in F-score relative to the decrease in recall (*i.e.* $\Delta F/\Delta R$) at each step, (3) *RecBudget-Greedy*: algorithm similar to K-Greedy that maximizes ΔF at each step, (4) *RecBudget-BadCount*: chooses the entries in the decreasing order of the *number* of `Bad` results. Each algorithm is run until the given recall budget ρ is exhausted.

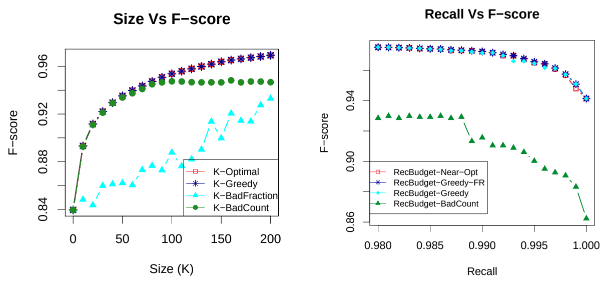


Figure 2: Refinement algorithms for simple rules: F-score.

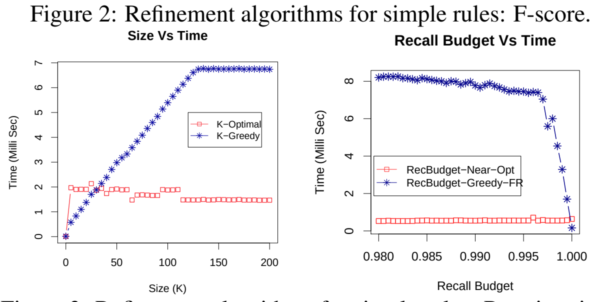


Figure 3: Refinement algorithms for simple rules: Running time.

Observations. First and foremost, Figure 2 shows that dictionary refinement improves the F-score for both size and recall constraints. For both constraints, the performance of our (near-) optimal algorithms and the greedy algorithms for complex rules are comparable on our datasets. They also perform as well or better than the other two algorithms. However, as we have shown in Example 1, the greedy algorithm may not give optimal solution for size constraint, and as discussed below, our algorithms run faster than the greedy algorithms.

Figure 3 gives the running time of the best two algorithms for both size and recall constraints, averaged over 100 runs. This figure shows that the algorithms are efficient and therefore can be embedded in interactive tools. After a certain value of size or recall budget, removing more dictionary entries does not improve the F-score; therefore the greedy algorithm stops and the curve becomes stable. For size (recall) constraint the greedy algorithms requires more time than the (near-) optimal algorithms at higher values of k (lower values of ρ, respectively). For size constraint, optimal and greedy algorithms run in $O(n \log n)$ and $O(kn)$ time respectively, n being the number of dictionary entries; the running times are similar for recall constraint. For $\Delta = 0.001$, K-Optimal converges in only 5 to 6 iterations. The advantage in running time of our algorithms is expected to be even more substantial for larger datasets.

7.2.2 Refinement Algorithms for Complex Rules

For complex rules, we compare the local search algorithms presented in Section 6.2 with other natural approaches. Similar to the case of simple rules, for **size constraint**, we test (1) *K-Greedy*: selects the next entry giving maximum ΔF, and (2) *K-BadCount* and (3) *K-BadFraction*: selects the next entry that has produced the maximum *number* and *fraction* of `Bad` results respectively. These algorithms are compared with (see Figure 4 (i)): (4) *K-LocalSearch*: our local search (hill-climbing) algorithm which finds the initial solution by K-BadFraction, and then finds the local maximum using Algorithm 2. Although we have also tried K-Greedy and K-BadFraction to give the initial solution, K-BadFraction outperforms these algorithms both as a standalone algorithm, and also when it is used as the initial solution for K-LocalSearch.

The corresponding algorithms for **recall constraint** are
(1) *RecBudget-Greedy-FR* (selects the next entry that maximizes $\Delta F/\Delta R$ at each step), (2) *RecBudget-BadCount*, (3) *RecBudget-*

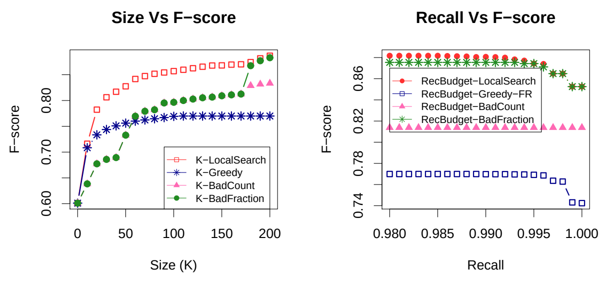


Figure 4: Refinement algorithms for complex rules: F-score.

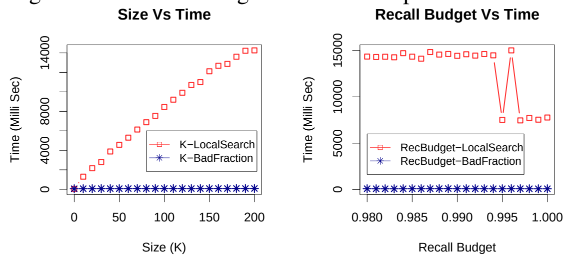


Figure 5: Refinement algorithms for complex rules: Running time.

BadFraction, and (4) *RecBudget-LocalSearch*: our proposed local search algorithm that finds the initial solution by RecBudget-BadFraction and then finds the local maximum by maximizing $\Delta F/\Delta R$ at each step of the search (see Figure 4 (ii)).

Observations. Figure 4 shows that the proposed local search algorithms perform better than the other algorithms for both size and recall constraints. Figure 5 compares the running time of the best two algorithms for both size and recall constraints averaged over 10 runs (local search and sorting by the fraction of bad results). Naturally, K-BadFraction and RecBudget-BadFraction have much better running time than the local search algorithms as they involve a single round of sorting of the entries. The local search algorithms K-LocalSearch and RecBudget-LocalSearch first execute K-BadFraction and RecBudget-BadFraction respectively, and then continue searching for better solutions. Based on the application (*e.g.* for finding the obvious candidate entries to be removed), K-BadFraction may be used instead of K-LocalSearch for faster execution. The runtime for RecBudget-LocalSearch is high even at $\rho = 1.0$ (unlike K-LocalSearch at $k = 0$) since at $\rho = 1.0$ many entries may get deleted that produce *no* `Good` results. Further, running time of RecBudget-LocalSearch is not monotone with ρ unlike K-LocalSearch. For lower values of ρ the algorithm can choose entries that consume most of the recall budget and stop earlier which cannot be selected when ρ is high.

7.3 Refinement with Incomplete Labeling

Now we evaluate our label estimation approaches for *incomplete labeling* of the results. We estimate the label of each unlabeled result using the algorithms in Section 5; the actual labels are retained for labeled results. Then we run the refinement algorithms on the entire resultset. We call this the ESTIMATED approach, and compare it with the *NOT-ESTIMATED* approach, where the algorithms are run *only* on the labeled resultset.

In our tests we choose a random subset of the results in our dataset and define it to be the labeled resultset (that is, the rest of the labels are hidden from our label estimation and refinement algorithms). The set of deleted entries returned by each algorithm for both of these approaches are evaluated against the actual `Good`/`Bad` labels for the entire dataset to obtain the F-score after refinement. This is repeated 10 times, using different random subsets of labeled results and the mean F-scores are plotted in Figure 6 for different

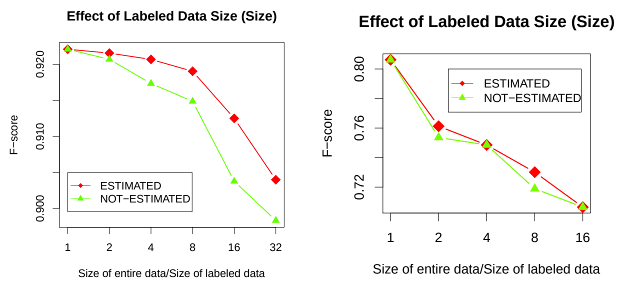


Figure 6: Effect of the labeled data size (simple and complex rules).

sizes of labeled result set and size constraint (measured as the fraction of all the results in the dataset). We omit the plots for recall constraints for both simple and complex rules due to lack of space, and discuss them in text.

7.3.1 *Incomplete Labeling for Simple Rules*

As mentioned in Section 5.3 (Observation 2), the label of a result produced by an entry is simply the fraction of `Good` results produced by that entry (entries with no labeled results are ignored), and therefore, the EM algorithm is not run for this case. Figure 6 (i) shows that, for size constraint, ESTIMATED approach gives better F-score over the NOT-ESTIMATED approach, and the benefit is higher when fewer results are labeled (indeed, both approaches are equivalent when fraction of labeled results equals 1). However, for recall constraint, these two approaches give essentially the same result. This is as expected, since our near-optimal algorithm chooses the entries according to the fraction of `Bad` results, which is the same for both labeled and the entire resultset.

7.3.2 *Incomplete Labeling for Complex Rules*

We first estimate the entry-precisions by the EM algorithm that uses the provenance and labels of the labeled results. (ref. Section 5.3). The EM algorithm is fast enough for practical purposes (it converges in about 15-20 steps and takes about 2 seconds to run). Then the missing labels are estimated by evaluating the provenance of the unlabeled results using the estimated entry-precisions (ref. Section 5.2). Figure 6 (ii) shows that the ESTIMATED approach is better or at least as good as the other approach. However, the improvement is not as significant as in the case of simple rules (also in the case of recall constraint), possibly at least in part because of our simplifying independence assumption not holding in text data. Finding a more practical model toward estimating missing labels will be an interesting future work. Nevertheless, our label estimation technique may be useful in other applications, *e.g.* in view maintenance in relational setting, if the independence assumption in the source data holds.

7.4 Qualitative Evaluation

Figure 7 shows the first 10 entries returned by our (near-) optimal algorithms for simple rules when 10% of the results are labeled. We used the Person extractor described earlier, and indeed, these entries are incorrect or ambiguous as person names. We also see that the entries removed for recall constraint are the ones with all `Bad` results (they improve precision and F-score without reducing recall). However, for size constraint, entries with non-zero `Good` counts may also be chosen as the number of `Good` and `Bad` results also play an important role in this case.

For complex rules, we list the entries returned by K-LocalSearch for $k = 10$. The entries are *administration, coalition, force, government, guess, U.N., forces, lot*, some collected from more than one dictionaries (and therefore are treated as different entries). For

K-Optimal		
	$(good, bad)$ Count in Result Set	
Entry	Labeled	Labeled + Unlabeled
china	(0, 12)	(0, 100)
kong	(0, 11)	(0, 70)
june	(0, 9)	(0, 97)
hong	(1, 10)	(2, 71)
september	(0, 8)	(0, 101)
king	(0, 5)	(6, 20)
louis	(1, 6)	(4, 33)
long	(0, 4)	(0, 66)
cleveland	(0, 4)	(0, 30)
april	(0, 4)	(0, 30)

RecBudget-Near-Opt		
	$(good, bad)$ Count in Result Set	
Entry	Labeled	Labeled + Unlabeled
china	(0, 12)	(0, 100)
kong	(0, 11)	(0, 70)
june	(0, 9)	(0, 97)
september	(0, 8)	(0, 101)
king	(0, 5)	(6, 20)
long	(0, 4)	(0, 66)
cleveland	(0, 4)	(0, 30)
april	(0, 4)	(0, 42)
january	(0, 4)	(0, 28)
re	(0, 4)	(0, 54)

Figure 7: Top 10 entries output for size and recall constraints.

instance, the entry *U.N.* produces some `Good` results, *e.g. U.N. Security Council* and *U.N. Foundation*, and many `Bad` results when extracted by itself, because in these instances it overlaps with a person's job description, *e.g. Jeremy Greenstock, British Ambassador to U.N.*, and is labeled as part of Person mentions in [3]. Looking at these entries, the human supervisor may investigate the `Good` and `Bad` results they produce, and delete these entries or further refine the rules to remove the false positives in the output of the system. For instance, our system identifies *U.N.*, an example entry that could be handled by adding a special rule for filtering Organization mentions identified as part of a Person's position.

8. CONCLUSION AND FUTURE WORK

In this paper we studied one important aspect of building a high quality information extraction system, that of refining dictionaries used in the extractor. We provided rigorous theoretical analysis and experimental evaluation of the optimization problems that such refinement entails. We also proposed and evaluated a statistical approach for coping with small labeled data. First and foremost, our experimental results show that dictionary refinement can significantly increase the quality (F-score) of extraction using even a small amount of labeled data. Further, our experimental results validate the effectiveness of our algorithms, as well as show which of the several other natural algorithms perform better on real data sets.

There are several interesting future directions. Although the independence assumption in our statistical modeling is a good starting point, correlation among the results and their labels is highly likely to exist for text data. A more detailed modeling allowing correlation will be an interesting future work. Another interesting directions include adaptively labeling a corpus for dictionary refinement given a budget on the number of labels, and fully incorporating non-monotonic operators into our framework, including set difference and operators for removing span overlap.

9. REFERENCES

[1] In *www.census.gov*.

[2] In *www.geonames.org*.

[3] *Automatic Content Extraction 2005 Evaluation Dataset*. 2005.

[4] E. Agichtein and L. Gravano. *Snowball*: Extracting Relations from Large Plain-Text Collections. In *ACM DL*, pages 85–94, 2000.

[5] N. Ashish, S. Mehrotra, and P. Pirzadeh. XAR: An Integrated Framework for Information Extraction. In *WRI Wold Congress on Computer Science and Information Engineering*, 2009.

[6] P. Buneman, S. Khanna, and W.-C. Tan. On propagation of deletions and annotations through views. In *PODS*, pages 150–158, 2002.

[7] X. Chai, B.-Q. Vuong, A. Doan, and J. F. Naughton. Efficiently incorporating user feedback into information extraction and integration programs. In *SIGMOD*, 2009.

[8] J. Cheney, L. Chiticariu, and W. Tan. Provenance in databases: Why, how, and where. *Foundations and Trends in Databases*, 1(4):379–474, 2009.

[9] L. Chiticariu, R. Krishnamurthy, Y. Li, F. Reiss, and S. Vaithyanathan. Domain adaptation of rule-based annotators for named-entity recognition tasks. In *EMNLP*, pages 1002–1012, 2010.

[10] W. W. Cohen and S. Sarawagi. Exploiting dictionaries in named entity extraction: combining semi-markov extraction processes and data integration methods. In *KDD*, pages 89–98, 2004.

[11] D. G. Corneil and Y. Perl. Clustering and domination in perfect graphs. *Discrete Applied Mathematics*, 9(1):27 – 39, 1984.

[12] H. Cunningham. JAPE: a Java Annotation Patterns Engine. Research Memorandum CS – 99 – 06, University of Sheffield, May 1999.

[13] N. N. Dalvi, K. Schnaitter, and D. Suciu. Computing query probability with incidence algebras. In *PODS*, pages 203–214, 2010.

[14] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *JOURNAL OF THE ROYAL STATISTICAL SOCIETY, SERIES B*, 39(1):1–38, 1977.

[15] H. Elmeleegy, J. Madhavan, and A. Halevy. Harvesting relational tables from lists on the web. *PVLDB*, pages 1078–1089, 2009.

[16] D. Eppstein and D. S. Hirschberg. Choosing subsets with maximum weighted average. *J. Algorithms*, 24(1):177–193, 1997.

[17] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Methods for domain-independent information extraction from the web: an experimental comparison. In *AAAI*, 2004.

[18] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. 1979.

[19] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *PODS*, pages 31–40, 2007.

[20] J. Hoffart, M. A. Yosef, I. Bordino, H. Fürstenau, M. Pinkal, M. Spaniol, B. Taneva, S. Thater, and G. Weikum. Robust disambiguation of named entities in text. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, EMNLP '11, 2011.

[21] D. Jurafsky and J. Martin. *Speech and language processing: an introduction to natural language processing, computational linguistics, and speech recognition*. Pearson Prentice Hall, 2009.

[22] J. Kazama and K. Torisawa. Inducing gazetteers for named entity recognition by large-scale clustering of dependency relations. In *ACL*, pages 407–415, 2008.

[23] B. Kimelfeld, J. Vondrák, and R. Williams. Maximizing conjunctive views in deletion propagation. In *PODS*, pages 187–198, 2011.

[24] Z. Kozareva. Bootstrapping named entity recognition with automatically generated gazetteer lists. In *EACL: Student Research Workshop*, 2006.

[25] R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, S. Vaithyanathan, and H. Zhu. SystemT: a system for declarative information extraction. *SIGMOD Record*, 37(4):7–13, 2008.

[26] B. Liu, L. Chiticariu, V. Chu, H. V. Jagadish, and F. R. Reiss. Automatic Rule Refinement for Information Extraction. *PVLDB*, 2010.

[27] M. P. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. *COMPUTATIONAL LINGUISTICS*, 19(2):313–330, 1993.

[28] D. Maynard, K. Bontcheva, and H. Cunningham. Towards a semantic extraction of named entities. In *Recent Advances in Natural Language Processing*, 2003.

[29] A. Meliou, W. Gatterbauer, S. Nath, and D. Suciu. Tracing data errors with view-conditioned causality. In *SIGMOD*, 2011.

[30] A. Mikheev, M. Moens, and C. Grover. Named entity recognition without gazetteers. In *EACL*, pages 1–8, 1999.

[31] D. Nadeau, P. D. Turney, and S. Matwin. Unsupervised named-entity recognition: Generating gazetteers and resolving ambiguity. In *Canadian Conference on AI*, pages 266–277, 2006.

[32] F. Reiss, S. Raghavan, R. Krishnamurthy, H. Zhu, and S. Vaithyanathan. An algebraic approach to rule-based information extraction. In *ICDE*, pages 933–942, 2008.

[33] E. Riloff. Automatically constructing a dictionary for information extraction tasks. In *KDD*, 1993.

[34] W. Shen, P. DeRose, R. McCann, A. Doan, and R. Ramakrishnan. Toward best-effort information extraction. In *SIGMOD*, 2008.

[35] W. Shen, A. Doan, J. F. Naughton, and R. Ramakrishnan. Declarative information extraction using datalog with embedded extraction predicates. In *VLDB*, pages 1033–1044, 2007.

[36] E. F. Tjong Kim Sang and F. De Meulder. Introduction to the CoNLL-2003 shared task: language-independent named entity recognition. In *HLT-NAACL*, 2003.

[37] L. G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[38] C. J. van Rijsbergen. *Information Retrieval*. Butterworth, 1979.

[39] A. Yates, M. Banko, M.Broadhead, M. J. Cafarella, O. Etzioni, and S. Soderland. TextRunner: Open Information Extraction on the Web. In *HLT-NAACL (Demonstration)*, pages 25–26, 2007.

APPENDIX

A. PROOFS FROM SECTION 6.1

EXAMPLE 1. ***Greedy is not optimal.*** *Let* $n = 4$ *and* $k = 2$, $A = \{w_1, w_2, w_3, w_4\}$. *The pair of entry-precision and (relative) frequency* (p_{w_i}, f_{w_i}), $1 \leq i \leq 4$ *of these entries are* $(0.0284, 0.2374)$, *(0.0050, 0.2846), (0.0040, 0.2485), and (0.0033, 0.2295) respectively. Then the original F-score $F_A = 19.64 \times 10^{-3}$. Removing entries w_i, $1 \leq i \leq 4$ gives F-score 8.22×10^{-3}, 23.42×10^{-3}, 23.44×10^{-3}, and, 23.47×10^{-3} respectively. Hence greedy will choose entry w_4 in the first step. Given that w_4 is already chosen, removing w_1, w_2 and w_3 in addition gives resp. F-scores 8.90×10^{-3}, 31.21×10^{-3}, 30.70×10^{-3}. Hence the output of greedy is $\{w_4, w_2\}$. But the F-score after removing $\{w_2, w_3\}$ is 31.46×10^{-3}, which is better than the solution of greedy, and in this case also is the optimal solution. It can be verified that this example also shows non-optimality of other natural choices like choosing entries in the decreasing order of* fraction *or* count *of* `Bad` *results.*

NP-hardness: Simple Rules, Recall Constraint. Given positive integers $I = \{x_1, \cdots, x_n\}$ and an integer C, the goal of the subset-sum problem is to decide if there is $S \subseteq I$ such that $\sum_{x_i \in S} x_i = C$.

Construction. There are n dictionary entries $A = \{w_1, \cdots, w_n\}$ corresponding to $x_1, \cdots, x_n$. Each w_i has $f_{w_i} = cx_i$, and $p_{w_i} = \frac{1}{c}$ (there are $p_{w_i} f_{w_i} = x_i$ `Good` results). We choose $c = 3 + K$. In addition, there is a special entry w^* such that $f_{w^*} = 1$ and $p_{w^*} = 1$. The recall budget $\rho = \frac{1+B}{1+K}$. The goal is to check if there is a subset S' such that the F-score $F_{\overline{S'}} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$ and the recall $R_{\overline{S'}} \geq \rho$. The NP-hardness follows from the following lemma.

LEMMA 1. *There exists S' with $F_{\overline{S'}} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$ and $R_{\overline{S'}} \geq \rho$, if and only if the subset-sum instance has a solution.*

PROOF. (if) Let the subset-sum instance have a solution S such that $\sum_{x_i \in S} x_i = C$, i.e. $\sum_{x_i \in I \setminus S} x_i = K - C = B$. Let $S' = \{w_i : x_i \in S\}$. First, $R_{\overline{S'}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w} = \frac{1+\sum_{x_i \in I \setminus S} x_i}{1+\sum_{x_i \in I} x_i} = \frac{1+B}{1+K} = \rho$. Also $F_{\overline{S'}} = \frac{2\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S} f_w} = 2\frac{1+\sum_{x_i \in I \setminus S} x_i}{(1+\sum_{x_i \in I} x_i)+(1+\sum_{x_i \in I \setminus S} cx_i)} = \frac{2(1+B)}{(1+K)+(1+cB)}$.

(only if) Suppose S' is such that $F_{\overline{S'}} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$ and $R_{\overline{S'}} \geq \rho = \frac{1+B}{1+K}$. Wlog., assume that $w^* \notin S'$, otherwise we can exclude w^* from S' without decreasing the values of precision or recall (hence also the F-score). Hence all entries in S' corresponds to integers in the subset-sum problem, and Let the corresponding set of integers in the subset-sum problem be S. Now $R_{\overline{S'}} = \frac{\sum_{w \notin S} p_w f_w}{\sum_{w \in A} p_w f_w}$ $= \frac{1+\sum_{x_i \in I \setminus S} x_i}{1+K} \geq \frac{1+B}{1+K}$, or, $\sum_{x_i \in I \setminus S} x_i \geq B$ ——- (I) Again, $F_{\overline{S'}} = 2\frac{\sum_{w \notin S'} p_w f_w}{\sum_{w \in A} p_w f_w + \sum_{w \notin S'} f_w} = \frac{2(1+\sum_{x_i \in I \setminus S} x_i)}{(1+K)+(1+c\sum_{x_i \in I \setminus S} x_i)} \geq \frac{2(1+B)}{(1+K)+(1+cB)}$. Rearranging and simplifying both sides, $(c-2-K)\sum_{x_i \in I \setminus S} x_i \leq (c-2-K)B$. Since $c = 3+K$, $\sum_{x_i \in I \setminus S} x_i \leq B$ ——- (II)

From (I) and (II), $\sum_{x_i \in I \setminus S} x_i = B$, or, $\sum_{x_i \in S} x_i = K - B = C$. Therefore $S \subseteq I$ is a solution for the subset-sum problem. □

B. UPDATE RULES FOR EM

Here we derive the update rules for the EM-based algorithm described in Section 5.3. Let us denote the parameter vector at iteration t to be $\vec{\theta}^t$. Suppose $c_{w_i, \tau_j, t} = \mathbb{E}[y_j^\ell | \tau_j, \vec{\theta}^t]$, where $\tau_j \in \texttt{Succ}(w_i)$ and $\texttt{Prv}(y_j^\ell) = w_i$. We show that the update rules for parameters p_i has a nice closed form: $p_i = \frac{C_1}{C_1+C_2}$, where $C_1 = \sum c_{w_i,\tau_j,t}$ and $C_2 = \sum(1 - c_{w_i,\tau_j,t})$, where the sum is over $1 \leq j \leq N$ such that $\tau_j \in \texttt{Succ}(w_i)$. This gives $\vec{\theta}^{t+1}$, estimation of the parameters in the $t+1$-th round. The log-likelihood of the observed data

$$q(\vec{x}; \vec{\theta}) = \log P(\vec{x}|\vec{\theta}) = \sum_{j=1}^{N} \log P(\tau_i|\vec{\theta})$$

The complete data version of the problem will have the observed data $\vec{x} = \langle \tau_1, \cdots, \tau_N \rangle$ as well as the hidden data $\vec{y} = \langle y_j^\ell \rangle_{j \in [1,N], b \in [1,\ell]}$.

The expected log-likelihood of the complete data given the observed data $\vec{x}$ and current parameter vector $\vec{\theta}^t$ will be given by

$$\mathbb{E}[q(\vec{x}, \vec{y}; \vec{\theta})|\vec{x}, \vec{\theta}^t] = \sum_{j=1}^{N} \sum_{\vec{y}_j} \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] \log \Pr[\tau_j, \vec{y}_j|\vec{\theta}]$$
$$= \sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] \log \Pr[\tau_j, \vec{y}_j|\vec{\theta}]$$
$$= \sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] \log[\Pr[\tau_j|\vec{y}_j, \vec{\theta}] \Pr[\vec{y}_j|\vec{\theta}]]$$
$$= \sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] \log \Pr[\vec{y}_j|\vec{\theta}] \quad = K(say)$$

In the third step of the above derivation, for $\vec{y}_j$ such that $\phi_j(\vec{y}_j) \neq \tau_j, \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] = 0$, and in the fifth step, for $\vec{y}_j$ such that $\phi_j(\vec{y}_j) = \tau_j, \Pr[\tau_j|\vec{y}_j, \vec{\theta}] = 1$. Note that, given the current guess of parameters $\vec{\theta}^t = \langle p_1^t, \cdots, p_n^t \rangle$, $\Pr[\vec{y}_j|\tau_j, \vec{\theta}^t]$ can be easily computed and is a constant. For $\vec{y}_j$ such that $\phi_j(\vec{y}_j) = \tau_j$, $\Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] = \frac{\Pr[\vec{y}_j|\vec{\theta}^t]}{\Pr[\tau_j|\vec{\theta}^t]} = \frac{\prod_{y_j^\ell = 1} p^t_{\texttt{Prv}(y_j^\ell)} \prod_{y_j^\ell = 0}(1 - p^t_{\texttt{Prv}(z_j^\ell)})}{\sum_{\phi_j(\vec{z}_j) = \tau_j}\left[\prod_{z_j^\ell = 1} p^t_{\texttt{Prv}(z_j^\ell)} \prod_{z_j^\ell = 0}(1 - p^t_{\texttt{Prv}(z_j^\ell)})\right]}$ (we slightly abuse the notation here: $p^t_{\texttt{Prv}(y_j^\ell)} = p_i^t$, where $\texttt{Prv}(y_j^\ell) = w_i$).

Next we rewrite K by expanding and collecting coefficients of $\log p_i$ and $\log(1 - p_i)$ for every word w_i, $i \in [1, n]$. Then, $K =$

$$\sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t] \log \Pr[\vec{y}_j|\theta]$$
$$= \sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} V_{jt} \log[\prod_{\ell=1}^{\ell} \Pr[y_j^\ell|\theta]] \quad \left(\text{let } V_{jt} = \Pr[\vec{y}_j|\tau_j, \vec{\theta}^t]\right)$$
$$= \sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} V_{jt} \log[\prod_{y_j^\ell = 1} p_{\texttt{Prv}(y_j^\ell)} \prod_{y_j^\ell = 0} (1 - p_{\texttt{Prv}(y_j^\ell)})]$$
$$= \sum_{j=1}^{N} \sum_{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j} V_{jt} [\sum_{y_j^\ell = 1} \log[p_{\texttt{Prv}(y_j^\ell)}] \sum_{y_j^\ell = 0} \log(1 - p_{\texttt{Prv}(y_j^\ell)})]$$
$$= \sum_{i=1}^{n} \sum_{\substack{j=1, \tau_j \in \\ \texttt{Succ}(w_i)}}^{N} \sum_{\substack{\vec{y}_j : \phi_j(\vec{y}_j) = \tau_j, \\ \texttt{Prv}(y_j^\ell) = w_i}} V_{jt} \left[y_j^\ell \log p_i + (1 - y_j^\ell) \log(1 - p_i)\right]$$
$$= \sum_{i=1}^{n} \sum_{\substack{j=1, \tau_j \in \texttt{Succ}(w_i) \\ \texttt{Prv}(y_j^\ell) = w_i}}^{N} \mathbb{E}[y_j^\ell|\tau_j, \vec{\theta}^t] \log p_i + (1 - \mathbb{E}[y_j^\ell|\tau_j, \vec{\theta}^t]) \log(1 - p_i)$$
$$= \sum_{i=1}^{n} \sum_{\substack{j=1, \tau_j \in \texttt{Succ}(w_i) \\ \texttt{Prv}(y_j^\ell) = w_i}}^{N} c_{w_i,\tau_j,t} \log p_i + (1 - c_{w_i,\tau_j,t}) \log(1 - p_i)$$

In the above equations, $c_{w_i,\tau_j,t} = \mathbb{E}[y_j^\ell|\tau_j, \vec{\theta}^t]$. In the **E-step**, for every word w_i, and for every result $\tau_j \in \texttt{Succ}(w_i)$, we compute the expectation of y_j^ℓ (the ℓ-th bit of $\vec{y}_j$) given the current parameter vector $\vec{\theta}^t$, where $\texttt{Prv}(y_j^\ell) = w_i$. This can be computed from the probabilities $\Pr[\vec{y}_j|\tau_j, \vec{\theta}^t]$. So for every result τ_j, if ϕ_j takes b inputs, we have a vector of real numbers of size b after the E-step.

In the **M-step**, we maximize the expression K w.r.t. parameter vector $\vec{\theta}$ to get the next guess of the parameters $\vec{\theta}^{t+1}$.

For every $i \in [1, n]$, $\frac{\delta K}{\delta p_i} = 0 \Rightarrow \sum_{\substack{j=1, \tau_j \in \texttt{Succ}(w_i), \\ \texttt{Prv}(y_j^\ell) = w_i}}^{N} \left[\frac{c_{w_i,\tau_j,t}}{p_i} - \frac{1 - c_{w_i,\tau_j,t}}{1 - p_i}\right] = 0 \Rightarrow \frac{C_1}{p_i} - \frac{C_2}{1 - p_i} = 0$ (collecting the constants) $\Rightarrow p_i = \frac{C_1}{C_1 + C_2}$ □