I4E: Interactive Investigation of Iterative Information Extraction

Anish Das Sarma
Yahoo Research CA, USA
anishdas@yahoo-inc.com

Alpa Jain
Yahoo Research CA, USA
alpa@yahoo-inc.com

Divesh Srivastava
AT&T Labs-Research NJ, USA
divesh@research.att.com

ABSTRACT

Information extraction systems are increasingly being used to mine structured information from unstructured text documents. A commonly used unsupervised technique is to build *iterative information extraction (IIE)* systems that learn task-specific rules, called *patterns*, to generate the desired tuples. Oftentimes, output from an information extraction system may contain unexpected results which may be due to an incorrect pattern, incorrect tuple, or both. In such scenarios, users and developers of the extraction system could greatly benefit from an investigation tool that can quickly help them reason about and repair the output.

In this paper, we develop an approach for interactive post-extraction investigation for IIE systems. We formalize three important phases of this investigation, namely, *explain* the IIE result, *diagnose* the influential and problematic components, and *repair* the output from an information extraction system. We show how to characterize the execution of an IIE system and build a suite of algorithms to answer questions pertaining to each of these phases. We experimentally evaluate our proposed approach over several domains over a Web corpus of about 500 million documents. We show that our approach effectively enables post-extraction investigation, while maximizing the gain from user and developer interaction.

Categories and Subject Descriptors

H.0 Information Systems [**Investigation**]

General Terms

Algorithms, Experimentation, Management

Keywords

Information extraction, interactive investigation, debugging, explain, diagnose, repair

1. INTRODUCTION

Recent developments in knowledge-driven tasks such as question answering, opinion mining, and trend analysis have led to a significant interest in automatically extracting structured information from text documents such as newspaper articles, emails, etc. Along this direction, several information extraction (IE) systems have been built that generate an instance of some entity (e.g., company name, president of a country) or an instance of a relation (e.g., senators and their affiliations or books and their authors). Examples of real-life extraction systems include, Gate[1], DBLife[2], DIPRE [4], KnowItAll [15], Rapier [7], Snowball [2]. While existing extraction systems have been a fundamental block in bridging the gap between unstructured and structured information, oftentimes, output from an information extraction system may contain unexpected, and potentially incorrect data. The goal of this paper is to build an approach that would allow users to interactively understand IE results and rectify the system through feedback.

A commonly used information extraction technique for large-scale information extraction is called *iterative information extraction (IIE)* [19, 2, 30, 29, 28]. (We discuss other information extraction approaches later in this paper.) Iterative information extraction systems follow a working hypothesis that tuples from a relation tend to occur in similar contexts. Naturally, in most real-world extraction applications, it is not feasible to know apriori all possible contexts in which tuples of a relation may occur, thus, necessitating an *iterative* process: Starting with a relatively small set of seed tuples, these extractors iteratively learn *patterns* that can be instantiated to identify new tuples.

EXAMPLE 1. *Consider a relation* `actor⟨Movie, Actor⟩` *seeded by a tuple, ⟨Top Gun, Tom Cruise⟩ which occurs in the text, "Top Gun, movie starring Tom Cruise." Using this occurrence, an IIE system may learn the pattern, "⟨Movie⟩, movie starring ⟨Actor⟩." Extraction patterns are, in turn, applied to text to identify new instances of the relation at hand. For instance, the above pattern when applied to the text, "Star Wars, movie starring Alec Guinness," can generate a new instance, ⟨Star Wars, Alec Guinness⟩.*

At each iteration, the newly found tuples are augmented to the list of seed tuples, and the process terminates when a termination condition is met. In practice, extraction methods may assign an extraction score to each tuple and instead of augmenting all identified tuples to the seed tuples, it may augment only the top-k tuples as determined by the extraction scores.

EXAMPLE 1. *(***continued***) Upon adding the newly generated tuple ⟨Star Wars, Alec Guinness⟩ to our seed instances, we may learn a new extraction pattern, "⟨Movie⟩ films starring ⟨Actor⟩", from the sentence, "Star Wars films starring Alec Guinness." Let us*

SIGMOD'10, June 6–11, 2010, Indianapolis, Indiana, USA.

[1] `www.gate.ac.uk`
[2] `www.dblife.cs.wisc.edu`

refer to this pattern as p_1. Using p_1 on "Hollywood films starring Brad Pitt," we may generate a tuple ⟨Hollywood, Brad Pitt⟩ and similarly also generate "⟨Walt Disney, Johny Depp⟩".

Given a relation instance consisting of tuples such as in the example above, a natural question to ask is: *How was the tuple ⟨Hollywood, Brad Pitt⟩ generated?* Furthermore, given that we know that pattern p_1 generated it, some follow up questions may be: *Were there other tuples generated by p_1? Were there any other patterns generated by these tuples extracted using p_1? Can we distinguish between tuples that are only associated with p_1 and those that have patterns other than p_1 supporting it? If we eliminate p_1, which tuples will be eliminated from the output?* At first glance, some of these questions may appear identical; however, we shall later see that there are subtle differences in answering these questions.

There are multiple benefits of building an investigation tool for IIE, in addition to the obvious benefit of giving users useful insight into the extraction result. First, it helps in *explaining* to the user and system developer, the output from running an IIE over a collection of documents. (Traditionally, upon termination, the extraction method generally presents a set of tuples along with their extraction scores, offering limited or no insights into how a tuple was generated or how a given tuple impacted or interacted with other tuple generation.) Second, the investigation tool can help in *diagnosing* an IIE system. Similar to a program execution, IIE involves data flow and control flow which together reason about the output from the execution. In case of IIE, we may want to reason about the effect of altering thresholds (i.e, control flow) or removing an extraction pattern (i.e., data flow). Finally, investigations can lead to *repairing* the IIE system by fixing patterns and thresholds, thus improving the IIE result: Understanding the overall impact of a pattern or tuple on the output of an IIE system can help users understand the tradeoffs of eliminating parts of the system.

In this paper, we focus on the problem of building an *interactive investigation tool* for iterative information extraction, called I4E. We shall see that in addition to supporting investigations, I4E is able to provide guidance by showing *influential* tuples and patterns, and thus make recommendations to aid the repair process. Beyond the conceptualization of I4E to support the three phases of investigation—*explain*, *diagnose*, and *repair*—described above (and formalized later), the main contributions of this paper are:

- **Representing iterative IE:** We propose a principled graph-based network that integrates tuples, patterns, and various trace information at each iteration, for representing IIE. Our representation captures sufficient information to carry out complex investigation operations, yet is simple and succinct enough to scale to large-scale extraction scenarios such as the Web.

- **Explain, diagnose, and repair operations:** We present a set of fundamental queries that users may be interested in, for each stage of explain, diagnose, and repair. We give efficient algorithms for answering these questions. As we will see, some of these questions have *optimal* algorithms that are tractable at Web-scale, whereas some other questions have NP-hard worst-case complexity, for which we provide efficient *approximate* solutions.

- **Chaining operations:** We present techniques for *chaining* the fundamental operations of explain, diagnose, and repair to perform more sophisticated, interleaved investigations.

- **Experimental evaluation:** We perform an extensive experimental evaluation over six real-world datasets generated from a Web corpus of 500 million documents. Our experiments show that we are able to maximize the benefit of interactive investigation, thus significantly improving the quality of IE with minimal feedback. We examine the space and time overhead incurred by our investigation algorithms and show that our techniques can be applied efficiently at Web-scale.

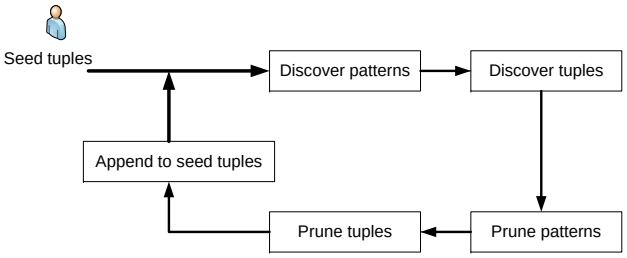

Figure 1: Architecture of iterative information extraction.

The rest of the paper is structured as follow: Section 2 presents necessary background on IIE, shows how we characterize it, and discusses the problem we focus on. Section 3 presents fundamental blocks for explain, diagnosis, and repair and Section 4 then shows how these blocks can be interleaved for chained investigations. Section 5 reports our extensive experimental evaluation. Section 6 presents related work, and we conclude with future work in Section 7.

2. ITERATIVE INFORMATION EXTRACTION

We begin our discussion by briefly describing the steps in an iterative information extraction (Section 2.1). Then, we present our approach to representing the execution of an IIE (Section 2.2), and discuss the problem we focus on (Section 2.3).

2.1 Background

The primary goal of information extraction systems is to obtain a set of *tuples* of a pre-defined relation, from a set of unstructured text *documents*. Under the iterative information extraction paradigm, these tuples are obtained by applying certain task-specific rules, called *extraction patterns*. Extraction patterns capture common ways of representing tuples of the target relation in a natural-language form.

IIE systems follow a working hypothesis that tuples from a relation tend to occur in similar contexts. Naturally, in most real-world extraction applications, it is not feasible to know apriori all possible contexts in which tuples of a relation may occur, thus, necessitating an *iterative* process. IIE is typically bootstrapped with a relatively small set of seed tuples T from the target relation, a (potentially empty) set of patterns P, and a database of documents D, which is typically a slice of the Web. Starting with the seed tuples an IIE system iterates over the following main stages, as shown in Figure 1:

(1) Discover patterns: For each tuple $t \in T$, an IIE system identifies occurrences of each tuple in the documents in D. Based on the textual context in which t occurs, candidate extraction patterns are identified. For each tuple t, we denote by $P_p(t)$ the set of patterns produced using t; similarly, for each pattern p, we denote by $T_g(p)$ the set of tuples that generated pattern p.

(2) Discover tuples: Apply the current set of patterns on each document in D and obtain a set T' of new tuples. For each tuple t, we denote by $P_g(t)$ the set of patterns that generated t; similarly, for each pattern p, we denote by $T_p(p)$ the set of tuples produced by p.

(3) Prune patterns: Unfortunately, information extraction is a noisy process and oftentimes, we may learn unreliable patterns that can, in turn, produce erroneous tuples. Therefore, IIE systems assign a *confidence score* to each pattern based on individual tuples that generate the pattern as well as the collective set of tuples produced by the pattern.

DEFINITION 2.1. **[Pattern confidence score]** *Consider a pattern p that was generated using a tuple t after processing a document d in D. Let $s_p(t, o)$ be a function that assigns a score to p based on the tuple t and the context o in which pattern p occurs. After processing all the documents in D, let $T_g(p)$ be the set of tuples that generated p and $T_p(p)$ be the set of (new) tuples produced using p. The* confidence score *for p, $S_p(p) = F_p(T_g(p), T_p(p), s_p)$.* □

Several methods have been proposed to assign confidence scores to patterns [2]; As a concrete example, F_p may be defined as the fraction of tuples in $T_p(p)$ that occur in our seed set of tuples. In this paper, we assume that the score function $F_p(T_g(p), T_p(p), s_p)$ has been provided, such as in [2, 13, 30, 14]. Upon assigning a confidence score to each pattern in P, IIE systems eliminate from P all patterns with confidence scores below a threshold τ_p.

(4) Prune Tuples: Analogous to pattern confidence scores, discovered tuples are also assigned a *confidence score*. The confidence score of a tuple may depend on the confidence score of the patterns that generated it; additionally, we may also corroborate evidence and confidence scores from different patterns to assign an aggregate tuple confidence score.

DEFINITION 2.2. **[Tuple confidence score]** *Consider a tuple t that was generated using a pattern p after processing D. Let $s_t(p, o)$ be a function that assigns a score to t based on the pattern p and the context o in which tuple t occurs. After processing all the documents in D, let $P_g(t)$ be the set of patterns that generated t. The* confidence score *for t, $S_t(t) = F_t(P_g(t), s_t)$.* □

We assume that a scoring function to assign a tuple confidence score has been provided [2, 13, 30, 14]. After assigning a confidence score to each tuple in T', IIE systems eliminate from T' all tuples that have a score less than a threshold τ_t, then set $T = T \cup T'$. For cases where the minimum confidence threshold τ_t is not constant across different iterations, we may need to recompute the confidence scores for all the tuples in T' as well as T to eliminate tuples with scores less than τ_t.

Note that the four steps outlined above present a high-level overview of IIE. Several details are omitted. For instance, the various steps need not all be performed in every iteration, or may be performed in different orders. Further, we may wish to perform pruning only at the end of all iterations, or in batches. And, at each iteration, for efficiency we may choose to perform discovery (of patterns or tuples) only based on "new" tuples or patterns obtained in this iteration. These challenges and tradeoffs are not a focus of the paper and only touched upon when they impact our results; our goal in this paper is to interactively investigate any IIE, irrespective of which stages were applied when.

We summarize our notation in Table 1 and discuss examples that illustrate the above concepts.

EXAMPLE 2. *Consider the task of extracting a* `disease outbreak` ⟨`disease, location`⟩ *[35] relation, for which an IIE has learnt an extraction pattern p_1 ='⟨d⟩ outbreak sweeping throughout ⟨l⟩,' where d and l are instantiated over values for disease and location, respectively. A simple example of tuple confidence score function, i.e., $s_t(p, o)$, assumes an edit-distance-based similarity matching between the context of a candidate tuple and the pattern. Specifically, if* max *is the maximum number of terms in an extraction pattern and x is the number of word transformations for a tuple context to match the pattern, then the confidence score is computed as $1 - \frac{x}{max}$ ($x \leq$ max). Given a text fragment, "*A H1N1 flu outbreak sweeping throughout Mexico alarmed ...*," and* max $= 4$, *the extraction system generates the tuple $t_1 = \langle$H1N1 flu, Mexico$\rangle$ with a score of 1.0 ($x = 0$).*

Symbol	Description
P	Set of patterns learned by IIE
T	Set of tuples produced by IIE
$P_g(t)$	Set of patterns that generated tuple t
$P_p(t)$	Set of patterns produced using tuple t
$T_g(p)$	Set of tuples that generated pattern p
$T_p(p)$	Set of tuples produced using pattern p
F_p	Score function to assign confidence to patterns
F_t	Score function to assign confidence to tuples

Table 1: Notation

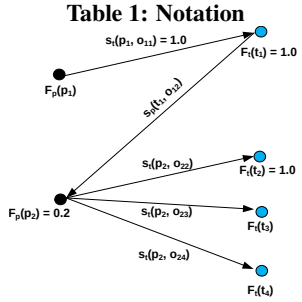

Figure 2: Sample EBG graph.

A tuple may also be associated with multiple $s_t(p, o)$ scores, since facts are often repeated across text documents or a tuple may be produced by multiple non-identical patterns [13].

EXAMPLE 2. **(continued).** *Another pattern, '⟨d⟩ outbreak is sweeping ⟨l⟩' may process the same text fragment to generate the same tuple but now with a score of 0.5 ($x = 2$). The total tuple confidence score for t_2 is aggregated using $F_t(P_g(t), s_t)$.*

2.2 Characterizing IIE

In this section, we describe a simple graph-based representation to capture the necessary tracing information in an IIE, so as to allow users and developers to effectively carry out post-extraction investigations. Given an iteration I_i, we focus on the set of tuples and patterns that were retained at the end of iteration I_i[3]. To characterize iteration I_i (and all other iterations), we define an *enhanced bipartite graph (EBG)*:

DEFINITION 2.3. *[EBG] An* enhanced bipartite graph (EBG) *$G = (P, T, E_1, E_2)$ is a directed bipartite graph consisting of two classes of nodes: the set P of* patterns' *nodes ("p" nodes) and the set T of* tuples' *nodes ("t" nodes), and a set of directed edges $E = E_1 \cup E_2$, where $E_1 \subseteq P \times T$ and $E_2 \subseteq T \times P$. An edge $(p, t) \in E_1$ (denoted $p \rightarrow t$) connects a p node to a t node, and depicts that tuple t was generated by applying pattern p; an edge $(t, p) \in E_2$ (denoted $t \rightarrow p$) connects a t node to a p node and depicts that pattern p was generated using tuple t. Multiple edges may originate from or reach to a node.* □

Each node is annotated with the confidence score assigned to the pattern or tuple represented by the node as well as the first iteration in which it was created. Each edge in the graph stores information about the score generated using the tuple and pattern that it connects, and also maintains the iteration number i in which the edge

[3]In this paper we focus on actual IIE execution that took place. Extending our approach to support "why not"-style questions in the spirit of [21] would require tracing eliminated tuples and patterns.

was generated. We also store separately the threshold that was applied at each iteration to prune out tuples. As an example, consider the EBG graph in Figure 2 for patterns p_1 and p_2 from Example 2. Pattern p_1 generated t_1 which, in turn, generated p_2, as indicated by the directional edges. The confidence scores for t_1 and p_2 are 1.0 and 0.2 respectively. We note that EBG may contain cycles; for instance, the tuple t_3 may re-generate pattern p_1.

At the end of an iteration I_i, for each pattern p and tuple t to be retained, our algorithm to generate EBG, $G = (P, T, E_1, E_2)$ performs the following steps:

1. If $p \notin P$, add a node p to P. If $t \notin T$ add a node for t to T.
2. Introduce into E_1 an edge (p, t), from p to a tuple t, if t was produced using p.
3. Annotate each edge $p \rightarrow t \in E_1$ with the score of t obtained from p applied to the document from which t was derived.
4. Introduce into E_2 an edge (t, p), from t to a pattern p if p was produced using t.
5. Annotate each edge $t \rightarrow p$ with the score of p obtained from t applied to d.

We now develop a suite of techniques to enable explain, diagnose, and repair, collectively named EDR, each extending the EBG representation.

2.3 The Need for an Investigation Approach

EXAMPLE 2. *(***continued***) Suppose tuple t_1 is used to generate a new pattern after identifying the tuple's occurrence in the text '...* checking of H1N1 flu is Mexico's way of avoiding *...'. Suppose the IIE, using $F_p(T_g(p), T_p(p), s_p)$, assigned a confidence score of 0.2 to $p_2 = \langle d \rangle$ is $\langle l \rangle$. Applying p_2 to* 'Measles is America' *may generate an incorrect tuple t_2 = '⟨Measles, America⟩' with confidence score of 1.*

As discussed earlier, users may be interested in investigating the sequence of steps leading to any tuple. Specifically, users may be interested in learning about t_2. As an explanation for t_2, we could present the set of patterns, $\{p_2\}$, that generated it, or the confidence score for patterns τ_p used in this iteration, e.g., $\tau_p = 0.1$. Armed with this information, the user may run further *diagnosis* via follow-up queries such as, *What is the influence of eliminating p_2?*, for which we may present the set of affected patterns and tuples. Suppose the answer to this query is that only pattern p_2 and tuple t_2 are affected. Now, the user may want to *repair* the output by modifying the value for τ_p or eliminating p_2. As a key contribution, we formalize three main phases of an investigation approach:

1. *Explain* involves retracing the "history" of IIE execution starting from a set of extracted tuple(s) (Section 3.1).
2. *Diagnose* studies the effect of various components of IIE and identifies components on which feedback would be most useful in improving the result of IIE (Section 3.2).
3. *Repair* modifies the result of extraction *incrementally* after rectifying the problematic components of extraction (i.e., deleting some patterns, updating their scores, or changing some thresholds) (Section 3.3).

We now describe the problem that we focus on in this paper.

PROBLEM 2.1. *Let $I_1, I_2, \cdots, I_n$ be consecutive iterations of an iterative information extraction over a text database D with score functions to assign confidence scores for extraction patterns and tuples. Develop techniques to efficiently support* explanations*,* diagnosis*, and* repair *of the output generated by the extraction system after any iteration I_k, for any tuple t, or for any pattern p.*

The problem we address is generic and can subsume several other scenarios. For instance, in addition to patterns and tuples, explanations may involve *source text* that generated a pattern or a tuple. For our discussion, we assume that we are given as "black boxes," the scoring functions for assigning pattern and tuple confidence scores, and information pertaining to the text source is available to the black box. Nevertheless, the algorithms and representation presented in this paper do not rely on this interaction, and we believe that our algorithms can be easily extended to support text source for cases where such a black box interaction is unavailable. We note that any subset of the three stages above may be performed in any sequence. The rest of the paper presents algorithms for each stage which are independent of other stages.

3. EXPLAIN, DIAGNOSE AND REPAIR

We now lay out the fundamental building blocks for enabling explanations (Section 3.1), diagnosis (Section 3.2), and repair (Section 3.3). In this section, we focus on "one-step EDR"; more complex investigations based on *chaining* these fundamental blocks is considered in Section 4. Our discussion for each stage presents illustrative questions that naturally arise in this phase, followed by algorithms to answer the questions using EBG.

3.1 Explaining Extraction Output

Intuitively, given a set of tuples, explanation traverses its lineage backward, and so we are interested in building the history for each tuple. More concretely, we need to address the following questions:

- E1 Given a tuple t, determine the set $P_g(t)$ of patterns that generated t, i.e., contributed to increasing t's score?
- E2 Given a tuple t, which pattern contributed to t the most? That is, dropping which pattern would reduce the score of t the most?
- E3 Given a tuple t, which was the first iteration that discovered t?
- E4 Determine the most *influential* patterns in the entire IIE result, i.e., rank the patterns in order of their *impact* on the final result. We consider various definitions of *impact* for a pattern p: (a) $I_t(p)$, using the number of result tuples p produced, i.e., $|T_p(p)|$, (b) $I_o(p)$, using the number of tuples *only* p produced, and (c) $I_s(p)$, using the total score contribution of p aggregated over all tuples. Roughly, E4 aims to answer E1 and E2 for the *entire batch of tuples* in the result. In addition to ranking all patterns, we also consider the problem of returning the set of K most influential patterns based on each of the three measures of impact.

Before proceeding, we note that some EDR operations (notably E1–E3 above) are easily answered using an EBG. We briefly discuss these operations, before proceeding to the significantly more challenging E4; also recall, we consider chaining to perform more sophisticated investigations in Section 4.

3.1.1 Algorithms for Enabling Explanations

Given the EBG $G = (P, T, E_1, E_2)$ and tuple $t \in T$, we can return as answer to E1 all patterns $p \in P$ with an outgoing edge to t. Let $P_g(t) = \{p_1, \ldots, p_k\}$ be the set of all patterns that generated t. For E2, we sort patterns associated with each incoming edge to t, by their contribution s_t and return as answer the highest ranking pattern[4]. Answers for E3 can be trivially generated from G.

[4]We assume the tuple scoring function, s_t, is *uniform*, i.e., does not discriminate between different patterns, and is *monotonic*. If the

Require: $G = (P, T, E_1, E_2)$, where $P = \{p_1, \ldots, p_n\}$
1: $\forall i : c_1(p_i) = c_2(p_i) = c_3(p_i) = 0$
2: **for** $i \in 1 \ldots n$ **do**
3: **for** $(p_i, t_j) \in E_1$ **do**
4: $c_1(p_i)$++
5: **if** $indeg(t_j) == 1$ **then**
6: $c_2(p_i)$++
7: **end if**
8: $c_3(p_i) += f(s_i, d_{(i,j)})$
9: **end for**
10: **end for**
11: *Sort* P based on c_1, c_2, c_3 and return these rankings.

Algorithm 1: Algorithm for E4

We now turn to answering E4. Algorithm 1 shows how to solve E4, given the EBG G, assuming a primitive function *indeg* that returns the in-degree of any node in G. In effect, we maintain three counts for a pattern, each corresponding to the three notions of impact. For $I_t(p)$, we are interested in ranking purely based on $|T_g(p)|$, and thus, we rank patterns based on the number of edges in G; for $I_o(p)$, we count the tuples only contributed to by a single pattern, and thus, we discard edges to t nodes in G with in-degree greater than 1; for $I_s(p)$, we rank by score contribution, and thus, for every pattern p_i we scan each outgoing edge p_i, t_j and aggregate the scores $s_t(p, t_j)$ associated with it. We show the correctness and complexity of Algorithm 1 using the following theorem.

THEOREM 3.1. *Given an EBG $G = (P, T, E_1, E_2)$, Algorithm 1 returns the solution to E4 in $O(M + N \log N)$, where $N = |P|$ and $M = |E_1|$, assuming $f(\cdot)$ returns the score contribution of a pattern to a tuple in $O(1)$.*

Next we consider answering a variant of E4 where we are interested in returning the set of K patterns that have the highest aggregate impact, as determined by each of the three measures, namely, $I_t(p)$, $I_o(p)$, and $I_s(p)$. For this, we reuse Algorithm 1, and then return the K patterns in sorted order. Interestingly, as captured by the following theorem, while picking the top K using Algorithm 1 gives us the optimal solution based on impact measures $I_o(p)$ and $I_s(p)$ it only returns an *approximate* solution for $I_t(p)$. The following theorem also shows that finding the optimal solution for $I_t(p)$ is intractable, hence the approximate solution returned by Algorithm 1 is a practical solution.

THEOREM 3.2. *Given an EBG $G = (P, T, E_1, E_2)$ where $N = |P|$ and $M = |E_1|$:*

1. *We can obtain optimal set of K patterns based on impact measures $I_o(p)$ and $I_s(p)$ by picking the top K patterns based on the sorted order of Algorithm 1 in $O(M+N \log N)$.*
2. *Finding the optimal set of K patterns based on impact measure $I_t(p)$ is NP-complete in M and N.*
3. *Returning the top K patterns based on the sorted order of Algorithm 1 gives a $(1 - \frac{1}{e})$-approximation to E4 based on impact measure $I_t(p)$.*

PROOF.
1. Note that the impact of a pattern p_i based on measure $I_o(p)$ or $I_s(p)$ is independent of whether pattern p_j is picked in the set for E4. Therefore, the total impact of a set of patterns is given by the sum of impacts of each pattern. Therefore, greedily picking the best K patterns based on their independent impacts yields an optimal solution to E4.

2. To show NP-hardness of E4 under impact measure $I_t(p)$, we give a reduction from the NP-hard set cover problem [16]: Given a universal set $U = \{1, 2, \ldots, n\}$ and subsets $S_i \subseteq U$, $1 \leq i \leq m$, find the fewest subsets whose union is U. We reduce set cover to E4 by constructing m patterns $p_1, \ldots, p_m$ and n tuples $t_1, \ldots, t_n$. p_i contributes to tuple t_j if and only if $j \in S_i$. There exists a solution to the set cover of size K if and only if there is a set of K patterns whose combined impact under measure $I_t(p)$ is all n tuples, proving the NP-hardness of E4 under impact measure $I_t(p)$. Further, it can be seen easily that E4 is in NP: Given a set of patterns, we can compute in PTIME the total impact of the set of problems. Therefore, E4 is NP-complete.

3. Using an inverse construction as in (2) above, we can reduce E4 to an identical K-coverage problem, whose goal is to pick k sets such that the union of the k sets is the largest possible. Since this construction is an L-reduction (approximation-preserving reduction), and since the greedy algorithm for K-coverage yields a $(1 - \frac{1}{e})$-approximation [20], our result follows. ∎

3.2 Diagnosing the Extraction Output

Intuitively, diagnosis performs a "forward pass" to determine all extraction results that were affected by specific pattern(s) or threshold(s). The fundamental questions answered by this phase are:

- D1 Given a pattern p, determine the set $T_p(p)$ of tuples produced by p.
- D2 Given a pattern p, determine all tuples that would get eliminated if p is removed. (Note the subtle difference between D1 and D2. D1 asks for all tuples generated by p *and possibly other patterns*, while D2 asks for all tuples contributed to *only* by p.)
- D3 Given a pattern p, which was the first iteration that discovered p?
- D4 Find a set of K most *influential* tuples, i.e., find the set of K tuples that are contributed to by the largest number of patterns. Note this question is analogous to E4 from Section 3.2. However, here we are only interested in finding K tuples and not ranking all tuples: the total number of tuples is likely to be large and ideally, we would like to present users with a small set of tuples to obtain feedback on whether these tuples are correct. Therefore, these tuples have to be carefully picked to maximize the impact of the feedback on the output.

3.2.1 Algorithms for Enabling Diagnosis

D1, D2, and D3 are answered in a very similar fashion as the corresponding explanation questions. Given the EBG $G = (P, T, E_1, E_2)$ and pattern $p \in P$, to answer D1, we identify $T_p(p)$ as all $t \in T$ with incoming edges (p_i, t) to t. To answer D2, we eliminate from $T_p(p)$ all tuples that have an in-degree of greater than one, while D3 is answered simply based on the iteration numbers stored in G.

Next let us turn to D4, which is the most challenging diagnosis question. In fact, finding the optimal set of K tuples that are influenced by the largest number of patterns is intractable in general. Once again, greedily picking the best K tuples based on their individual number of patterns contributing to them gives an efficient approximate solution. We don't repeat a detailed algorithm here as it is very similar to Algorithm 1.

THEOREM 3.3. *Given an EBG $G = (P, T, E_1, E_2)$ where $N = |T|$ and $M = |E_1|$, finding an optimal solution to D4 is NP-complete in M and N. A greedy algorithm picking the top K tuples*

tuple scoring function is a complete black-box, we must evaluate the score by dropping one pattern at a time and pick the pattern that caused the largest reduction in the tuple score.

Require: $G = (P, T, E_1, E_2)$, K
for $i \in |P| \ldots 1$ **do**
  for $S \subseteq P, |S| = i$ **do**
    for Every partitioning S_K of S into K partitions **do**
      bool=*true*
      $R = \emptyset$
      for Every partition $s \in S_K$ **do**
        if $\exists t \in T$ s.t. every $p \in s$ contributes to t **then**
          $R = R \cup \{t\}$
        else
          bool=*false*
          break
        end if
        if bool **then**
          return R; **exit**
        end if
      end for
    end for
  end for
end for

Algorithm 2: Algorithm for D4 when the number of patterns is small compared to the number of tuples.

yields a $(1 - \frac{1}{e})$-approximation to the total number of contributing patterns.

PROOF. The hardness proof is by a reduction from set cover, similar to that for Theorem 3.2, with sets and elements interchanged. Details are omitted. Similarly, the approximation guarantee follows from the greedy approximation of the K-coverage Problem. ∎

While the above presents a practical solution to D4, we can obtain an even better algorithm based on the fact that in practice the total number of patterns is significantly smaller than the number of tuples. A single pattern typically generates many tuples, while a single tuple is contributed to by few patterns. Next we present an efficient algorithm when the number of patterns is small, whereas the number of tuples can be large.

Consider an EBG $G = (P, T, E_1, E_2)$ with $M = |P|$ and $N = |T|$, and $M \ll N$. Algorithm 2 presents an efficient exact solution for D4[5]. Intuitively, given the input K, we consider all subsets $S \subseteq P$ of patterns that may be in the contributing set for K tuples, in descending order of the size of S. Whenever we find an S that contributes to K tuples, we return S and terminate. For a given S, to check whether there are K tuples that are contributed to by S we use the following property: If all patterns in S contribute to at least one tuple in a set of K tuples, then there exists a partition of S into K sets $\{S_1, \ldots, S_K\}$ such that for each S_i there is a tuple t_i that is contributed to by every pattern in S_i. Therefore, Algorithm 2 returns an optimal solution of D4. The following theorem establishes the running time complexity of Algorithm 2. Note that the running time is linear in N when the number of patterns is constant, and nearly polynomial (i.e., exponential in $\log N \log \log N$) when M is $O(\log N)$. (We shall see in Section 5 that our experiments over real-world datasets corroborate the assumption of the following theorem. For example, in the `directors` domains, the top-5 patterns generated more than 150K tuples (see Section 5).)

THEOREM 3.4. *Given an EBG $G = (P, T, E_1, E_2)$ with $M = |P|$, $N = |T|$ tuples, Algorithm 2 returns an optimal solution to D4 in $O(NM2^M K^{M+1})$. In particular, if M is a fixed constant, the running time is $O(N)$ since $K \leq M$. Alternatively, if $M = O(\log N)$, Algorithm 2 runs in $O(N^{\log \log N})$.*

[5]Details of standard procedures like finding subsets of a given size, and partitioning a set into K pieces are omitted from Algorithm 2.

PROOF. The optimality of Algorithm 2 is evident from the discussion above. The total number of subsets of P is 2^M. For each subset S, which is of size $\leq M$, we consider all possible partitions of S into K partitions. Finally, for each partition, we need to check if there is a single tuple that is contributed to by each pattern in the partition. This check for all partitions can be performed in $O(NMK)$. Therefore, the total running time is $O(NM2^M K^{M+1})$. If M is $O(\log N)$, the running time is

$$N \log N 2^{\log N} K^{\log N + 1} \leq N^2 \log N (\log N)^{\log N + 1}$$
$$= N^2 \log^2 N (\log N)^{\log N} \sim O(N^{\log \log N})$$ ∎

The above algorithms can subsume several other notions of influence for D4. For instance, if we used tuple confidence scores to measure influence, there is a tractable algorithm to solve D4 optimally, as in the case of Theorem 3.2. If we considered influence based on $|P_p(t)|$, size of the set of patterns produced by tuples (as opposed to set $P_g(t)$ of patterns that generated tuples), we use similar techniques as above using E_2 edges in the EBG instead of E_1 edges. Finally, if we want influence based on a combination of patterns that generate a tuple as well as those produced by it, we consider the set $E = E_1 \cup E_2$ of edges.

3.3 Repairing the Extraction Output

To incrementally revise the result of IIE, the fundamental operations we consider are:

- R1 One or more patterns are deleted.
- R2 The score of one or more patterns is modified. (Setting the score of a pattern to 0 is equivalent to deleting it.)
- R3 Some thresholds on tuples or patterns are modified.
- R4 Each of a (small) set of tuples has been annotated (by a user) as correct or incorrect. We would like to modify the IIE so that the users annotations are respected, i.e., revise the scores of other tuples and patterns to reflect the users annotations. (Note that similar annotations of patterns by a user are executed by setting the score of correct patterns to 1, deleting incorrect patterns and then using R1 and R2.)

3.3.1 Algorithms for Enabling Repair

Given the EBG $G = (P, T, E_1, E_2)$ and a pattern $p \in P$ that needs to be deleted, we solve R1 as follows. Consider the set $T_p(p) = \{t \in T | (p, t) \in E_1\}$ and the set $only(p) = \{t \in T | (p, t) \in E_1, \text{in-deg}(t) = 1\}$. All tuples in $only(p)$ are deleted, and the score of every tuple in $(T_p(p) - only(p))$ is recomputed. (Clearly, deleted tuples now may cause further deletion of patterns, and so on. Recall in this entire section we only consider one-step modifications. Chaining sequences of modifications is discussed in Section 4.) To solve R2, we recompute the score of every tuples in $T_p(p)$ using Definition 2.2. Note that if p's score is increased, a tuple t that was pruned out in earlier iterations may now satisfy the threshold, because of a boosted score due to p. Such tuples aren't added in the EBG for efficiency. We briefly discuss this point further in future work (Section 7).

To solve R3, we have two options: (1) Augment the EBG to explicitly record, for each pattern and tuple, the sequence of scores through every iteration; (2) Use the iteration numbers stored in EBG. Option 2 requires more time to solve R3 but has a lower space overhead. On the other hand, if pattern and tuple scores don't change frequently, the space overhead of Option 1 isn't too much, and the running time of R3 is lower. To solve R3, when we store the sequence of scores for each pattern and tuple (Option 1 above), we simply remove tuples and patterns that did not satisfy the modified threshold at the specified iteration.

Require: $G = (P, T, E_1, E_2)$, iteration I, modify $\tau \rightarrow \tau'$
if $\tau' > \tau$ **then**
  for $t \in T$ **do**
    $P_I(t) = \emptyset$
    for $(p, t) \in E_1$ **do**
      if $iter(p, t) \leq I$ **then**
        $P_I(t) = P_I(t) \cup (p, t)$
      end if
    end for
    if $P_I(t) \neq \emptyset$ **then**
      Recompute score $S_t(t) = F_t(P_I(t), s_t)$ (Def. 2.2).
    end if
    if $S_t(t) < \tau'$ **then**
      remove(t, G).
    end if
  end for
else
  return
end if

Algorithm 3: Algorithm for R3: Repairing the threshold for tuple pruning.

When we store only the iteration number on every edge and the threshold applied at each iteration (Option 2 above), we distinguish two cases for R3. First, when the threshold in some iteration for tuples or patterns is reduced, no change is made: All tuples and patterns that survived the pruning continue to remain in the result. As mentioned earlier, some tuples or patterns that were eliminated may now survive pruning, but these tuples and patterns weren't stored in the EBG. Second, suppose the threshold at iteration I of tuples is increased from τ to τ'. (Modifications to pattern thresholds are handled in a similar fashion.) We consider the set T_I of all tuples that were born in iteration I, obtained by selecting from G all tuples whose incoming edges have labels I and greater only. We recompute the scores of these tuples and eliminate tuples whose revised scores are below τ'. Further chaining of the effect of removing these tuples is considered in Section 4. Algorithm 3 summarizes the algorithm for solving R3 when the threshold for tuples in iteration I is modified from τ to τ'. It assumes a function $iter(e)$ that returns the iteration number at which edge e was created. Further, the algorithm uses a function *remove* that removes a node, its edges, and optionally applies chaining. The following fairly evident theorem states the correctness and complexity of Algorithm 3 assuming suitable indexes to retrieve edges of nodes quickly.

THEOREM 3.5. *Algorithm 3 correctly repairs IIE for R3 in* $O(|T| + |E_1|)$.

Finally, let us consider R4. Suppose a user annotates a set T^+ of tuples as correct and a set T^- of tuples as incorrect. We modify the execution of IIE as follows. For every pattern p that generates a tuple in T^-, we delete pattern p and solve R1. For every tuple in T^+, we set the score of T^+ to 1, delete all tuples in T^-, and recompute the score of every pattern p generated through some tuple(s) in $T^+ \cup T^-$. We then apply R2 to repair IIE based on the new scores of affected patterns. We note an important subtlety underlying R4: A set of annotations T^+ and T^- may be *inconsistent*, e.g., there may not exist any assignment of scores to patterns that are consistent with all tuples in T^+ being present (with score 1) and all tuples in T^- being absent. As an extreme example, if a tuple $t \in T^+$ and $t \in T^-$, we have an inconsistency which can be resolved by appropriately notifying the user. In case of such inconsistent input, I4E is able to use EBG to pinpoint the conflicting patterns and tuples causing the inconsistency; these conflicts are then presented to the user or developer for resolution.

Require: $G = (P, T, E_1, E_2)$, tuple t
$Q_t = \{t\}$, $Q_p = \emptyset$, $\mathcal{P} = \emptyset$
traversed=$\emptyset$
while $((Q_t \neq \emptyset)$ OR $(Q_p \neq \emptyset))$ **do**
  if $Q_t \neq \emptyset$ **then**
    $tn = pop(Q_t)$
    for $p' \in (E(tn) - traversed)$ **do**
      $Q_p = Q_p \cup \{p'\}$
    end for
    traversed=*traversed* $\cup \{tn\}$
  else
    $pn = pop(Q_p)$
    $\mathcal{P} = \mathcal{P} \cup \{pn\}$
    for $t' \in (E(pn) - traversed)$ **do**
      $Q_t = Q_t \cup \{t'\}$
    end for
    traversed=*traversed* $\cup \{pn\}$
  end if
end while
return $\mathcal{P}$

Algorithm 4: Algorithm for E5: Finding all patterns that contribute to tuple t

4. CHAINING EDR OPERATIONS

So far, we looked at operations fundamental to interactive IIE, and provided "one-step" algorithms that traverse a fixed number of directed edges in the EBG. For instance, in response to E1, we were interested only in patterns that contributed directly by generating tuple t; however, there may be a sequence of tuple and pattern generations leading up to tuple t. Alternatively, we may want to know the impact of modifying the score of p not just on tuples p generated directly, but also on tuples indirectly generated from p through a sequence of patterns and tuples. In this section, we consider complex (multi-step) investigations by interleaving the fundamental operations. Obviously, the total number of possible investigative questions is infinite, and hence it is impossible to enumerate all possible algorithms. However, using a series of examples, we argue that the explain, diagnose, and repair operations from Section 3 form the basis for more complex investigations through *chaining*. Next, we consider complex interactions based on each of the three phases—explain, diagnose, and repair—a user may want to perform on the result of IIE, and show how they are implemented by chaining the individual operations from Section 3.

4.1 Chaining Explanations

We study two examples of chained explanations, obtained by extending E1 and E4 from Section 3.1 respectively. First consider the following extension of one-step E1 from Section 3.1:

E5 Given a tuple t, find all patterns that (directly or indirectly) contributed to t.

Given an EBG $G = (P, T, E_1, E_2)$ and tuple $t \in T$, we are interested in $\mathcal{P} = \{p \in P \mid \exists \text{ path from } p \text{ to } t\}$. We can obtain the set $\mathcal{P}$ using a standard traversal of G through edges in $E_1 \cup E_2$, avoiding cycles, to find all reachable nodes. We restrict the set of reachable nodes to those in P to obtain the solution to E5. Algorithm 4 describes the traversal and the theorem below summarizes our result for E5. Note that Algorithm 4 assumes a function $E(\cdot)$ that performs the explanation from Section 3.1 for a tuple (pattern resp.) to return the set of patterns (tuples resp.) that generated it.

THEOREM 4.1. *Algorithm 4 returns the correct solution to E5 in* $O(N \log N)$*, where* $N = |G|$ *gives the total number of nodes and edges in the EBG* G.

Note that E5 only asked for the *set* of contributing patterns, and not the exact nature of contribution. For instance, we may wish to know that pattern p_1 contributes to t_2 as p_1 generated tuple t_1, which generated pattern p_2, which in turn generated t_2. In general, we may wish to obtain the entire "derivation tree" of a tuple. The traversal of G can be extended easily to record edges, so as to obtain the subgraph of G that has a directed path to the input tuple t.

Next we briefly consider the extension of one-step E4 from Section 3.1. Under impact measure $I_o(p)$, the result of one-step E4 is identical to chaining because $I_o(p)$ only considers tuples that are generated by exactly one pattern. If however, we are interested in impact measure $I_t(p)$, given a pattern we need to determine all tuples that p (directly or indirectly) contributed to. That is, given the EBG $G = (P, T, E_1, E_2)$, $p \in P$, we need to determine all $t \in T$ such that there exists a path from p to t. We can determine the set of all reachable tuples for every p using a standard breadth-first shortest path algorithm [11]. Once the set of reachable tuples has been obtained, we can simply rank all patterns, or employ an approach similar to Theorem 3.2 to pick the K most influential patterns. Finally, we solve chained E4 under impact measure I_s as follows. For every pattern p, we update p's score to 0 and compute the updated scores of all tuples (as shown under chained repair below), and aggregate all tuple scores. We then pick the K most influential patterns or rank them, as appropriate.

4.2 Chaining Diagnosis

We consider the following extension of D2 from Section 3.2:

D5 Given a pattern p, find all tuples that would get deleted if p were removed.

In Section 3.2 we only considered tuples that were directly generated from a pattern. However, if a pattern is deleted, all tuples generated from it are deleted, which in turn may cause several other tuples and patterns to be deleted. Just as E5 was solved using the building block of E1, D5 can be solved in an identical fashion using the building blocks corresponding to D2: Determining all tuples (patterns respectively) that would get deleted if a given pattern (tuple respectively) is deleted. As in Algorithm 4, we traverse G and iteratively find tuples and patterns that are necessarily deleted; the exact algorithm is omitted.

THEOREM 4.2. *Given an EBG $G = (P, T, E_1, E_2)$, pattern p, D5 can be solved in $O(N \log N)$, where $N = |G|$ gives the total number of nodes and edges in G.*

4.3 Chaining Repair

We consider repairing the score of all affected tuples when a pattern's score is modified, an extension of R2:

R5 When the score of a pattern p is modified, repair the scores of all extracted tuples.

Recall from Section 2 that we assume a black-box scoring function for tuples and patterns. One option for solving R5 would be to repeatedly solve R2: determine the modified scores for the set of tuples directly contributed to by p, then modify scores of patterns based on the modified tuples, and so on. However, in the presence of cycles in the EBG, the above procedure may result in a large number of iterations (even infinite if the scoring function doesn't converge to a fixed point). An alternative approach, facilitated by EBG, is to effectively redo the scoring in an iterative fashion, by applying R2 to *snapshots* of EBG at the end of every iteration.

DEFINITION 4.1. *Given an EBG $G = (P, T, E_1, E_2)$ resulting from I iterations of IIE, the* snapshot *of G at iteration $1 \le i \le I$ denoted $G|_i$ is the EBG at the end of iteration i of IIE.* □

Require: $G = (P, T, E_1, E_2)$, pattern p, iteration i, score $s \to s'$
1: $G_s = G|_i$
2: $score(p) = s'$
3: $apply(G_s, G)$
4: **for** $j = i..I$ **do**
5: $\quad G_s = G|_i$
6: $\quad G_s = S(G_s)$
7: $\quad apply(G_s, G)$
8: **end for**

Algorithm 5: Algorithm for R5: Repair scores of an IIE result when the score of pattern p at iteration i is modified from s to s'

We can compute the snapshot of G in linear time when we maintain iteration numbers on each edge during IIE. To solve R5, we start with the iteration in which p's score is modified. We successively (1) revise scores for each snapshot of G, (2) apply the modified scores to the entire EBG, (3) proceed to the next snapshot. Algorithm 5 provides the pseudo code for solving R5, assuming a function S that executes the black-box function for computing scores in an EBG, and a function *apply* that copies modified scores from a snapshot to an entire EBG.

THEOREM 4.3. *Given an EBG $G = (P, T, E_1, E_2)$, pattern p whose score at iteration i is repaired from s to s', Algorithm 5 solves R5 in $O((I - i)K|G|)$, where I is the total number of iterations of IIE, K is the time taken for one call of the black-box scoring function.*

5. EXPERIMENTAL EVALUATION

We now present our experimental evaluation of our proposed techniques. We begin by describing our data collection methods (Section 5.1). Then, we discuss our experiments on examining the utility of I4E—our interactive investigation approach—, for a diverse set of relations (Section 5.2). We then experimentally evaluate the space and time overhead introduced by our EBG-based framework (Section 5.3). As we will see, our approach provides significant "return on investment": we are able to effectively utilize I4E's algorithms to explain and diagnose IIE results, and fix them through minimal interaction, while introducing acceptable overheads. We present results on the related issue of trading off overhead and completeness of an EBG representation (Section 5.4).

5.1 Experimental Settings

Data sources: We used a collection of 500 million web pages crawled by Yahoo! search engine crawl.

Iterative information extraction method: For our IIE, we reimplemented a state-of-the-art bootstrapping exraction technique described by Pasca et al. [29] for large-scale datasets such as Web corpora. Other related IIE systems such as Snowball [2], Espresso [30] follow a similar paradigm varying in their scoring methods.

Extracted relations: As extraction tasks, we focus on six relations:

1	`actors`:	⟨movie, actor⟩
2	`books`:	⟨book, author⟩
3	`directors`:	⟨movie, directors⟩
4	`mayor`:	⟨U.S. city, mayor⟩
5	`sen-party`:	⟨senator, affiliated party⟩
6	`sen-state`:	⟨senator, state⟩

For each relation, we run our IIE methods for 10 iterations. Table 2 shows the number of tuples generated for each relation. We also studied the distribution of $|T_g(p)|$, i.e., the number of tuples that generate a pattern, and $|P_g(t)|$ the number of patterns that generate a tuple. Figure 3 shows these distributions for `actors`, and Figure 4 shows these distributions for `books`. (We omit graphs for

domain	size	domain	size
actors	14,414	mayor	28,514
books	142,337	sen-party	2,119
directors	230,766	sen-state	14,582

Table 2: Size of the relations in our experiments.

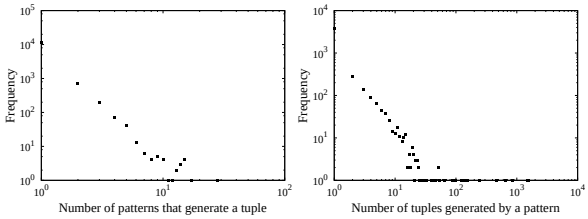

Figure 3: Actors relation: (a) Number of patterns generating a tuple (b) Number of tuples generated by a pattern

other domains due to space restrictions, but the trends are similar.) As seen in Figures 3 and 4, a large proportion of the patterns are generated from a few tuples; similarly a large proportion of tuples are generated using a few patterns. Comparing Figures 3 and 4 with the data in Table 2, we also confirm our hypothesis from Section 3.2 that the number of extraction patterns learned by an IIE are relatively small compared to the number of generated tuples.

5.2 Effectiveness of I4E Algorithms

To examine the utility of the proposed I4E algorithms, we recruited a human annotator to prototype a repair scenario. Based on the IIE output, for a relation, we carried out two experiments: (1) **patterns**-based repair and (2) **tuples**-based repair. For the patterns-based repair, the annotator was shown a pattern and requested to identify whether the pattern is valid for the relation for which it was generated, after being given a brief description of components like information extraction, patterns, and tuples. For instance, for `actors`, users maybe asked: *Is the pattern, '<Movie>-based films starring <Actor> going to generate only valid tuples for our* `actors` *relation?'* The annotation response was recorded to be either 'correct' or 'wrong.' Analogously, for the tuple-based repair, the annotator was shown a tuple and requested to identify whether the tuple is a valid instance of the relation. It is noteworthy that since the number of patterns is relatively smaller than the number of tuples used, an investigation scenario in practice may begin with a pattern-based repair. As we will see, our proposed approach rapidly repairs tuples after annotating only a handful of patterns.

For comparison, we developed three methods to pick the next pattern to show to the annotator. The first method, `P-Inf`, computes the influence for each pattern (see Section 3) and presents them in decreasing order of influence. The second method, `P-Scr`, orders the patterns in decreasing order of confidence order assigned by the extractor. The third method, `P-Rnd`, randomly picks the next unseen pattern; we simulate the result of `P-Rnd` as an average over all possible orderings.

To evaluate the benefit of seeking human feedback on a set of patterns, we use a "low-level" metric, namely, *the total number of repaired (or resolved) tuples* in the output. Our evaluation methodology is as follows: Annotators were requested to label each pattern as *correct* or *wrong*, and we note the number of repaired tuples depending on the annotation. Suppose a pattern p was confirmed to be correct by a user, all tuples in the set $T_p(p)$ of tuples produced by p are resolved to **true**, and can be thus, considered repaired. On the other hand, if p is marked as wrong, each tuple $t \in T_p(p)$ may or may not be resolved, since a tuple t may be produced by other potentially correct patterns. A tuple t is resolved to **false** if and only if all patterns in the set $P_g(t)$ of patterns that generated t have been annotated as wrong by the user.

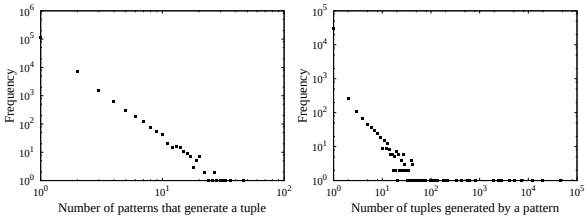

Figure 4: Books relation: (a) Number of patterns generating a tuple (b) Number of tuples generated by a pattern

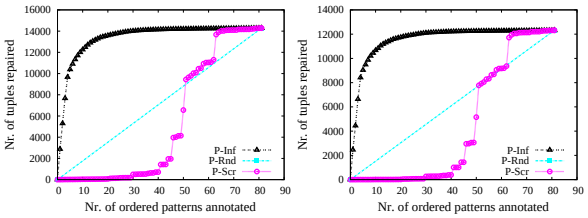

Figure 5: Gains when annotated pattern is (a) correct and (b) wrong for the actors relation.

Given a batch B of annotated patterns for which human feedback was received, we examine the total number of repaired tuples for cases when patterns were labeled correct as well as the total number of repaired tuples when patterns were labeled wrong. Note that applying I4E naturally does not require human feedback to be processed separately; this step is performed solely for our experimental evaluation in order to understand in-depth each of the two important scenarios.

For the actors relation, Figure 5 shows the number of repaired tuples for varying number of patterns annotated, when patterns were marked correct (Figure 5(a)) and when patterns were marked wrong (Figure 5(b)). From the figures, we observe that ordering patterns by their influence increases the number of repaired tuples substantially faster than that using a naive approach of random ordering, or even using confidence scores to order pattern. In particular, after annotating only a few (about 5 to 10) tuples, `P-Inf` resolves the status of about 75% of the tuples in the output. As an interesting observation, based on the performance of `P-Scr`, we observe that the highest scoring extraction pattern may not be the most influential pattern. In our experiments, the most influential pattern, i.e., the first pattern fetched using `P-Inf` is '⟨m⟩ film starring ⟨a⟩', which generated 2415 tuples in the output. The highest scoring pattern, i.e., the first pattern fetched using `P-Scr` is 'movie casino royale, starring', which generated 3 tuples. We observe a similar trend for other relations. Specifically, Figures 6, 7, 8, 9, and 10, respectively, compares the performance of these methods for `books`, `directors`, `sen-party`, `sen-state`, and `mayor`. One interesting observation from Figures 5–10 is that for each relation, the shape of the correct and wrong graphs are similar; e.g., the `P-Scr` curves in Figure 8(a) and 8(b) are similar. This is because, although the absolute values of the gains depend on whether the pattern is correct or wrong, the overall shape of the curve is determined by its steps corresponding to the most influential patterns, which appear at the same point in the pattern ordering. Next we discuss a few other interesting observations for the different domains.

In general, the patterns picked using `P-Scr` prove to be specific and are associated with a relatively small set of (correct) tuples, and thus the gain from annotating such patterns is small. For instance,

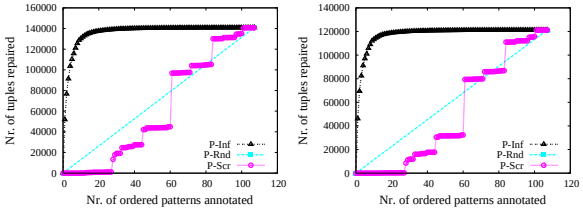

Figure 6: Gains when annotated pattern is (a) correct and (b) wrong for the books relation.

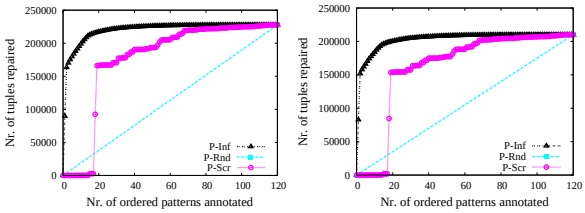

Figure 7: Gains when annotated pattern is (a) correct and (b) wrong for the `directors` relation.

for `sen-party`, the top-2 patterns generated using `P-Scr` are, 'presidential candidates u.s. senator' and 'presidential bid of sen.'; in contrast, the top-2 patterns generated using `P-Inf` are, 'u.s. senator' and 'senator and presidential candidate.' Interestingly, for some relations such as, `directors` (see Figures 7), we may have `P-Scr` perform similar to `P-Inf`: after annotating 20 patterns, the performance for `P-Scr` is close to `P-Inf`, although the number of repaired tuples are higher for `P-Inf`. To gain intuition into this, we observed that the first pattern picked for annotation using `P-Scr` is, 'has a new director' (influence = 2), and that using `P-Inf` is, 'directed by' (influence = 89745). At position 18, `P-Scr` picks the latter pattern and therefore, rapidly resolves a large number of tuples. Overall, we observed that for almost all relations `P-Scr` initially picks patterns that are reliable but specific to the relation and the gains from using `P-Scr` increase substantially (as shown by a step in all graphs) only when an influential high-scoring pattern is selected.

For the actors relation, Figure 11 shows results from tuple-based repair where annotators were shown top-100 tuples using two different methods, namely, `T-Inf` and `T-Scr`, which order tuples by their influence and confidence score respectively. (Tuple-based repair graphs for other relations are similar, and omitted due to space constraints.) When a tuple is annotated wrong, all patterns associated with it are repaired to **false**. However, for a pattern to be considered repaired to **true**, all the tuples associated with it have to be annotated correct. Therefore, when tuples are annotated correct, very few patterns are repaired. We observed that using `T-Scr`, we got tuples that shared patterns and therefore, `T-Scr` repairs slightly more patterns than `T-Inf`. However, more patterns are repaired when tuples are annotated wrong, and `T-Inf` repairs around 25% more patterns than `T-Scr`. A key observation from Figures 5–11 is that for a fixed number of annotations, we can quickly repair relatively larger number of tuples by using pattern-based repair than the number of patterns repaired using the tuple-based repair.

5.3 Overhead of I4E

As discussed in Sections 2 and 3, I4E algorithms rely on building an EBG graph for each iteration. In this section, we examine the overhead in space and time incurred by I4E over a conventional IIE system (without investigation capabilities).

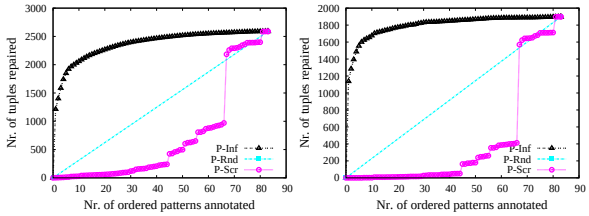

Figure 8: Gains when annotated pattern is (a) correct and (b) wrong for the `sen-party` relation.

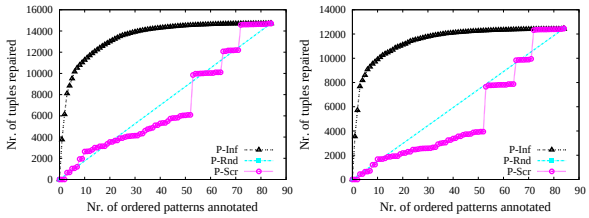

Figure 9: Gains when annotated pattern is (a) correct and (b) wrong for the `sen-state` relation.

5.3.1 *Space Overhead*

We begin by identifying two cases involving a standard, unmodified IIE. In the first case, *score recomputation*, the IIE assumes that at any iteration the tuples generated (including seed tuples as well as the newly identified tuples) may need to be reevaluated. For instance, this may be the case when the minimum threshold of confidence scores applied to the tuples changes at each iteration. Therefore, IIE *will* need to store information similar to that maintained in an EBG, e.g., the list of tuples generated by each pattern as well as list of tuples produced by each pattern. Under this scenario, the only overhead incurred to enable I4E algorithms is during the final iteration. An unmodified IIE may chose to not materialize this information only in the final iteration. Table 3 shows the relative space overhead incurred by I4E algorithms for this scenario, for varying total number of iterations. We measure the relative overhead as $\frac{s_n - s_o}{s_o} \cdot 100$, where s_o is the space requirements for the IIE system and s_n is the space requirements for an I4E enabled IIE system. We observe that for a space overhead less than 15%, an IIE system can support I4E algorithms, furthermore this overhead "amortizes" across iterations and the overhead can be as low as 5% when 15 iterations are run.

domain	**iterations**		
	5	10	15
`actors`	14.1	6.67	4.31
`books`	13.22	6.66	4.10
`directors`	13.00	6.21	4.04
`mayor`	13.13	6.23	4.13
`sen-party`	15.31	7.21	4.71
`sen-state`	14.23	6.70	4.40

Table 3: Relative increase (%) in space introduced by EBG for various relations and iterations for score recomputation.

The second case involving an unmodified IIE is *score no-recomputation*, where IIE computes scores for each tuple in the first iteration it was observed and thus, no tracing information regarding tuples or patterns need to be maintained. Note that the IIE still needs to maintain a list of tuples and patterns generated by each iteration, but the connection between them is not needed. Table 4 shows the relative space overhead incurred by an IIE method that enables I4E algorithms (see column `all`), when 15 iterations are performed. As expected, the overhead in this case is higher than that in the case of

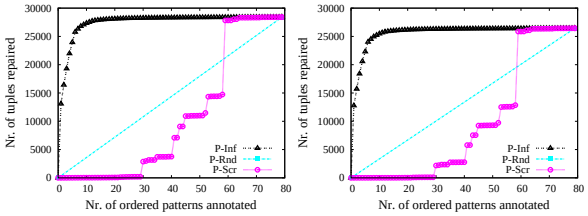

Figure 10: Gains when annotated pattern is (a) correct and (b) wrong for the `mayor` relation.

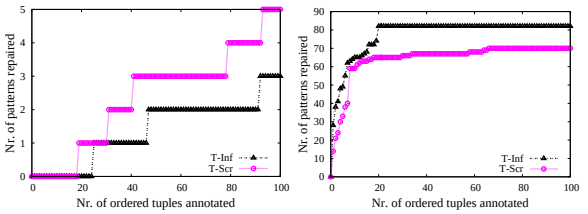

Figure 11: Gains when annotated tuple is (a) correct and (b) wrong for the `actors` relation.

score recomputation. For some relations, we may double the space utilization by enabling I4E algorithms. Intuitively, if each pattern generates two tuples, we need to store twice the amount of information as unmodified IIE. As an optimization, we examined the overhead if we were to prune the EBG based on the influence of patterns. Recall, from Section 5.1 most tuples are generated by a few patterns. Specifically, we only store the edges associated with top-K influential patterns. Table 4 lists these values for K = 5, 10, and 15. For most cases, reducing the number of patterns to follow substantially reduces the space overhead. Naturally, this space reduction comes at the price of "coverage", i.e., eliminating patterns can reduce the coverage of tuples (see Section 5.4).

5.3.2 *Time Overhead*

We examined I4E's time overhead for both the score recomputation and score no-recomputation cases. Table 5 shows the relative time overhead for the score recomputation case varying the number of iterations, and Table 6 shows those for score no-recomputation case when 15 iterations are performed. The relative time overhead for n iterations is measured as $\frac{t_n - t_o}{t_o}$, where t_o is the time to complete i iterations by an unmodified IIE and t_n is the time to complete i iterations by I4E enabled IIE. Analogous to the space overhead, the time overhead for score recomputation is always very small, and further decreases as the number of iterations is increased. Even for score no-recomputation, we observe very low for most relations. Further, the time overhead for no-recomputation reduces substantially if we focus on edges associated with top-K influential patterns. For `sen-party`, the high time overhead is due to the small size of the relation, as compared to the relatively higher processing cost involved.

domain	# patterns			
	5	10	15	all
actors	30.2	52.5	63.9	113.2
books	34.3	55.6	61.2	98.4
directors	33.7	46.8	55.7	94.9
mayor	37.3	56.2	59.7	97.1
sen-party	45.2	60.1	69.1	138
sen-state	21.5	41.7	52.7	115.2

Table 4: Relative increase (%) in space introduced by EBG for various relations and # patterns for no score recomputation.

domain	iterations		
	5	10	15
actors	2.39	1.01	0.65
books	2.37	1.28	0.80
directors	7.30	6.51	1.3
mayor	1.71	0.91	0.62
sen-party	12.40	6.22	4.12
sen-state	2.89	1.33	0.86

Table 5: Relative increase (%) in time introduced by EBG for various relations and iterations for score recomputation.

domain	# patterns			
	5	10	15	all
actors	5.61	12.05	17.05	21.27
books	2.75	9.29	13.02	22.66
directors	3.85	4.54	15.9	19.56
mayor	0.37	1.05	12.71	21.31
sen-party	30.1	49.1	61.8	71.2
sen-state	1.23	2.25	16.64	23.32

Table 6: Relative increase (%) in time introduced by EBG for various relations and # patterns for no score recomputation.

5.4 Overhead vs. Coverage Tradeoff

In the previous section, when computing the space and time overhead for score no-recomputation case, we observed that the space overhead can be reduced by storing only top-K influential patterns. However, this naturally comes at the cost of completeness of the EBG representation. For instance, eliminating edges associated with some patterns may leave out tracing information about some tuples. To examine the extent of this incompleteness, we measured the fraction of output tuples that are completely represented for varying number of influential patterns, called *coverage*. Table 7 shows the results. As we can see, with a space overhead of 30% maintaining 5 patterns, we have a coverage in excess of 70% in all relations. When we maintain 15 patterns, the space overhead incurred is 50–65%, but coverage increases to ∼85–95%.

5.5 Evaluation Conclusion

In summary, we established the utility of our investigation approach over a variety of relations. By using influence measures, I4E effectively guides users to identify patterns as well as tuples that can aid the most in a repair process. Furthermore, we extensively studied the overhead in space and time when using EBG, and observed that I4E introduces an acceptable overhead. Finally, we studied the tradeoff between representation completeness of I4E and the overhead introduced by it.

6. RELATED WORK

Information extraction has received significant attention in the recent years (see [32, 15, 2, 27, 28] and references therein). Research efforts have focused on improving the extraction accuracy [32, 15, 2, 27, 28] or managing extraction uncertainty using probabilistic database [18, 6] or handling dynamic extraction scenarios [9].

To allow users of IE to handle the uncertainty of the extraction output, earlier work [23] has shown how to build optimizers for extraction tasks for a user-specified quality requirement [24, 25]. Along this direction, [26] presented ranking algorithms to fetch a few good tuples from the extraction output as specified by the users. Our paper introduces a novel problem of interactively investigating output of an information extraction (IE) and allowing users to trace, diagnose, and repair any unexpected output.

Close to our work is the study of *provenance* (or *lineage*) in databases: at a high-level, provenance has a similar goal, of providing transparency in query answers over a database. There is a

domain	top-5		top-15		all patterns	
	overhead	coverage	overhead	coverage	overhead	coverage
actors	30.2	72.7%	63.9	92.2%	113.2	100%
books	34.3	78.3%	61.2	96.3%	98.4	100%
directors	33.7	79.0%	55.7	93.5%	94.9	100%
sen-party	45.2	71.4%	69.1	84.4%	138	100%
sen-state	21.5	77.7%	52.7	83.2%	115.2	100%

Table 7: Tradeoff between (1) correct-influence *coverage* and (2) space overhead, for top-K patterns

large body of previous work on provenance including but not limited to [3, 5, 8, 10, 12, 17, 31, 33, 34, 1]; the previous work on provenance spans various contexts such as data warehouses, probabilistic databases, and scientific workflows. We refer the reader to [33, 22] for surveys on provenance. The most relevant previous work on provenance is that of [21], which addresses the problem of deriving the provenance (explanations) for *non-answers* in extracted data. The paper considers conjunctive queries, and for every potential tuple t in an answer to a conjunctive query, the authors provide techniques for determining updates to base data that would produce t in the output. We focus on IIE results, which cannot be captured by conjunctive queries; moreover, our goal is to provide explanations for extracted tuples, and subsequently guiding the process of repairing the extraction system.

7. CONCLUSIONS AND FUTURE WORK

This paper presented I4E, a system for users and developers to interactively carry out post-extraction investigation. We formalize three fundamental phases of investigation: *explaining* the extraction result, *diagnosing* potentially erroneous components, and *repairing* the extraction result by fixing these components. We showed a simple data structure, EBG, that stores necessary information during extraction to support these phases. We presented a suite of algorithms to efficiently answer investigation questions for each of the three phases. While most questions allowed efficient algorithms, some questions (such as picking the K most influential patterns) were provably NP-hard. We provided efficient approximate solutions for each of the intractable questions. We then described techniques to perform more complex investigations by chaining the individual operations of explain, diagnose, and repair. We demonstrated the effectiveness of I4E through a detailed experimental evaluation over six real-world datasets obtained from a Web corpus of 500 million documents. We showed that I4E algorithms help in identifying and fixing an extraction system with minimal human feedback, which introducing little space or time overhead.

While I4E laid the foundation for introducing transparency and subsequent improvement of information extraction systems, several interesting challenges remain open. First, in this paper we focused on iterative information extraction systems, and extending our approach to other (non-iterative) extraction systems is an important next step. Second, while the primary goal of our approach is to perform post-extraction investigation, an interesting by-product of our work is the process of extraction can be optimized. For instance, we may decide to retain a tuple in the pruning stage even if it does not meet the threshold, since at a later stage the tuple's score may be increased due to the discovery of new patterns. Fully exploring how our EBG facilitates such extraction-specific optimization is an interesting research direction. Third, incorporating textual context as a first-class component of I4E and further developing the theory of chaining are specific extensions to our work. Finally, applying graph-compression techniques on EBG is an orthogonal aspect that can compliment the investigation performance.

8. REFERENCES

[1] Open Provenance Model. *http://twiki.ipaw.info/bin/view/Challenge/OPM*, 2009.

[2] E. Agichtein and L. Gravano. Snowball: Extracting relations from large plain-text collections. In *DL*, 2000.

[3] O. Benjelloun, A. Das Sarma, A. Halevy, and J. Widom. ULDBs: Databases with uncertainty and lineage. In *Proc. of VLDB*, 2006.

[4] S. Brin. Extracting patterns and relations from the world wide web. In *WebDB*, 1998.

[5] P. Buneman, A. Chapman, and J. Cheney. Provenance management in curated databases. In *Proc. of ACM SIGMOD*, 2006.

[6] M. J. Cafarella, C. Re, D. Suciu, O. Etzioni, and M. Banko. Structured querying of web text: A technical challenge. In *Proceedings of CIDR-07*, 2007.

[7] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *IAAI*, 1999.

[8] A. Chapman and H. V. Jagadish. Issues in building practical provenance systems. *IEEE Data Engineering Bulletin*, 2007.

[9] F. Chen, A. Doan, J. Yang, and R. Ramakrishnan. Efficient information extraction over evolving text data. In *ICDE*, 2008.

[10] L. Chiticariu, W. Tan, and G. Vijayvargiya. DBNotes: a post-it system for relational databases based on provenance. In *Proc. of ACM SIGMOD*, 2005.

[11] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press and McGraw-Hill, 2nd edition, 2001.

[12] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *VLDB Journal*, 12(1), 2003.

[13] D. Downey, O. Etzioni, and S. Soderland. A probabilistic model of redundancy in information extraction. In *Proceedings of IJCAI-05*, 2005.

[14] O. Etzioni, M. Cafarella, D. Downey, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[15] O. Etzioni, M. J. Cafarella, D. Downey, S. Kok, A.-M. Popescu, T. Shaked, S. Soderland, D. S. Weld, and A. Yates. Web-scale information extraction in KnowItAll (preliminary results). In *Proceedings of WWW-04*, 2004.

[16] M. R. Garey and D. S. Johnson. Computers and Intractability. *W. H. Freeman and Company*, 1979.

[17] T. J. Green, G. Karvounarakis, and V. Tannen. Provenance semirings. In *Proc. of ACM PODS*, 2007.

[18] R. Gupta and S. Sarawagi. Curating probabilistic databases from information extraction models. In *VLDB*, 2006.

[19] M. A. Hearst. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of COLING-92*. Association for Computational Linguistics, 1992.

[20] D. S. Hochbaum and A. Pathria. Analysis of the greedy approach in problems of maximum k-coverage. *Manuscript*, 1994.

[21] J. Huang, T. Chen, A. Doan, and J. F. Naughton. On the provenance of non-answers to queries over extracted data. *PVLDB*, 1(1), 2008.

[22] R. Ikeda and J. Widom. Data lineage: A survey. Technical report, Stanford University, 2009.

[23] P. G. Ipeirotis, E. Agichtein, P. Jain, and L. Gravano. Towards a query optimizer for text-centric tasks. *ACM Transactions on Database Systems*, 32(4), Dec. 2007.

[24] A. Jain, A. Doan, and L. Gravano. Optimizing SQL queries over text databases. In *ICDE*, 2008.

[25] A. Jain, P. G. Ipeirotis, A. Doan, and L. Gravano. Join optimization of information extraction output: Quality matters! Technical Report CeDER-08-04, New York University, 2008.

[26] A. Jain and D. Srivastava. Exploring a few good tuples from text databases. In *ICDE*, 2009.

[27] I. Mansuri and S. Sarawagi. A system for integrating unstructured data into relational databases. In *ICDE*, 2006.

[28] M. Paşca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. Names and similarities on the web: Fact extraction in the fast lane. In *Proceedings of ACL06*, July 2006.

[29] M. Paşca, D. Lin, J. Bigham, A. Lifchits, and A. Jain. Organizing and searching the world wide web of facts - step one: The one-million fact extraction challenge. In *Proceedings of AAAI-06*, 2006.

[30] P. Pantel and M. Pennacchiotti. Espresso: leveraging generic patterns for automatically harvesting semantic relations. In *Proc. of ACL*, 2006.

[31] C. Re and D. Suciu. Approximate lineage for probabilistic databases. In *Proc. of VLDB*, 2008.

[32] E. Riloff and R. Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of AAAI-99*, 1999.

[33] W.-C. Tan. Provenance in Databases: Past, Current, and Future. *IEEE Data Engineering Bulletin*, 2008.

[34] A. Woodruff and M. Stonebraker. Supporting fine-grained data lineage in a database visualization environment. In *Proc. of ICDE*, pages 91–102, 1997.

[35] R. Yangarber and R. Grishman. NYU: Description of the Proteus/PET system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, 1998.