Felix: Scaling up Global Statistical Information Extraction Using an Operator-based Approach

Feng Niu Ce Zhang Christopher Ré Jude Shavlik
University of Wisconsin-Madison
{leonn, czhang, chrisre, shavlik}@cs.wisc.edu

April 27, 2011

Abstract

To support the next generation of sophisticated information extraction (IE) applications, several researchers have proposed frameworks that integrate SQL-like languages with statistical reasoning. While these frameworks demonstrate impressive quality on small IE tasks, they currently do not scale to enterprise-sized tasks. To enable the next generation of IE, a promising approach is to improve the scalability and performance of such statistical frameworks.

Our technical observation is that many IE subtasks, such as coreference resolution or classification, can be solved by specialized algorithms that achieve both high quality and high performance. In contrast, current general-purpose statistical inference approaches are oblivious to these subtasks and so use a single algorithm independent of the subtask that they are performing. We present FELIX, in which programs are expressed in a general statistical inference language (called Markov logic). FELIX first breaks the program into a handful of subtasks, which can then be executed using predefined operators, i.e., statistical algorithms. A key challenge FELIX faces is to decide whether or not to materialize intermediate results from the operators. To attack this challenge, FELIX uses a cost-based approach that relies on the RDBMS optimizer. Using all of our techniques, we show that FELIX efficiently processes global IE programs on large real-world datasets while prior approaches crash or take days. FELIX, in turn, is able to execute programs that achieve higher quality than state-of-the-art IE approaches on three real-world datasets.

1 Introduction

The importance of text data to modern enterpises has led to intense interest in information extraction (IE) from both researchers and industry [7, 8, 16]. There are two major approaches to IE: (1) rule-based IE that builds IE programs from user-defined rules such as regular expressions and (2) learning-based IE that employs machine learning techniques such as CRFs [16]. Both rule-based and learning-based approaches treat IE as a function from small units of unstructured data (e.g., a document or a sentence) to small units of structured data (e.g., relational tuples). The difference between the approaches is in how this function is defined: manually crafted rules in the rule-based approach versus statistical models in the learning-based approach. Still, both approaches are *local.* That is, they map small units to small units. In contrast, a *global* IE approach is able to use information from anywhere in the corpus. The hope is that this global information will improve the quality of IE results.

For example, to extract affiliation relationships from webpages, a local IE system may use dictionaries to identify mentions (i.e., references) of persons and organizations. This process may extract several person-organization pairs from distinct webpages such as {('David', 'UW-Madison'), ('David', 'UWisc'), ('Jeff', 'UWisc')}. A local IE system could not conclude that 'Jeff' is actually affiliated with 'UW-Madison'. In contrast, by looking across pages, a global IE system could deduce that 'UW-Madison' and 'UWisc' refer to the same organization, and then link 'Jeff' with 'UW-Madison'. Applying this idea to DBLife[1], we can boost recall by 160% while maintaining the same precision (Appendix C.2). Such advantage has motivated researchers to start developing global IE systems [6, 34, 39].

In a global IE application, a developer often knows rules that are only likely (but not certain) to be correct. For example, *"if the set of all persons affiliated with one organization (say UWisc) is almost identical to the set of all persons affiliated with another (UW-Madison), then it is likely these organizations are the same."* These less precise rules cannot be expressed by traditional, precise languages like SQL. To support these less precise rules, a number of frameworks have been proposed that blend statistical assertions with traditional expressive languages, such as SQL or first-order logic, e.g., PRMs [12], BLOG [20], MLNs [26], PrDB [31], TUFFY [21]. Such frameworks allow developers to write sophisticated IE programs in a single unified language. In this project, we focus on one of these frameworks, called *Markov logic networks* (MLNs), that we are using as part of a DARPA 5-year grand challenge called *Machine Reading*.

Global frameworks like MLNs have demonstrated high quality on small datasets [24, 33], but performance and scalability remain critical challenges. In fact, while current systems allow one to express global IE programs, performance considerations often limit developers to much smaller units than the whole corpus. Thus, to develop the next generation of sophisticated IE applications, we argue that a promising approach is to improve the efficiency and scalability of such frameworks.[2]

The hypothesis in this work is that we can improve the scalability and performance of general-purpose statistical frameworks for IE applications by exploiting the fact that there are a handful of common subtasks in IE applications. For example, a crucial subtask in IE applications is *coreference resolution* (coref), in which we want to determine if two mentions refer to the same real-world entity, e.g., *"is Bill in one email the same as Bill Williams from accounting?"* Naïvely, coref of N entity mentions seems to require explicit representation of a quadratic search space that includes all $\binom{N}{2}$ pairs of mentions – which is untenable even for N in the tens of thousands. However, unaware that they are solving coref, current general-purpose frameworks do perform this quadratic processing and so cannot scale to large N. On the other hand, the problem of coref has been studied for decades, which has produced specialized algorithms that achieve both high quality and high performance [2, 4, 11]. This motivates the central technical question of this work: *"Can one combine the high quality and performance of specialized algorithms with the ease and flexibility of general-purpose frameworks?"*

Our system, FELIX, gives some evidence that the answer is yes. Our first technical contribution is an architecture that allows us to view these statistical tasks as *operators*. In FELIX, each operator encapsulates an algorithm that consumes and produces relations. The inputs and outputs of the operators are tied together using standard SQL queries. To understand the technical challenges that FELIX raises, we focus on incorporating an operator for coreference resolution. We choose coreference resolution because we have found that coref is a bottleneck when building global IE

[1] `http://dblife.cs.wisc.edu/`

[2] Although these frameworks have applications other than IE, we focus on IE for concreteness.

applications (e.g., in the Machine Reading project). Using a specialized coref operator, Felix executes complex IE programs on the DBLife dataset within several minutes with higher quality than DBLife's current rule-based approach. Additionally, prior global IE systems crash on the same task: we estimate that the MLN reference implementation, Alchemy [10], would require over 1TB of RAM to process the coreference operator perfectly (our prior system Tuffy that uses an RDBMS to process MLNs would require over 1TB of disk space).

The Felix architecture raises an immediate data management challenge. While each Felix operator may be very efficient by itself, the scale of data passed between operators (via SQL queries) can be staggering: the reason is that statistical algorithms may produce huge numbers of combinations (say all pairs of potentially matching person mentions). These large results may be killers for scalability, e.g., the complete input to coreference resolution on an Enron dataset has 1.2×10^{11} tuples. The saving grace is that a downstream operator may only consume a small fraction of the output. Thus, we hope that in some cases we do not need to produce these massive, scalability-killing intermediate results. For example, a popular coref algorithm repeatedly asks *"given a fixed word x, tell me all words that are likely to be coreferent with x."* Moreover, the algorithm only asks for a small fraction of the entire corpus. Thus, it is unnecessary to produce all possible matching pairs. Instead we can produce only those words that are needed on-demand (i.e., compute them lazily). On the Enron email dataset with 100K emails, we show that Felix using a lazy approach finishes within 92 minutes, whereas an eager approach crashed after generating about 1 billion tuples. Lazy processing is not a panacea. For other operators, an eager strategy may be orders of magnitude more efficient: in our previous work [21], we show that eager materialization for generic MLN inference results in orders of magnitude speedup compared to a lazy approach. Thus, choosing the right materialization strategy is crucial for performance.

Felix considers a richer space of possible materialization strategies than simply eager or lazy: it can choose to eagerly materialize one or more subqueries responsible for data movement between operators [25]. To choose from this set of possible strategies, our second contribution shows that we can leverage the cost-estimation facility in the RDBMS coupled with our knowledge about Felix's statistical operators to choose efficient query evaluation strategies in Felix. On the DBLife and Enron datasets, our cost-based approach finds execution plans that achieve two orders of magnitude speedup over eager evaluation and 2-5X speedup compared to lazy evaluation.

Related Work

The idea of global IE is not new; indeed, there is a trend in building global IE systems that fuses imprecise raw extractions, background databases, and domain knowledge [34,39]. However, instead of building specific IE systems, our goal is to support global IE system development by scaling up statistical reasoning frameworks that provide ease and flexibility of a general-purpose language combined with the high quality and efficiency of specialized algorithms.

Felix specializes to MLNs. There are, however, other general-purpose statistical frameworks such as PRMs [12], BLOG [20], Factorie [18], and PrDB [31]. Our hope is that the techniques we develop here could apply equally well to these other general-purpose approaches. We choose MLNs because of our work on the machine reading project, and because they have been successfully applied to a broad range of IE-related applications: natural language processing [28], ontology matching [39], and information extraction [41].

There is work on improving the performance of MLNs with alternative inference algorithms [27]. In contrast, the approach we study here moves away from the monolithic, one-algorithm inference

Schema
`psimHard`(per1, per2)
`psimSoft`(per1, per2)
`osimHard`(org1, org2)
`osimSoft`(org1, org2)
`coloc`(per, org)
`homepage`(per, page)
`omention`(page, org)
`faculty`(org, per)
∗`affil`(per, org)
∗`ocoref`(org1, org2)
∗`pcoref`(per1, per2)

Evidence
`coloc`('Ullman', 'Stanford Univ.')
`coloc`('Jeff Ullman', 'Stanford')
`coloc`('Gray', 'San Jose Lab')
`coloc`('J. Gray', 'IBM San Jose')
`coloc`('David', 'UW-Madison')
`coloc`('David', 'UWisc')
`coloc`('Jeff', 'UWisc')
`faculty`('MIT', 'Chomsky')
`homepage`('Joe', 'Doc201')
`omention`('Doc201', 'IBM')
. . .

weight	rule	
$+\infty$	$\texttt{pcoref}(p, p)$	(F_1)
$+\infty$	$\texttt{pcoref}(p1, p2) => \texttt{pcoref}(p2, p1)$	(F_2)
$+\infty$	$\texttt{pcoref}(x, y), \texttt{pcoref}(y, z) => \texttt{pcoref}(x, z)$	(F_3)
6	$\texttt{psimHard}(p1, p2) => \texttt{pcoref}(p1, p2)$	(F_4)
2	$\texttt{affil}(p1, o), \texttt{affil}(p2, o), \texttt{psimSoft}(p1, p2) => \texttt{pcoref}(p1, p2)$	(F_5)
$+\infty$	$\texttt{faculty}(o, p) => \texttt{affil}(p, o)$	(F_6)
8	$\texttt{homepage}(p, d), \texttt{omention}(d, o) => \texttt{affil}(p, o)$	(F_7)
3	$\texttt{coloc}(p, o1), \texttt{ocoref}(o1, o2) => \texttt{affil}(p, o2)$	(F_8)
4	$\texttt{coloc}(p1, o), \texttt{pcoref}(p1, p2) => \texttt{affil}(p2, o)$	(F_9)
	. . .	

Rules

Figure 1: An example Markov logic program that performs three tasks jointly: 1. discover affiliation relationships between people and organizations (`affil`); 2. resolve coreference among people mentions (`pcoref`); and 3. resolve coreference among organization mentions (`ocoref`). The remaining eight relations are evidence relations. In particular, `coloc` stores raw extraction results of person-organization co-occurrences.

paradigm. Theobald et al. [35] design specialized MaxSAT algorithms that efficiently solve a family of MLN programs. In contrast, we study how to scale MLN inference without limiting its expressive power.

This work builds on our recent work [21] on scaling up (monolithic) MLN inference, where we built a system called TUFFY[3] that achieves orders of magnitude scale-up and speed-up by using an RDBMS (instead of hand-coded nested loop join) for relational query processing and data partitioning. However, the scalability of TUFFY is still inherently limited by the fact that it performs monolithic MLN inference. For example, TUFFY still has quadratic space complexity when the program has coref subtasks. Although also built on an RDBMS, the overall idea and optimization techniques developed in FELIX are orthogonal to those in TUFFY.

Outline

- In Section 2, we describe a simple information extraction program expressed as an MLN and prior art on correlation clustering for coreference resolution.
- In Section 3, we describe our main technical contributions: (1) FELIX's architecture for an operator-based approach, (2) a best-effort algorithm to compile MLNs into FELIX operators, and (3) a cost-based approach to executing FELIX query plans.
- In Section 4, we validate our technical contributions on three datasets: (1) an NFL dataset used in Machine Reading, (2) a webpage corpus from DBLife, and (3) the Enron email datasets. On all three datasets, FELIX achieves significantly higher quality (precision/recall) than state-of-the-art IE approaches, whereas previous systems are not scalable.

In the appendix, we include additional details of our system and further experiments.

[3]`http://research.cs.wisc.edu/hazy/tuffy`

2 Preliminaries

We use a program that extracts affiliations from the Web to illustrate how MLNs encode global IE using rich rules and constraints. We then describe a scalable approach to coref – correlation clustering.

2.1 Markov Logic Networks in Felix

Consider the task of extracting affiliation relationships between people and organizations from Web text. Beginning with raw text, an IE system, such as DBLife [8] or SystemT [7], first attempts to extract all person and organization mentions. Transforming the raw text into clean relations is difficult. For example, a major challenge is that a single real-world entity may be referred to in many different ways. Once we have associated a mention to an entity, we can perform much more sophisticated reasoning, e.g., it is likely that a person is affiliated with only a small number of organizations. Below, we describe how MLNs can be used to perform both relationship discovery and coreference resolution to improve the quality of the raw extractions.[4]

Felix is a middleware system: it takes as input a standard MLN program, performs statistical inference, and outputs its results into one or more relations that are stored in a PostgreSQL database. An MLN program consists of three parts: schema, evidence, and rules. To tell Felix what data will be provided or generated, the user provides a *schema*. Some relations are standard database relations, and we call these relations *evidence*. These relations contain tuples that we assume are correct. In the schema of Figure 1, the first eight relations are evidence relations. For example, we know that 'Ullman' and 'Stanford Univ.' co-occur in some webpage, and that 'Doc201' is the homepage of 'Joe'. Other evidence includes string similarity information. In addition to evidence relations, there are also relations whose content we do not know, but we want the MLN program to predict; they are called *query relations*. In Figure 1, `affil` is a query relation since we want the MLN to predict affiliation relationships between persons and organizations. The other two query relations are `pcoref` and `ocoref`, for person and organization coreference, respectively.

In addition to the schema and evidence, we also provide a set of MLN rules to encode our knowledge about the correlations and constraints over the relations. An MLN rule is a first-order logic formula associated with an extended real value number called a *weight*. Infinite-weighted rules are called hard rules, which means that they must hold in any prediction that the MLN engine makes. In contrast, rules with finite weights are soft rules: a positive weight indicates confidence in the rule's prediction.[5] For example, as transitivity is a necessary condition for coreference, F_3 is a hard rule and must hold in any prediction of `pcoref`. Rules F_8 and F_9 use person-organization co-occurrences (`coloc`) together with coreference (`pcoref` and `ocoref`) to deduce affiliation relationships (`affil`). These rules are soft because co-occurrence in a webpage does not necessarily imply affiliation. Intuitively, when a soft rule is violated, we pay a *cost* equal to its weight (described below). For example, if `coloc`('Ullman', 'Stanford Univ.') and `pcoref`('Ullman', 'Jeff Ullman'), but not `affil`('Jeff Ullman', 'Stanford Univ.'), then we pay a cost of 4 because of F_9. An MLN inference algorithm attempts to find the prediction that minimizes this cost.

[4]Felix also supports transforming text to relations, say via conditional random fields, but we do not discuss it here.

[5]Roughly these weights correspond to the log odds of the probability that the statement is true. (The log odds of p is $\log \frac{p}{1-p}$) In general, these weights do not have a simple probabilistic interpretation [26].

Similarly, affiliation relationships can be used to deduce non-obvious coreferences. For instance, using the fact that 'David' is affiliated with both 'UW-Madison' and 'UWisc', Felix may infer that 'UW-Madison' and 'UWisc' refer to the same organization (rules on `ocoref` are omitted from Figure 1). If Felix knows that 'Jeff' co-occurs with 'UWisc', then it is able to conclude that 'Jeff' is affiliated with 'UW-Madison'.

Semantics An MLN program defines a probability distribution over possible worlds. Formally, we first fix a schema σ (as in Figure 1) and a domain D. Given as input a set of formula $\bar{F} = F_1, \ldots, F_N$ with weights $w_1, \ldots, w_N$, they define a probability distribution over *possible worlds* (deterministic databases), as follows. Given a formula F_k with free variables $\bar{x} = (x_1, \cdots, x_m)$, then for each $\bar{d} \in D^m$, we create a new formula $g_{\bar{d}}$ called a *ground formula* where $g_{\bar{d}}$ denotes the result of substituting each variable x_i of F_k with d_i. We assign the weight w_k to $g_{\bar{d}}$. Denote by $G = (\bar{g}, w)$ the set of all ground formulae of $\bar{F}$ and a function w that maps each ground formula to its assigned weight. Fix an MLN $\bar{F}$, then for any possible world (instance) I we say a ground formula g is *violated* if $w(g) > 0$ and g is false in I, or if $w(g) < 0$ and g is true in I. We denote the set of ground formulae violated in a world I as $V(I)$. The cost of the world I is

$$\text{cost}_{mln}(I) = \sum_{g \in V(I)} |w(g)| \tag{1}$$

Through cost_{mln}, an MLN defines a probability distribution over all instances as:

$$\Pr[I] = Z^{-1} \exp\{-\text{cost}_{mln}(I)\}$$

where Z is a normalizing constant.

Inference There are two main types of inference with MLNs: *MAP (maximum a posteriori) inference*, where we want to find a most likely world, i.e., a world with the lowest cost, and *marginal inference*, where we want to compute the marginal probability of each unknown tuples. In our previous work [21], we use an RDBMS to build a scalable MLN inference engine, Tuffy, that supports both MAP and marginal inference. Tuffy is an operator in Felix, and so Felix can perform both types of inference.

2.2 Logistic Regression and CRF in MLNs

Markov Logic Networks are based on exponential models (also called log-linear models) that are the formal semantics behind almost all graphical models [9, 37]. As a result, MLNs can express many special cases of log-linear models, including logistic regression and classification [15] and conditional random fields [16]. We demonstrate with examples.

Logistic Regression A common subtask in IE is document classification; e.g., we may want to select from a set of webpages those that contain biography information. A program may use the following rules for this subtask:

$$\begin{array}{rcl}
10 & \texttt{hasTitle}(d,t), \texttt{hasWord}(t, \text{`Biography'}) & \rightarrow \texttt{isBio}(d). \\
3 & \texttt{hasTitle}(d,t), \texttt{hasWord}(t, \text{`YouTube'}) & \rightarrow \neg\texttt{isBio}(d). \\
2 & \texttt{hasURL}(d,u), \texttt{hasWord}(u, \text{`bio'}) & \rightarrow \texttt{isBio}(d).
\end{array}$$

where the numbers on the left are weights, `has*` are evidence, and `isBio` is the classification relation.

These rules represent a logistic regression model (we present a rigorous proof in Appendix A.1). Thus, we could compute the exact probability of $\texttt{isBio}(d)$ for each document d using simple SQL aggregates. On the other hand, unaware of this subtask, a monolithic MLN system would run sample-based inference algorithms that produce only approximate answers.

Conditional Random Field As another special case of exponential models, a linear-chain conditional random field (CRF) model [16] can in fact be expressed in MLN directly. For example, suppose we use CRF to label sequences of words with features represented by evidence relation $\texttt{F}(seq, pos, val)$ and labels represented by query relation $\texttt{T}(seq, pos, tag)$. Then the following MLN rules encode the CRF model:

$$\begin{array}{rll} \infty & \texttt{T}(s,p,t1), \texttt{T}(s,p,t2) & \to t1 = t2. \\ w_{v,t1,t2} & \texttt{F}(s,p,v), \texttt{T}(s,p-1,t1) & \to \texttt{T}(s,p,t2). \end{array}$$

where the weight $w_{v,t1,t2}$ is a function of the current feature value v and the labels $t1, t2$ at position $p-1$ and p in sequence s, respectively. There are efficient dynamic programming-based algorithms that can solve both MAP and marginal inference of CRFs [16]. Felix implements these algorithms.

2.3 Correlation Clustering

Correlation clustering is a formalism of clustering for which there are efficient algorithms that have been shown to scale to (and to achieve high quality on) instances of the coref problem with millions of mentions [2, 3, 13]. Formally, correlation clustering treats the coreference problem as a graph partitioning problem. The input is a weighted undirected graph $G = (V, f)$ where V is the set of mentions with weight function $f : V^2 \mapsto \mathbb{R}$. Intuitively, the goal is to group together those mentions that refer to the same real-world entity. Formally, the goal is to find a partition $\mathcal{C} = \{C_i\}$ of V that minimizes the *disagreement cost*:

$$\text{cost}_{cc}(\mathcal{C}) = \sum_{C_i \in \mathcal{C}} \sum_{\substack{u \neq v \in C_i \\ f(u,v) < 0}} |f(u,v)| + \sum_{C_i \neq C_j \in \mathcal{C}} \sum_{\substack{u \in C_i, v \in C_j \\ f(u,v) > 0}} |f(u,v)|. \tag{2}$$

The intuition behind Eq. 2 is similar to Eq. 1: for any pair of mentions (x, y) s.t. $f(x, y) > 0$ (resp. $f(x, y) < 0$), an algorithm that does not put them in the same cluster (resp. different clusters) must pay a penalty. The goal of any algorithm is to minimize the sum of such penalties.

Arasu et al. [2] take correlation clustering as their underlying formal semantic for a rule-based language to solve what they call *deduplication* and is essentially our coref problem. We show in Appendix A.2 that the possible world that minimizes cost for a correlation clustering problem can be identified with the most likely world in a particular MLN.

A remarkable fact about correlation clustering (and our reason for choosing it) is that simple algorithms for correlation clustering have strong approximation guarantees [1]. Algorithm 1 is a simplified algorithm. Intuitively, the simplicity of the algorithm allows it to scale to large datasets.

Algorithm 1 A Correlation Clustering Algorithm [1]

```
Input: $G = (V, f)$, $f : V^2 \mapsto \mathbb{R}$
Output: $\mathcal{C}$: a partition of $V$
1: $\mathcal{C} \leftarrow \emptyset$
2: while $V \neq \emptyset$ do
3:     $v \leftarrow$ a uniformly random node in $V$
4:     $C_v \leftarrow \{v\} \cup \{u | f(u, v) > 0\}$ // get neighbors of $v$
5:     $\mathcal{C} \leftarrow \mathcal{C} \cup \{C_v\}$; $V \leftarrow V \setminus C_v$ // split as a new cluster
6: return $\mathcal{C}$
```

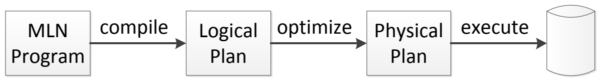

Figure 2: Execution Pipeline of Felix

3 The Felix System

We describe the architecture and technical contributions of the FELIX system: (1) an operator-based approach to statistical inference in MLNs and (2) a cost model for choosing efficient data movement strategies in FELIX.

3.1 Felix Architecture

We describe the FELIX architecture that executes an MLN program using a pre-defined handful of operators (see Figure 1). To execute an MLN program, a program is transformed by FELIX in several phases as illustrated in Figure 2. Each of the phases essentially mirrors the textbook phases in an RDBMS: FELIX first *compiles* a query (here an MLN program) into a *logical plan of statistical operators*. Then, FELIX performs *optimization* (code selection). A key optimization here is to pick whether or not to materialize different portions of the operators. The output of code selection is a sequence of statements that are then executed (by the *executor*). In turn, the executor may call an RDBMS (PostgreSQL in our case) or special purpose operator code. We describe each of these phases below. First, we describe statistical operators.

Statistical Operators are a key concept in FELIX that each encapsulate a statistical task, e.g., coreference resolution, classification, or running generic MLN inference (Table 1). The algorithmic details of a statistical operator are encapsulated inside the operator, but the statistical operator does expose its data access patterns via *adorned views* (described below).

Operator	**Description**
Coref	Coreference resolution
Logistic regression	Classification
CRF	Sequential Labeling
TUFFY	Generic MLN inference

Table 1: FELIX Operators

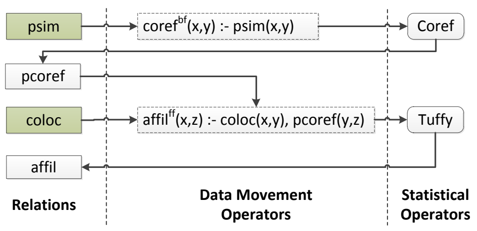

Figure 3: An example operator graph. Relations in shaded boxes are evidence relations.

More precisely, a *statistical operator* takes as input one or more relations and produces as output one or more relations (denoted $o_1, \dots, o_l$). Formally, the input relations are specified by Datalog-like queries $Q_1(\bar{x}_1), \dots, Q_N(\bar{x}_N)$. The body of a query Q_i may refer either to evidence relations, which are standard database tables, or to the output of other statistical operators. The queries are also *adorned*: Each variable in the head of a query is associated with a binding-type, which is either b (bound) or f (free). Given a query Q_i, denote by $\bar{x}^{\mathsf{b}}$ (resp. $\bar{x}^{\mathsf{f}}$) the set of bound (resp. free) variables in its head. Then we can view Q_i as a function mapping an assignment to $\bar{x}^{\mathsf{b}}$ (i.e., a tuple) to a set of assignments to $\bar{x}^{\mathsf{f}}$ (i.e., a relation). To concisely denote binding patterns, we follow Ullman [36]. A query Q of arity $a(Q)$ is written as $Q^\alpha(\bar{x})$ where $\alpha \in \{\mathsf{b}, \mathsf{f}\}^{a(Q)}$. We refer to each Q_i as a *data movement operator* to avoid overloading the term query. For TUFFY, the binding pattern of its input data movement operators are always all-free; but for other operators, the patterns may vary.

Example 1 A Coref operator may use the following data movement operator:

$$\mathtt{Q}^{\mathsf{bf}}(x, y) \quad \leftarrow \quad \mathtt{affil}(x, o), \mathtt{affil}(y, o).$$

which is adorned as bf. Thus, during execution, the Coref operator will send to this data movement operator requests such as $x =$'Jeff', and expect to receive a set of names $\{y | \mathtt{Q}(\text{'Jeff'}, y)\}$.

Given as input an MLN program Γ, the output of FELIX is an instantiation of the query relations specified in Γ. There are three phases that FELIX goes through to instantiate these relations: (1) compilation, (2) optimization, and (3) execution. We describe them in the order that FELIX performs them.

(1) Compilation takes as input an MLN program, and returns a logical plan that is essentially a graph whose nodes are *relations*, *data movement operators*, and *statistical operators*. A relation may be either an evidence relation or a relation output by a statistical operator. One data movement operator Q^α is created for each input in a statistical operator (thus, the same query may be replicated multiple times); the node representing Q^α has an edge to the node that represents its corresponding statistical operator. There is an edge from each base relation mentioned in the body of Q^α to Q^α. Figure 3 illustrates an example operator graph. For the moment, we assume that there are no cycles in this graph and so the resulting structure is a directed acyclic graph (DAG). This DAG is output by compilation and then consumed by the next phase, *optimization*. Compilation is the subject of Section 3.2.

(2) Optimization takes as input a DAG of operators and produces a DAG of *statements*. Statements are of two forms: (1) a prepared SQL statement or (2) a statement encoding the necessary

information to run a statistical operator; e.g., the number of iterations that a Tuffy operator should run, where it should fetch its data from, etc. A key responsibility of *optimization* is to decide on whether or not to materialize intermediate results of statistical operators (and which portions of these results to materialize). This is the subject of Section 3.3. Optimization may take a single data movement operator Q and produce many statements (e.g., if Felix materializes several subqueries of Q).

(3) Execution takes the DAG of statements produced by the optimization phase and executes them using PostgreSQL (for SQL statements), Tuffy (for MLN statements), or the code for a particular operator. Felix supports two types of parallelization: 1) Operator-level parallelism: If two operators in the DAG are independent to each other, Felix can run them in parallel; and 2) Data-level parallelism: each operator itself may partition the data and run sub-operators in parallel internally. For example, the Tuffy operator supports graph partitioning-based parallelization, and the LR and CRF operators support static analysis-based parallelization (Section 3.2).

3.2 Compilation

We can always compile the entire MLN as a single Tuffy operator (which can process any MLN). However, Felix tries to find as many of the subtasks it has specialized operators for as possible. Compilation in Felix consists of the following steps: 1) relation characterization; 2) program decomposition; and 3) data movement planning. All three steps are based on static analysis of the input MLN.

Relation Characterization For each non-evidence relation P, we analyze the MLN rules containing P to find out what properties P satisfies as it occurs in the MLN rules. Example properties include symmetry, reflexivity, transitivity, non-recursiveness, linear-chain-recursiveness, and having-key-constraints. The same set of properties are also used to characterize each available operator, and so there is a matching relationship between the set of non-evidence relations and the set of operators. For example, in Figure 1, the relation `pcoref` satisfies symmetry, reflexivity, and transitivity; thus we could use the Coref operator (which solves equivalence relations) to perform inference on `pcoref`.

Program Decomposition In this step, we assign each non-evidence relation and related MLN rules to one Felix operator, and build the logic plan over these operators. Note that the mapping between relations and operators are many-to-many (e.g., we *could* assign `pcoref` to either Coref or Tuffy). To address this issue, the current compiler of Felix implements a greedy algorithm that assigns each relation to the first matching operator in a predefined precedence order over all available operators. This precedence prioritizes more specialized operators: Coref $>$ CRF $>$ LR $>$ Tuffy. Similarly, we assign the MLN rules in a similar manner; e.g., if a rule can be used by both a Coref operator and an LR operator, we assign this rule to the Coref operator. Rules that contain relations assigned to different operators introduce dependencies among involved operators. For example, in Figure 1, if we assign `affil` to Tuffy and `pcoref` to Coref, then F_5 makes this Coref operator dependent on the Tuffy operator. We build the logic plan using such dependencies. To ensure that the resulting plan is a DAG, we break some dependencies if adding them would lead to loops. In Appendix C.5, we present experiments with cyclic plans.

Data Movement Planning Each Felix operator has a mini-compiler that transforms its MLN rules into a set of conjunctive queries over input relations. Furthermore, depending on the specific statistical algorithm, those conjunctive queries are adorned with different access patterns. For example, an LR operator (resp. CRF operator) would retrieve each object (resp. sequence) being classified (resp. labeled) in turn, and so the query would be bound on the variable representing the object (resp. sequence). Those queries are represented as adorned views in the logical plan.

3.3 Cost-based Optimization

In Felix, data are passed between statistical operators using the data movement operators. We have observed that a critical bottleneck in Felix's execution is the efficiency of data movement operators. Felix currently optimizes each Q_i individually, i.e., Felix does not consider sharing across data movement operators during optimization.

Recall that RDBMSs can execute queries both eagerly (using standard SQL) and lazily (via prepared statements). We have found that both types of execution are helpful when executing Felix plans. If an operator needs repeated access to the entire result of Q_i, it is often more efficient to let the RDBMS produce all of the results and materialize them in an intermediate relation. On the other hand, some statistical operators may inspect only a small fraction of their search space and so such eager materialization is inefficient. For example, the number of edges examined the Coref operator is roughly linear in the number of nodes, regardless of graph density. Moreover, some statistical operators like Coref operate on a very large search space that is specified by the input (e.g., quadratic in the dataset size). In some cases, this input may be so large that an eager materialization strategy would exhaust the available disk space. For example, on an Enron dataset, materializing the following query would require over 1TB of disk space:

$$\begin{aligned}\texttt{maylink}^{\textsf{bb}}(x,y) \quad \leftarrow \quad & \texttt{mention}(x, name1), \texttt{mention}(y, name2),\\ & \texttt{mayref}(name1, z), \texttt{mayref}(name2, z).\end{aligned}$$

Felix is, however, not confined to fully eager or fully lazy. In Felix, we have found that intermediate points (e.g., materializing a subquery of Q_i) can have dramatic speed improvements. We describe Felix's cost-based optimizer that explores a search space of intermediate results; similar to a System-R-style cost-based RBDMS optimizer, Felix enumerates the plan space and chooses the plan with the lowest (predicted) cost. Almost all cost estimation is done by the underlying RDBMS (here, PostgreSQL). To understand the tradeoffs, we consider an example.

Example 2 The following query is used by the Coref operator to perform person coreference on an Enron dataset:

$$\begin{aligned}\texttt{coref}^{\textsf{bf}}(x,y) \quad \leftarrow \quad & \texttt{mention}(x, d1, name), \texttt{mention}(y, d2, name),\\ & \texttt{wrote}(s, d1), \texttt{wrote}(s, d2).\end{aligned}$$

Eagerly materializing this query would generate around half a billion tuples, only a small portion of which will be accessed by Coref. On the other hand, if we are to evaluate the Coref operator's requests on the fly, we would need to pay substantial overhead for repeatedly executing this complex query. Thus, one may consider the following decomposition:

$$\begin{aligned}\texttt{coref}^{\textsf{bf}}(x,y) &\leftarrow \texttt{q}_1(x,n,s),\texttt{q}_1(y,n,s).\\ \texttt{q}_1(x,n,s) &\leftarrow \texttt{mention}(x,d,n),\texttt{wrote}(s,d).\end{aligned}$$

and materialize q_1 while incrementally evaluating `coref`. This way, we have lower materialization cost (on $\texttt{q}_1$) compared to an eager strategy, and lower incremental evaluation cost (on `coref`) compared to a lazy strategy. Other decompositions are also possible.

FELIX now must decide which decomposition to use to get the best performance. To make this decision, FELIX uses a cost model.

To define our cost model, we introduce some notation. Let $Q(\bar{x}) \leftarrow g_1, g_2, \ldots, g_k$ be a data movement operator. Let $G = \{g_i | 1 \leq i \leq k\}$ be the set of subgoals of Q. Let $P = \{G_1, \ldots, G_m\}$ be a partition of G; i.e., $G_j \subseteq G$, $G_i \cap G_j = \emptyset$ for all $i \neq j$, and $\bigcup G_j \subseteq G$. Intuitively, a partition represents a possible materialization strategy: each element of the partition represents a query (or simply relation) that FELIX is considering materializing. That is, the case of one $G_i = G$ corresponds to a fully eager strategy. The case where all G_i are singleton sets corresponds to a lazy strategy.

More precisely, define $Q_j(\bar{x}_j) \leftarrow G_j$ where $\bar{x}_j$ is the set of variables in G_j shared with $\bar{x}$ or any other G_i for $i \neq j$. Then, let query $Q'(\bar{x}) \leftarrow Q_1, \ldots, Q_m$. With this notation, we write our cost model:

$$\text{ExecCost}(P, t) = t \cdot \text{Inc}_Q(Q') + \sum_{i=1}^{m} \text{Mat}(Q_i)$$

Here, $\text{Mat}(Q_i)$ is the cost of eagerly materializing Q_i, t is the total number of lazy incremental evaluations of Q' performed by the statistical operator, and $\text{Inc}_Q(Q')$ is the average cost of lazy incremental evaluation on Q'.

Cost Estimation Both t and $\text{Inc}_Q(Q')$ depend on the access pattern of the statistical operator relying on Q. To optimize Q in FELIX, we must be able to estimate both parameters. We have developed estimates for t for the TUFFY and Coref operators. For example, in the Coref operator, we observe that t is proportional to the number of clusters in the final graph. On the other hand, $\text{Inc}_Q(Q')$ is the cost of one evaluation of an adorned query, say $Q'^{\textsf{bf}}$. As the subgoals in Q' are not actually materialized in the RDBMS, we cannot directly ask the RDBMS for the incremental cost $\text{Inc}_Q(Q')$.[6] In Appendix B.1, we describe a simple strategy to use PostgreSQL to estimate these costs.

With this cost model, the second half of our optimizer is to enumerate different decompositions. The number of possible decompositions is exponential in the size of the query, but in our applications the rules are small. Thus, we can estimate the cost of each alternative, and we pick the one with lowest ExecCost.

We show that this algorithm allows FELIX to pick efficient eager and lazy strategies for different data movement operators (and avoid catastrophically bad materialization plans). Additionally, we demonstrate that extending the space to consider partial materializations achieves 2-5X speedup on our real-world datasets compared to pure lazy strategies, and orders of magnitude speedup compared to pure eager strategies.

[6]PostgreSQL does not fully support "what-if" queries, although other engines do, e.g., for indexing tuning.

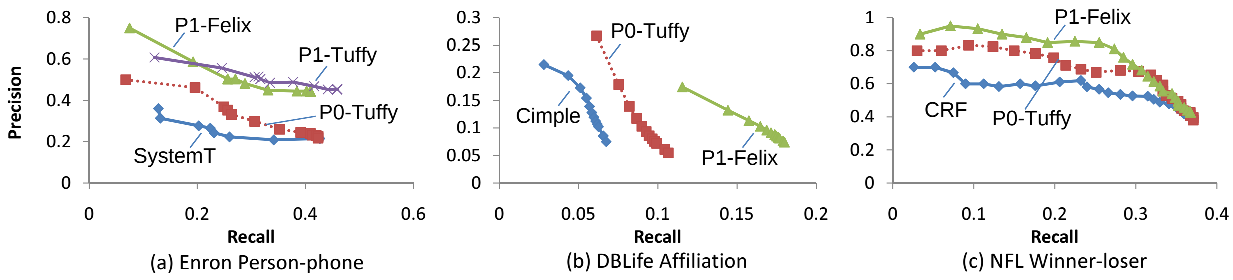

Figure 4: High-level Quality Results of Different IE Systems. Axes are zoomed into interesting regions.

3.4 Execution

Similar to how RDBMS executes a logical plan of relational operators, there are two types of parallelization opportunities when Felix executes its statistical operators: operator parallelism and data parallelism.

Operator Parallelism As the logical plan is represented as a DAG, there may exist operators independent to each other. For example, a logical plan for the MLN in Figure 1 may consists of three operators

$$\{\text{Coref}(\texttt{pcoref}), \text{Coref}(\texttt{ocoref}), \textsc{Tuffy}(\texttt{affil})\}$$

with the Tuffy operator depending on the two Coref operators. Felix will be able to recognize that the two Coref operators are independent to each other, and thus run them in parallel.

Data Parallelism Many statistical operators (e.g., LR and CRF) often operate on small data units independently (e.g., classifying documents or labeling sentences), and therefore can be easily parallelized. Thus, an operator can partition the data and assign the partitions to "sub-operators". To identify such "subsubtasks", we borrow ideas from the Datalog literature [30] that use linear programming to perform static analysis for data partitioning and decide how to split this operator into "sub-operators" each working on a portion of the data. We can run these sub-operators in parallel, and the union of their output is guaranteed to be the same as that of the original operator. As a simple example, note that the variable s (sequence ID) occurs in all literals in the CRF rules in Section 2.2. Felix is able to take advantage of this fact and assign the sequences to different sub-operators that can run in parallel.

4 Experiments

In this section, we validate that global IE programs can offer substantial quality (precision/recall) benefits over local IE programs whenever they can scale. We show that Felix's technical contributions allow global IE programs to scale when they otherwise could not. Then, we validate through a lesion study that the materialization strategies our cost-based optimizer chooses are the key to our higher scalability compared to prior approaches.

Datasets and Competitors Table 2 contains statistics about the three datasets that we select: (1) **DBLife**, where we extract persons, organizations, and affiliation relationships between them from a collection of academic webpages; (2) **Enron**, where we identify person mentions and associated phone numbers in the Enron email dataset; and (3) **NFL**, where we extract football game results from sports news articles. On DBLife, we compare against the rule-based system Cimple that is behind the DBLife web portal. On Enron, we compare against SystemT from IBM.[7] Our rules for the person-phone task are described in two papers on SystemT [17,19]. The NFL dataset is a testbed for an ongoing DARPA Machine Reading project that involves multiple universities and research institutes. We use conditional random fields (CRF), a state-of-the-art local learning-based IE approach, as the baseline on NFL. We also experiment with two state-of-the-art MLN implementations: (1) Alchemy [10], the reference implementation of MLNs; and (2) Tuffy [21], an RDBMS-based implementation of MLNs used in the Machine Reading project.

There are two versions of Enron: Enron-random (**Enron-R**)[8] is a small subset of Enron emails with manually annotated person-phone ground truth; and **Enron**[9] is the full dataset. We use Enron-R for quality assessment, and Enron for performance evaluation.

	Enron-R	**Enron**	**DBLife**	**NFL**
raw data size	1.6MB	700MB	300MB	20MB
#documents	680	225K	22K	1.1K
#mentions	486	2,500K	700K	100K

Table 2: Dataset statistics

Experimental Setup Alchemy is implemented in C++. Tuffy and Felix are both implemented in Java and run on top of PostgreSQL 8.4. Felix uses Tuffy as a library. Unless specified otherwise, all experiments are run on a 2.4GHz Intel Core2 with 4 GB of RAM running RHEL 5. Unless noted otherwise, multithreading is disabled on all systems for fair comparison.

4.1 High-level Quality and Performance

We empirically validate that Felix can efficiently execute complex IE programs and produce results with significantly higher quality than local IE approaches. Furthermore, we show that monolithic MLN systems such as Tuffy or Alchemy do not scale when the program contains challenging subtasks such as coref. To support these claims, we compare the performance and quality of different global IE systems (Tuffy, Alchemy, and Felix) together with state-of-the-art local IE approaches (both rule-based and learning-based) on the three datasets listed above.

On each dataset, we use two MLN programs for global IE – one without coref (denoted P0), and the other with coref (denoted P1) – and run each on Alchemy, Tuffy and Felix. Because Felix and Tuffy behave identically on P0 programs, we only consider P0-Tuffy. On P1 programs, Alchemy and Tuffy runs marginal inference; Felix first executes Coref operators, and then executes its Tuffy operator. We run Alchemy, Tuffy, and Felix for at most 3000 seconds.

To summarize the output quality of each system, we draw precision-recall curves: we take

[7]`http://www.almaden.ibm.com/cs/projects/systemt/`

[8]`http://www.cs.cmu.edu/~einat/datasets.html`

[9]`http://bailando.sims.berkeley.edu/enron_email.html`

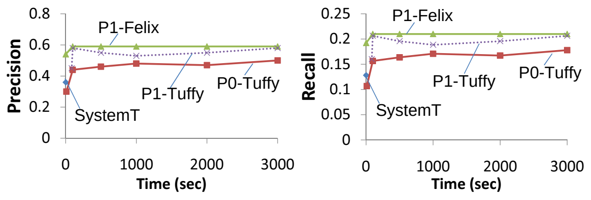

Figure 5: Time-Quality Plot on Enron Person-phone

ranked lists of predictions from each system[10] and measure precision/recall of top-k results while varying the number of answers returned. The quality of each system is shown in Figure 4.

On all three tasks, we see that the quality of global IE systems is superior to local IE approaches. For example, on Enron, P1 systems improved the precision of SystemT by 100% while retaining the same recall. On DBLife, Felix improves the recall by about 200% while maintaining similar precision. On NFL, Felix is able to boost the precision to above 90% from CRF's 60%. This demonstrates an advantage of global IE over local IE. Also, we see that Felix can scale in cases when neither Tuffy nor Alchemy does. On all six programs, Alchemy either crashed after running out of memory or took longer than 3000 seconds, and so there are no curves for Alchemy. The fundamental reason is that Alchemy is a pure-memory implementation and handles relational operations poorly [21]. Tuffy crashed on the P1 programs of both DBLife and NFL after consuming tens of gigabytes of disk space, and so there are no P1-Tuffy curves for DBLife and NFL. The reason Tuffy can run P1 on Enron is that the input data is very small – less than 500 mentions. We conclude that (1) global IE programs offer significant quality improvement, and (2) Felix processes such programs more scalably than prior approaches.

To understand our results more deeply, we describe the experimental methodology on the Enron dataset in this section. Similar methods are used on DBLife and NFL. We defer the detailed descriptions of the methodology of DBLife and NFL to Appendix C.2.

Enron We use the **Enron-R** dataset to evaluate global IE systems together with SystemT. SystemT uses dictionaries for person name extraction, and regular expressions for phone number extraction. To extract person-phone relationships, SystemT uses a fixed window size to identify person-phone co-occurrences, and translate those co-occurrences directly to person-phone relationships. We vary this window size to produce a precision-recall curve of SystemT, as shown in Figure 4(a). We next write a simple MLN program P0 to replace SystemT's relation extraction part (using the same entity extraction results). Instead of fixed window sizes, P0 uses MLN rule weights to encode the strength of co-occurrence and thereby confidence in person-phone relationships. In addition, we write soft constraints such as "*a phone number cannot be associated with too many persons.*" On top of P0, we add in a set of coreference rules to perform person coref, and call this new program P1. We run P0 and P1 on Alchemy, Tuffy and Felix. Alchemy crashed on both programs.

We plot the quality results in Figure 4(a). We see that P1 clearly dominates P0, which in turn dominates SystemT. For example, at the same recall, P1 achieves about twice as high precision as SystemT. This suggests that a global approach to IE can significantly improve extraction quality.

[10]For global IE systems, the ranking is from marginal probabilities. Ranking for local IE systems is described below.

In Figure 5, we plot the precision/recall of top-100 predictions from each system as a function of time. Compared to SystemT, We see that P0-Tuffy quickly converges to achieve about 35% improvement in both precision and recall; P1-Felix achieves close to 65% improvement in both precision and recall. Although Tuffy runs P1 on Enron-R, on all other datasets Tuffy crashes while running P1 (including the full Enron dataset).

4.2 Felix's Cost-based Optimization

We validate that the cost-based materialization tradeoff in Felix produces strategies that outperform both eager and lazy materialization approaches. We focus on the person coreference operator on the DBLife dataset and several Enron datasets and compare the performance of different strategies: 1) **Eager**, where all data movement operators are evaluated eagerly; 2) **Lazy**, where all data movement operators are evaluated lazily; 3) **Opt**, where we decide the materialization strategy for each data movement operator based on the cost model in Section 3.3.

	DBL	**E-5k**	**E-20k**	**E-50k**	**E-100k**
Eager	65 min	27 sec	17.5 min	153 min	>5 hr
Lazy	10 min	44 sec	314 sec	27 min	92 min
Opt	4 min	15 sec	76 sec	4.5 min	30 min

Table 3: Performance comparison of different materialization strategies for person coreference operators on DBlife and Enron. **Eager** on **E-100k** crashed after exhausting disk space.

As shown in Table 3, the performance of the eager materialization strategy suffers as the dataset size increases. The lazy strategy performs much better, and the cost-based approach can further achieve 2-5X speedup. This demonstrates that our cost-based materialization tradeoff is crucial to the efficiency of Felix. In Appendix C, we validate technical details of our optimizer and describe how cyclic Felix plans can further improve quality.

4.3 Subtask Performance Comparison

CRF using Tuffy and Alchemy *Add experiment*

Coref using Tuffy and Alchemy We validate that Felix's Coref operator has both higher performance and higher quality compared to generic MLN inference systems for coref subtasks. To do this, we take a 1000-document subset of the DBLife dataset, and run organization coref with Felix, Tuffy, and Alchemy. Felix runs its Coref operator, while Tuffy and Alchemy performs generic MLN MAP inference. We measure the running time and output quality (precision/recall) of all three systems. (We use the ACM dataset as ground truth for organization entities.)

	Felix	Tuffy	Alchemy
Time	3 sec	3 min	46 min
Precision	0.355	0.201	0.246
Recall	0.043	0.043	0.047

Table 4: Performance and Quality Comparison of Different Coref Solvers

As shown in Table 4, Felix is two orders of magnitude faster than Tuffy, and three orders of magnitude faster than Alchemy. In fact, we found that both Tuffy and Alchemy crash when we increase the dataset size by a factor of 2. Moreover, Felix achieves substantially higher output quality than Tuffy and Alchemy– at roughly the same recall, Felix has around 50% higher precision.

4.4 Parallelization of Felix Operators

Add experiments.

5 Conclusion

While a global approach to information extraction allows higher quality and richer applications than local IE approaches, the frameworks that support global IE do not scale beyond small datasets. We observe that a key bottleneck of current global frameworks is their monolithic approach to statistical inference, and we develop an alternative approach that is operator-based. In this novel approach, different subtasks are handled by specialized algorithms but inside one architecture. We observe that materialization strategies have a large impact on performance, and we propose a cost-based approach. Using these techniques, our system Felix is able to scale to complex IE programs on large datasets and generates results that have higher quality than state-of-the-art local IE approaches.

A Material for Preliminaries

A.1 Logistic Regression in MLNs

In logistic regression, we model a Bernoulli variable y conditioned on $x_1, \cdots, x_k$ using $\Pr(y = 1) = \frac{1}{1+e^{-z}}$, where

$$z = \beta_0 + \beta_1 x_1 + \cdots + \beta_k x_k,$$

and $\beta_i \in (-\infty, +\infty)$ are given parameters. Suppose that $x_i \in \{0, 1\}$, then we can model the same distribution in an MLN program as follows. Let the domain be $D = \{1, \ldots, k\}$. Let $R(z)$ be an evidence relation such that $R(i)$ is true if and only if $x_i = 1$. Let $Q()$ be a zero-arity relation (i.e., a Boolean variable). Consider an MLN Γ that has k rules $R(i) => Q$ each with weight β_i, for $i = 1 \ldots k$. In addition, there is a singleton rule Q with weight β_0. Then there are only two possible worlds modeled by Γ, one with Q being true, and the other with Q being false. First consider rules with positive weight β_i. In the world Q is true, all rules with positive weights are satisfied, and so these rules contribute 0 to the cost. In the world Q is false, only rules with $R(i)$ being true (and thus $x_i = 1$) are violated, and so $\sum_{i:\beta_i>0} \beta_i x_i$ contributes to the cost. For rules with negative weights, we can calculate similar costs. In sum, the cost of the world where Q is true is $\sum_{i:\beta_i<0,x_i=1} |\beta_i| x_i + \sum_{i:\beta_i<0,x_i=0} |\beta_i|(1 - x_i)$, and the cost of the world where Q is false is $\sum_{i:\beta_i<0,x_i=0} |\beta_i|(1 - x_i)$. By definition of MLN semantics, we can show that Q being false has the same distribution as y.

A.2 Correlation Clustering in MLNs

We show that coref subtasks in MLN programs can be formulated as correlation clustering and thus can be handled using simple and efficient algorithms [2].

Given an MLN program Γ, for each non-evidence relation r, we can analyze the program to determine if r satisfies the three properties of equivalence (reflexivity, symmetry, and transitivity) by examining the hard rules in Γ. For example, from formulae $F_1 - F_3$ in Figure 1, we can decide that `pcoref` is an equivalence relation. Suppose there are k equivalence relations $r_1, \ldots, r_k$. Recall that each possible world of Γ can be represented as an assignment to a vector of Boolean variables $\bar{v} = \bar{x}_0 \cup \bar{x}_1 \cup \ldots \cup \bar{x}_k$ where $\bar{x}_i$ $(i > 0)$ corresponds to possible tuples in r_i and $\bar{x}_0$ is the union of all other unknown tuples. Define $\bar{x}_{-i} = \bar{v} \setminus \bar{x}_i$. Then MAP inference of Γ on $\bar{x}_i$ conditioned on $\bar{x}_{-i}$ is essentially a correlation clustering problem. For example, if we fix the state of all the relations except `pcoref`, then the program in Figure 1 becomes a set of rules for correlation clustering of person mentions: F_4 and F_5 generate edges with weights 6 and 2, respectively. In addition, rule F_9 can generate edges with weight -4; to see why, note that F_9 can be rewritten as

$$\texttt{coloc}(p1, o), \neg\ \texttt{affil}(p2, o), \texttt{person}(p2) \texttt{=>} \neg\ \texttt{pcoref}(p1, p2),$$

where `person` is the set of person names. If the same edge is generated by multiple rules, then the weight of this edge is the sum of the weights of the rules. If an edge is generated by none of the rules, then the edge has weight 0. Those weights define a correlation clustering problem in the form of a weighted graph, say G_i.

To understand how to run inference, consider the following scheme. We can decompose MAP inference on Γ into $k+1$ parts, with each part denoted by $\Gamma(\bar{x}_i)$, meaning that we try to find an optimal truth assignment to $\bar{x}_i$ conditioned on an assignment on $\bar{x}_{-i}$, where $0 \leq i \leq k$. We run $\Gamma(\bar{x}_0)$ with standard MAP inference algorithm of MLNs, but for $i > 0$, we run $\Gamma(\bar{x}_i)$ with the efficient correlation clustering algorithm. In a round robin scheduling, the parts are executed in the following order: Initialize a truth assignment to $\bar{v}$ that is consistent with hard rules; for $t = 1 \ldots T$, for $i = 0 \ldots k$, run $\Gamma(\bar{x}_i)$. This is a special form of the Gauss-Seidel scheme [21].

The following proposition shows that the Coref operator solves an equivalent optimization problem as MAP inference in MLNs.

Proposition A.1. *Let $\Gamma(\bar{x}_i)$ be a part of Γ corresponding to a coref subtask; let G_i be the correlation clustering problem transformed from $\Gamma(\bar{x}_i)$ using the above procedure. Then an optimal solution to G_i is also an optimal solution to $\Gamma(\bar{x}_i)$*

We next show that, for a certain family of MLN programs, the Coref operator actually performs approximate MLN inference.

Theorem A.1. *Let $\Gamma(\bar{x}_i)$ be a coref subtask with rules generating a complete graph where each edge has a weight of either $\pm\infty$ or w s.t. $m \leq |w| \leq M$ for some $m, M > 0$. Then the correlation clustering algorithm running on $\Gamma(\bar{x}_i)$ is a $\frac{3M}{m}$-approximation algorithm in terms of the log-likelihood of the output world.*

Proof. In Arasu et al. [2], it was shown that for the case $m = M$, a variant of Algorithm 1 achieves an approximation ratio of 3. If we run the same algorithm, then in expectation the output violates no more than $3\mathbf{OPT}$ edges, where $\mathbf{OPT}$ is the number of violated edges in the optimal partition. Now with weighted edges, the optimal cost is at least $m\mathbf{OPT}$, and the expected cost of the algorithm output is at most $3M\mathbf{OPT}$. Thus, the same algorithm achieves $\frac{3M}{m}$ approximation. □

B Material for Systems

B.1 Cost Estimation For Section 3.3

The cost model in Section 3.3 requires estimation of the individual terms in ExecCost. There are three components: (1) the materialization cost of the eager queries, (2) the cost of a single incremental portion of the query, and (3) the number of times that query will be executed (t). We consider them in turn.

Computing (1), the subquery materialization cost $\text{Mat}(Q_i)$, is straightforward to estimate using PostgreSQL's EXPLAIN feature. As is common for many RDBMSs, the unit of PostgreSQL's query evaluation cost is not time, but instead an internal unit (roughly proportional the cost of 1 I/O). Felix performs all calculations in this unit.

Computing (2), the cost of a single incremental evaluation, is more involved: we do not have Q_i actually materialized (and with indexes built), so we cannot directly measure $\text{Inc}_Q(Q')$. For simplicity, consider a two-way decomposition of Q into Q_1 and Q_2. We consider two cases: (a) when Q_2 is estimated to be larger than PostgreSQL assigned buffer, and (b) when Q_2 is smaller (i.e. can fit in available memory).

To perform this estimation in case (a), Felix makes a simplifying assumption that the Q_i are joined together using index-nested loop join (we build indexes below). Exploring clustering opportunities for Q_i is future work.

Then, we force the RDBMS to estimate the detailed costs of the plan $\mathcal{P} : \sigma_{\bar{x}'=\bar{a}}(Q_1) \bowtie \sigma_{\bar{x}'=\bar{a}}(Q_2)$, where Q_1 and Q_2 are views, $\bar{x}' = \bar{a}$ is an assignment to the bound variables $\bar{x}' \equiv \bar{x}^{\mathsf{b}}$ in $\bar{x}$, and the $\bowtie$ operator is forced to be executed using nested loop join. From the detailed cost estimation, we extract the following quantities: (1) n_i: be the number of tuples from subquery $\sigma_{\bar{x}}(Q_i)$; (2) n: the number of tuples generated by $\mathcal{P}$; and (3) c: the cost (in PostgreSQL's unit) of the $\bowtie$ operator in $\mathcal{P}$. Let p be the number of tuples per disk page. We observed that PostgreSQL's estimation satisfies $c \propto n_1\lceil n_2/p \rceil$ provided that n_1 is not too small. Thus, we can use $\alpha = c/(n_1\lceil n_2/p \rceil)$ to translate between I/O cost and PostgreSQL's cost unit.

Denote by $c' = \text{Inc}_Q(Q')$ the cost (in PostgreSQL unit) of executing $\sigma_{\bar{x}'=\bar{a}}(R_1) \bowtie \sigma_{\bar{x}'=\bar{a}}(R_2)$, where R_i is the materialized table of Q_i with proper indexes built. Without loss of generality, assume $n_1 < n_2$ and that n_1 is small enough so that $\bowtie$ in the above query is executed using nested loop join. On average, for each of the estimated n_1 tuples in $\sigma_{\bar{x}}(R_1)$, there is one index access to R_2, and $\lceil \frac{n}{n_1} \rceil$ tuples in $\sigma_{\bar{x}}(R_2)$ that can be joined; assume each of the $\lceil \frac{n}{n_1} \rceil$ tuples from R_2 requires one disk page I/O. Thus, there are $n_1\lceil \frac{n}{n_1} \rceil$ disk accesses to retrieve the tuples from R_2, and

$$c' = \alpha n_1 \left[\lceil \frac{n}{n_1} \rceil + \log |Q_2| \right] \tag{3}$$

where we use $\log |Q_2|$ as the cost of one index access to R_2 (height of a B-tree). Now both $c' = \text{Inc}_Q(Q')$ and $\text{Mat}(Q_i)$ are in the unit of PostgreSQL cost, we can sum them together, and compare with the estimation on other materialization plans.

In case (b), when Q_2 can fit in memory, we found that the above estimation tends to be too conservative – many accesses to Q_2 are cache hits whereas the model above still counts the accesses into disk I/O. To compensate for this difference, we multiply c' (derived above) with a fudge factor $\beta < 1$. Intuitively, we choose β as the ratio of accessing a main memory relation versus accessing a page on disk. We empirically determine β.

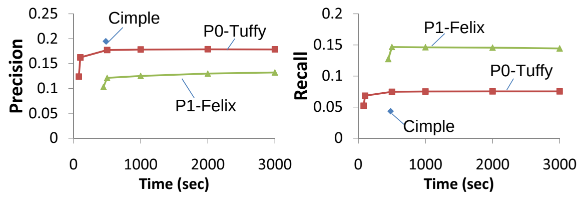

Figure 6: Time-Quality Plot on DBLife Affiliation

Component (3) is the factor t, which is dependent on the statistical operator. However, we can often derive an estimation method from the algorithm inside the operator. For example, our Coref operator implements an algorithm [2] that is similar to Algorithm 1. And so the number of requests to an input data movement operator can be estimated by the total number of mentions (using COUNT) divided by the average node degree (estimated using sampling).

B.2 Cyclic Execution Plans

The operators in a logical plan of Felix may contain cycles. In this case, instead of proceeding in a topological ordering through operators, we need to loop through cycles many times. To pick the operator that is to be executed first, we use a simple heuristic that is based on operator precedence. For example, in the current prototype of Felix, Coref has a higher precedence than Tuffy, and so we always run Coref first if Coref and Tuffy participate in a cycle. If two Coref operators are in a cycle, we choose one arbitrarily. In Appendix C.5, we empirically study the effect of cyclic execution plans on the result quality on the DBLife task.

C Material for Experiments

C.1 Experimental Setup

In all experiments that involve MLN marginal inference, we run 10^6 steps of random walk in each MC-SAT sample. We run all global IE systems for 3000 seconds, and periodically take snapshots of MC-SAT sampling results.

C.2 DBLife and NFL

We describe the detailed methodology in our experiments on the DBLife and the NFL datasets.

DBLife We compare the performance and quality of the global IE systems with Cimple. Cimple identifies person and organization mentions using dictionaries with regular expression variations (e.g., abbreviations, titles). In case of an ambiguous mention such as "J. Smith", Cimple binds it to an arbitrary name in its dictionary that is compatible (e.g., "John Smith"). Cimple then uses a proximity-based formula to translate person-organization co-occurrences into ranked affiliation tuples.

In the P0 program of DBLife, we take Cimple's entity extraction results, encode affiliation discovery using MLN rules, and run marginal inference to obtain a ranked list of affiliation tuples. We next open up the entity extraction phase: we perform part-of-speech tagging [29] on the raw text,

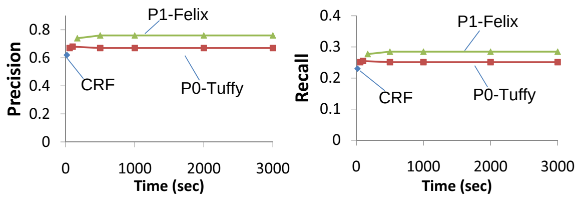

Figure 7: Time-Quality Plot on NFL Winner-loser

and then identify possible person/organization names using simple heuristics (e.g., common person name dictionaries and keywords such as "University"). To handle noise in the entity extraction results, our P1 program performs both affiliation extraction and coref resolution using ideas similar to Figure 1. We run both FELIX and TUFFY on P1. P1-TUFFY crashed without producing any results.

We plot the precision-recall curves of CIMPLE, P0-TUFFY, and P1-FELIX by varying the k in "*top-k affiliations of each person*" (Figure 4(b)).[11] For any given precision, P0-TUFFY has substantially higher recall than CIMPLE. This suggests that statistical approaches to global IE may help deterministic rule-based systems. Furthermore, with the addition of coref subtasks, P1-FELIX improved the output quality by yet another large margin: at the same precision, the recall of P1-FELIX is 2-4 times as high as CIMPLE. Comparing the top-20 affiliation predictions of each person, P1-FELIX achieves 160% improvement in recall with the same precision as CIMPLE. This further reinforces the advantage of global IE, and coref as a crucial IE subtask in particular. The fact that P1-FELIX runs efficiently while P1-TUFFY crashed shows that our operator-based approach is crucial to running global IE programs.

In Figure 6, we plot how the precision/recall of *"top-2 affiliations of each person"* evolve as FELIX does more iterations. Since CIMPLE does not use an iterative algorithm, it is represented by a dot in each graph. We see that the quality (precision and recall) of both P0-TUFFY and P1-FELIX improves as we run for longer duration, and then converges after a while. For example, in the recall graph, P0-TUFFY starts with about the same recall as the baseline value of CIMPLE; but by the time that TUFFY takes the same time as CIMPLE, the recall of TUFFY becomes about 50% higher than CIMPLE. On this small value of k, the precision of FELIX is about 40% lower than CIMPLE, FELIX's recall is 200% higher than CIMPLE. For higher values of k, FELIX maintains its recall edge with the comparable precision.

NFL On the NFL dataset, we extract winner-loser pairs. There are 1,100 sports news articles in the corpus. We obtain ground truth of game results from the web. We use 610 of the articles together with ground truth to train a CRF model that tags each token in the text as either WINNER, LOSER, or OTHER. We then apply this CRF model on the remaining 500 articles to generate probabilistic tagging of the tokens. Those 500 articles report on a different season of NFL games than the training articles, and we have ground truth on game results (in the form of winner-loser-date triples). We take the publication dates of the articles and align them to game dates. For each sentence containing a WINNER token with probability p and a LOSER token with probability q, if both tokens can be resolved to NFL team names, we emit a pair of these

[11]As we do not have ground truth for the DBLife corpus, we crawl a subset of the ACM author profile data (`http://www.acm.org/membership/author_pages`) and use it as ground truth. Since the ACM profile data is only a (noisy) subset of the real ground truth, it was no surprise that the overall quality numbers are low.

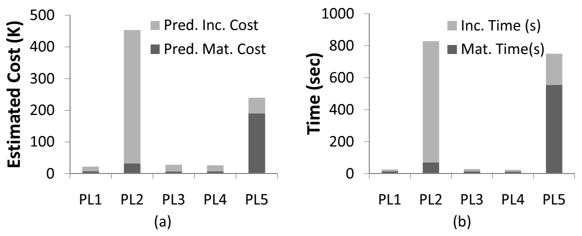

Figure 8: Comparison between (a) estimated cost and (b) actual running time of random plans

teams with score $(p + q)/2$. This generates a ranked list of winner-loser-date triples. We plot the precision-recall curve of this result in Figure 4(c).

The P0 program on NFL is adapted from the rules developed by a research team in the Machine Reading project. Those rules model simple domain knowledge such as "a winner cannot be a loser on the same day" and "a team cannot win twice on the same day." We run P0 with the CRF tagging results as input on TUFFY. As shown in Figure 4(c), P0 significantly improved upon the quality of CRF. Adding in coreference of the team mentions, we augment P0 into a program P1, and run it on both TUFFY and FELIX. P1-TUFFY crashed when attempting to generate 10^8 tuples during grounding. In contrast, P1-FELIX runs smoothly and produces result quality that is yet another big margin higher than P0. This three-stage improvement demonstrates the desirability of global IE approaches, and coref as an critical part of IE in particular. Moreover, to cope with complex programs such as P1, it is vital to take an operator-based approach (as done by FELIX) rather than a monolithic approach (as done by TUFFY).

In Figure 7, we take the top-100 predictions from each system, and see that FELIX runs with the same efficiency as TUFFY, but produced significantly higher-quality output. This again shows that coref is an important subtask in IE.

C.3 Accuracy of Cost Estimation

We verify that our cost model (Section 3.3) and estimation method (Appendix B.1) can approximate the relative running time cost of different materialization plans. To do this, we take the P1 program on Enron and run it on an Enron dataset with 5,000 emails. We generate five random materialization plans, ask FELIX's optimizer to estimate the cost of each plan, and then measure the actual running time (materialization + incremental evaluation) of the Coref operator with each plan. As shown in Figure 8, different plans can have orders of magnitude difference in both estimated cost and actual running time. The estimations of FELIX strongly correlate with the actual running time. Thus, not only will the optimizer of FELIX avoid catastrophic materialization plans, it can also pick among the plans that are the cheapest.

C.4 Efficiency of Felix's Optimizer

In this experiment, we show that FELIX's optimizer can explore a large number of materialization plans while incurring very small overhead. To do this, we take the P1 programs of Enron and DBLife, and run FELIX on Enron-100K and DBLife, respectively. We measure the number of plans explored by FELIX's optimizer and the total time spent on cost estimation. As shown in Table 5, the FELIX optimizer is efficient at exploring the search space of materialization plans. For example, on Enron, FELIX only took about one second to explore more than one hundred plans.

	DBLife	Enron
#data movement operators	6	9
#plans explored	37	129
time spent on planning	0.362 sec	1.132 sec

Table 5: Statistics for Felix planning

C.5 Cyclic Execution Plans

We evaluate the effect of running cyclic plans instead of pipeline plans on the DBLife affiliation task. Recall that the P1 program of DBLife resembles the program in Figure 1, and so contains cycles in the logical plan compiled by FELIX. In Section 4, we provided the result quality of running a pipeline execution plan that runs person and organization Coref operators first, and then performs the TUFFY operator. In this experiment, we run a cyclic plan instead, and measure the result quality after each round of cycling.

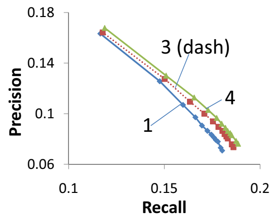

Figure 9: Quality improves as FELIX runs cyclic rounds.

We run the same pipeline for 8 rounds, but the quality converged after 4 rounds. The quality of round 2 is between round 1 and round 3. For clarity, we only plot rounds 1, 3, and 4 in Figure 9. We see that the result quality slightly improves as we run more than one round of the Coref-TUFFY operators. This result suggests that there may be some improvement with such cyclic plans. (We continue to explore this.) As a result, there may be an interesting tradeoff between performance and quality. For example, if quality is critical and the operators can execute very efficiently, it is preferable to run multiple rounds. On the other hand, if the task is not quality-critical and the round-by-round improvement is minimal, then one may decide to run one round only instead.

D Extended Related Work

The idea of exploiting rich correlations in information extraction has been extensively explored in the literature [5, 23, 38, 41]. While their goal is to explore the effectiveness of different correlations to particular applications, our work aims to scale up general frameworks to enable more systematic approaches to global IE. In StatSnowball [41], Zhu et al. demonstrate high quality results of a global IE approach using MLNs. To address the scalability issue of generic MLN inference, they made additional independence assumptions in their IE tasks. In contrast, the goal of FELIX is to automatically scale up statistical inference given the same input program.

Our materialization tradeoff strategy is related to view materialization and selection [14, 32] in the context of data warehousing. However, we have a very different problem setting: (1) our views are read-only, so we do not have maintenance cost; (2) our tradeoff is based on more predictable and finer-grained modeling of work loads; and (3) we leverage the RDBMS's estimator as core measurement in our cost model. The idea of lazy-eager tradeoff in view materialization or query answering has also been applied to probabilistic databases [40]. Their goal is efficiently maintaining the intermediate results, rather than choosing a materialization strategy. Similar in spirit to our approach is Sprout [22], which considers lazy-versus-eager plans for when to apply confidence computation. In contrast, we consider materialization between gray-box statistical operators.

References

[1] N. Ailon, M. Charikar, and A. Newman. Aggregating inconsistent information: ranking and clustering. *JACM*, 2008.

[2] A. Arasu, C. Ré, and D. Suciu. Large-scale deduplication with constraints using Dedupalog. In *ICDE 2009*.

[3] N. Bansal, A. Blum, and S. Chawla. Correlation clustering. *Machine Learning*, 56(1):89–113, 2004.

[4] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *TKDD*, 2007.

[5] S. Brin. Extracting patterns and relations from the world wide web. *WWW*, pages 172–183, 1999.

[6] A. Carlson, J. Betteridge, B. Kisiel, B. Settles, E. Hruschka Jr, and T. Mitchell. Toward an architecture for never-ending language learning. In *AAAI*, 2010.

[7] L. Chiticariu, R. Krishnamurthy, Y. Li, S. Raghavan, F. Reiss, and S. Vaithyanathan. SystemT: An algebraic approach to declarative information extraction. In *ACL 2010*.

[8] A. Doan and et al. DBLife: A community information management platform for the database research community.

[9] P. Domingos and D. Lowd. Markov Logic: An Interface Layer for Artificial Intelligence. 2009.

[10] P. Domingos et al. `http://alchemy.cs.washington.edu/`.

[11] I. Fellegi and A. Sunter. A theory for record linkage. *Journal of the American Statistical Association*, 64(328):1183–1210, 1969.

[12] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, 1999.

[13] A. Gionis, H. Mannila, and P. Tsaparas. Clustering aggregation. *TKDD*, 1(1):4, 2007.

[14] H. Gupta and I. Mumick. Selection of views to materialize in a data warehouse. *TKDE*, pages 24–43, 2005.

[15] J. Hilbe. *Logistic regression models.* CRC Press, 2009.

[16] J. Lafferty, A. McCallum, and F. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML*, 2001.

[17] B. Liu, L. Chiticariu, V. Chu, H. Jagadish, and F. Reiss. Automatic Rule Refinement for Information Extraction. *VLDB*, 3(1), 2010.

[18] A. McCallum, K. Schultz, and S. Singh. Factorie: Probabilistic programming via imperatively defined factor graphs. In *NIPS*, 2009.

[19] E. Michelakis, R. Krishnamurthy, P. Haas, and S. Vaithyanathan. Uncertainty management in rule-based information extraction systems. In *SIGMOD*. ACM, 2009.

[20] B. Milch, B. Marthi, S. Russell, D. Sontag, D. Ong, and A. Kolobov. BLOG: Probabilistic models with unknown objects. In *IJCAI*, 2005.

[21] F. Niu, C. Ré, A. Doan, and J. Shavlik. Tuffy: Scaling up Statistical Inference in Markov Logic Networks using an RDBMS. In *VLDB 2011*.

[22] D. Olteanu, J. Huang, and C. Koch. Sprout: Lazy vs. eager query plans for tuple-independent probabilistic databases. In *ICDE*, 2009.

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

[24] H. Poon and P. Domingos. Joint inference in information extraction. In *AAAI '07*.

[25] R. Ramakrishnan and J. Ullman. A survey of deductive database systems. *The journal of logic programming*, 1995.

[26] M. Richardson and P. Domingos. Markov logic networks. *Machine Learning*, 2006.

[27] S. Riedel. Cutting Plane MAP Inference for Markov Logic. In *SRL 2009*, 2009.

[28] S. Riedel and I. Meza-Ruiz. Collective semantic role labeling with Markov logic. In *CoNLL '08*.

[29] H. Schmid. Improvements in part-of-speech tagging with an application to German. 1999.

[30] J. Seib and G. Lausen. Parallelizing datalog programs by generalized pivoting. In *Proceedings of the tenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, PODS '91, pages 241–251, New York, NY, USA, 1991. ACM.

[31] P. Sen, A. Deshpande, and L. Getoor. PrDB: managing and exploiting rich correlations in probabilistic databases. *J. VLDB*, 2009.

[32] A. Shukla, P. Deshpande, J. Naughton, et al. Materialized view selection for multidimensional datasets. In *VLDB*, 1998.

[33] S. Singh, K. Schultz, and A. McCallum. Bi-directional joint inference for entity resolution and segmentation using imperatively-defined factor graphs. *PKDD*, 2009.

[34] F. Suchanek, M. Sozio, and G. Weikum. SOFIE: a self-organizing framework for information extraction. In *WWW*, pages 631–640. ACM, 2009.

[35] M. Theobald, M. Sozio, F. Suchanek, and N. Nakashole. URDF: Efficient Reasoning in Uncertain RDF Knowledge Bases with Soft and Hard Rules. *MPI Technical Report*, 2010.

[36] J. Ullman. Implementation of logical query languages for databases. *TODS*, 10(3):321, 1985.

[37] M. J. Wainwright and M. I. Jordan. *Graphical Models, Exponential Families, and Variational Inference*. Now Publishers Inc., Hanover, MA, USA, 2008.

[38] R. Wang and W. Cohen. Iterative set expansion of named entities using the web. In *ICDM'08*, 2009.

[39] D. Weld, R. Hoffmann, and F. Wu. Using Wikipedia to bootstrap open information extraction. *SIGMOD Record*, 2009.

[40] M. Wick, A. McCallum, and G. Miklau. Scalable Probabilistic Databases with Factor Graphs and MCMC. *VLDB*, 3(1), 2010.

[41] J. Zhu, Z. Nie, X. Liu, B. Zhang, and J. Wen. Statsnowball: a statistical approach to extracting entity relationships. In *WWW*, 2009.