Query-Driven Sampling for Collective Entity Resolution

Christan Grant
University of Oklahoma
cgrant@ou.edu

Daisy Zhe Wang
University of Florida
daisyw@cise.ufl.edu

Michael Wick
University of Massachusetts, Amherst
mwick@cs.umass.edu

Abstract

Entity Resolution is the process of determining records (mentions) in a database that correspond to the same real-world entity. Traditional pairwise ER methods can lead to inconsistencies and low accuracy due to localized decisions. Leading ER systems solve this problem by collectively resolving all records using a probabilistic graphical model and Markov chain Monte Carlo (MCMC) inference. However, for large datasets this is an extremely expensive process. One key observation is that, such exhaustive ER process incurs a huge up-front cost, which is wasteful in practice because most users are interested in only a small subset of entities.

In this paper, we advocate pay-as-you-go entity resolution by developing a number of query-driven collective ER techniques. We introduce two classes of SQL queries that involve ER operators — selection-driven ER and join-driven ER. We implement novel variations of the MCMC Metropolis Hastings algorithm to generate biased samples and selectivity-based scheduling algorithms to support the two classes of ER queries. Finally, we show that query-driven ER algorithms can converge and return results within minutes over a database populated with the extraction from a newswire dataset containing 71 *million mentions.*

1. Introduction

Entity resolution (ER) is the process of identifying and linking/grouping different manifestations (e.g., mentions, noun phrases, named entities) of the same real world object. It is a crucial task for many applications including knowledge base construction, information extraction, and question answering. For decades, ER has been studied in both database and natural language processing communities to link database records or to perform entity resolution over extracted mentions (noun phrases) in text.

ER is a notoriously difficult and expensive task. Traditionally, entities are resolved using strict pairwise similarity, which usually leads to inconsistencies and low accuracy due to localized, myopic decisions [15]. More recently, collective entity resolution methods have achieved state-of-the-art accuracy because they leverage relational information in the data to determine resolution jointly rather than independently [3]. However, it is expensive to run collective ER based on probabilistic graphical models (GMs), especially for cross-document entity resolution, where ER must be performed over millions of mentions.

In previous approaches, collective ER is performed exhaustively over all the mentions in a data set, returning all entities. Researchers have developed new methods to perform large-scale cross-document entity resolution over parallel frameworks [11, 15]. However, in many ER applications, users are only interested in one or a small subset of entities. This key observation motivates query-driven ER, an alternative approach to solving the scalability problem for ER.

Compared to previous ER models and algorithms, query-driven techniques in this paper scale to data sets that are in many cases three orders of magnitude larger. Moreover, the ER model in this paper is general enough to take both bibliographic records and mentions extracted from unstructured text. Query-driven ER techniques over GMs can also be generalized for other applications to perform query-driven inference.

Because exhaustive ER is expensive it is common to use blocking techniques to partition the data set into approximately similar groups called canopies. Query-driven ER in this paper differs from blocking in two important ways: 1) deterministic blocks are replaced by a pairwise distance-based metric, and 2) blocks (or canopies) are implicit to the query-driven ER data set and do not have to be created in advanced. The latter point, implicit blocking, is realized using a data structure created based on the similarity to a query mention. This data structure allows parameters to include or remove mentions from the working data set. This property is similar to the iterative blocking technique [14], which is shown to improve ER accuracy. Such an approach can dramatically amortize the overall ER cost suitable for the pay-as-you-go paradigm in dataspaces [9].

To support ER driven by queries, we develop three sampling algorithms for MCMC inference over graphical models. More specifically, instead of a uniform sampling distribution, we sample on a distribution that is biased to the query. We develop a query-driven sampling techniques that maximizes the resolution of the target query entity (target-fixed) and biases the samples based on the pairwise similarity metric between mentions and query nodes (query-proportional). We also introduce a hybrid method that performs query-proportional sampling over a fixed target. We develop two optimizations to the query-proportional and hybrid methods to model the similarity and dissimilarity between the mentions and the query entity, i.e., attract and repel scores. In the first target-fixed algorithm, we adapt the samples to resolve the query entity. The second query-proportional algorithm, selects mentions based on their probabilistic similarity to the query entity. The third hybrid algorithm combines the two approaches. A summary of approaches can be found in Table 3.

When a user is interested in resolving more than one entity we employ multi-node ER techniques. To implement multi-node ER queries, single-node ER techniques may be naively performed iteratively to resolve one entity at a time. However, such an algorithm can lead to un-optimized resource allocation if the same number of samples is generated for each target entity, or low throughput if one of the entities has a disproportionately low convergence rate. To alleviate this problem, we present three multi-query ER algorithms that schedule the sample generation among query nodes in order to improve overall convergence rate.

In summary, the contributions of this paper are the following:

- We define a query-driven ER problem for cross-document, collective ER over text extracted from unstructured data sets;
- We develop three single-node algorithms that perform focused sampling and reduce convergence time compared to a non-query-driven baseline (Section 3). We develop two influence functions that use attract and repel techniques to grow or shrink query entities (Section 4.1);
- We develop scheduling algorithms to optimize the overall convergence rate of the multi-query ER (Section 4.2). The best scheduling algorithm is based on selectivity of different target entities (Section 4.3).

The results show that query-driven ER algorithms is a promising method of enabling realtime, ad-hoc, ER-based queries over large data sets. Single node queries of different selectivity converge to a high-quality entity within 1-2 minutes over a newswire data set containing 71 million mentions. Experiments also show that such real-time ER query answering allows users to iteratively refine ER queries by adding context to achieve better accuracy (Section 5).

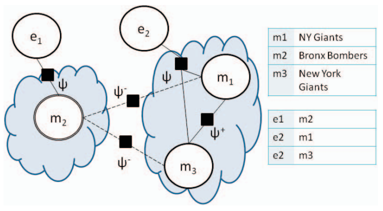


Figure 1: Three node factor graph. Circles (random variables) with m_i represent mentions and those with e_i represent entities. Clouds are added for visual emphasis of entity clusters

2. Preliminaries

In this section, we present a foundation of concepts discussed in this paper. We start with an introduction of factor graphs then discuss sampling techniques over this model. Finally, we formally introduce state-of-the-art entity resolution approaches and explain the origin.

2.1. Factor Graphs Graphical models are a formalism for specifying complex probability distributions over many interdependent random variables. Factor graphs are bipartite graphical models that can capture arbitrary relationships between random variables through the use of factors [8]. As depicted in Figure 1, links always connect random variables (represented as circles) and factor nodes (represented as black squares). Factors are functions that take as input the current setting of connected random variables, and output a positive real-valued scalar indicating the compatibility of the random variables settings. The probability of a setting to all the random variables is a normalized product of all the factors. Intuitively, the highest probability settings have variable assignments that yield the highest factor scores.

Formally, a factor graph $\mathcal{G} = \langle \mathbf{x}, \psi \rangle$ contains a set of random variables $\mathbf{x} = \{x_i\}_1^n$ and factors $\boldsymbol{\psi} = \{\psi_i\}_1^m$. Each factor ψ_i maps the subset of variables it is associated with to a non-negative compatibility value. The probability of a setting ω among the set of all possible settings Ω occurring in the factor graph is given by a probability measure:

$$\pi(\omega) = \frac{1}{Z} \sum_{x \in \omega} \prod_{i=1}^{m} \psi_i(x^i), \quad Z = \sum_{\omega \in \Omega} \sum_{x \in \omega} \prod_{i=1}^{m} \psi_i(x^i)$$

where x^i is the set of random variables that neighbor the factor $\psi_i(\cdot)$ and Z is the normalizing constant.

Querying graphical models produces the most likely setting for the random variables. A query on a factor graph is defined as a triple $\langle x_q, x_l, x_e \rangle$ where x_q is the set of nodes in question, x_l is a set of latent nodes (entities) that are marginalized and x_e is a set of evidence nodes (observed mentions). A query task is a sum over the all latent variables and the maximization of the query probability. A query over the factor graph is defined as

$$\mathcal{Q}(x_q, x_l, x_e, \pi) = \text{argmax}_{x_q} \sum_{v_l \in x_l} \pi(x_q \cup v_l \cup x_e).$$

To obtain the best setting of the queries in question, inference is required.

Several methods exist for performing inference over factor graphs. The entity resolution factor graph, being pairwise, is dense and highly connected. This property suggests the best methods for inference are Markov Chain Monte Carlo (MCMC) methods; in particular, we use a Metropolis Hastings variant [8]. We refer the reader to literature previous work for a detailed discussion on inference over factor graphs and a deviation of the technique [16].

The idea of MCMC-MH is to propose modifications to a current setting and use the model to decide whether to accept or reject the proposed setting as a replacement for the current settings. When the models are being scored only the factors touching nodes with changed values, the Markov blanket, needs to be recomputed. We accept or reject changes so the model can iteratively proceed to an optimal setting.

More formally, consider an MCMC transition function $T : \Omega \times \Omega \to [0, 1]$ where given the current setting ω we can sample a subsequent setting ω'.

The probability of accepting a transition given a graphical model distribution π is:

$$A(\omega, \omega') = \min\left(1, \frac{\pi(\omega')T(\omega, \omega')}{\pi(\omega)T(\omega', \omega)}\right). \qquad (2.1)$$

Additionally, the intractable partition function Z is canceled out, making sample generation inexpensive. This property allows us to calculate the probability of accepting the next state by simply computing the difference in score between the next and current state [16].

We say the algorithm converges when a steady state is reached.[1] Intelligently sampling next states decreases the time to convergence. Convergence in MCMC is difficult to verify [5], we discuss convergence estimation in Section 5.1.

2.2. Cross-Document Entity Resolution Cross-document ER is the problem of clustering mentions that appear across independent sets of documents into groups of mentions that correspond to the same real world entity. These ER tasks typically assume a set of preprocessed documents and perform linking across documents [2, 11]. The scale of the cross-document ER problem is typically several orders of magnitude more than intra-document ER. There are no document boundaries to limit inference scope and all entity mentions may be distributed arbitrarily across millions of documents.

To model cross-document ER, let $\mathcal{M} = \{m_1, \ldots, m_{|\mathcal{M}|}\}$ be the set of mentions in a data set. Each mention m_i contains a set of attribute-value data points. Let $\mathcal{E} = \{e_1, \ldots, e_{|\mathcal{M}|}\}$ represent the set of entities where each e_i contain zero or more mentions. Note, we assume the maximum number of entities is no more than the number of mentions and no less than 1. Each mention may correspond to a unique entity or all mentions may correspond to a single entity.

The baseline method of entity resolution is a straightforward application of the MCMC-MH algorithm. We show pseudo code for the baseline method in Algorithm 1.

Algorithm 1 The baseline entity resolution algorithm using Metropolis-Hastings sampling

INPUT: A set of unresolved entities $\mathcal{E}$ each with one mention m.
INPUT: A positive integer $samples$.
OUTPUT: A set of resolved entities $\mathcal{E}$.

```
1: while $samples$-- $> 0$ do
2:     $e_i \sim_u \mathcal{E}$
3:     $e_j \sim_u \mathcal{E}$
4:     $m \sim_u e_i$
5:     $\mathcal{E}' \leftarrow$MOVE$(\mathcal{E}, m, e_j)$
6:     if SCORE$(\mathcal{E}) <$ SCORE$(\mathcal{E}')$ then
7:         $\mathcal{E} \leftarrow \mathcal{E}'$
8:     end if
9: end while
       return $\mathcal{E}$
```

Algorithm 1 takes as input a set of entities $\mathcal{E}$ and $samples$ which is the number of iterations of the algorithm or a function to estimate convergence. The algorithm samples two entities from the entity set and moves one random[2] mention into the other entity. After the move, the algorithm checks for an improvement in the overall score of the model. If the model score improves, the changes are kept, otherwise the proposed changes are ignored. The SCORE function sums the weights of all the edges in the given entity to obtain a value for the model. This is equivalent to the probability of the setting $\pi(\cdot)$ as described in Section 2.1.

3. Query-Driven Entity Resolution Algorithms

Query-driven ER is an understudied problem; in this section we describe our approach to query-driven ER with one entity (single-query ER) and with multiple entities (multi-query ER). First, we give a graphical intuition of query-driven ER algorithms.

3.1. Single-query ER Single-query ER algorithms are the class of algorithms that resolve a single query-node. In

[1]We refer to literature for a more detailed description of convergence [16].

[2]Given a set X, the function $x \sim_u X$ makes a uniform sample from the set X into a variable x.

particular, the target-fixed ER algorithm aims to focus a majority of the proposals on resolving the query entity. The algorithm fixes the query node as the target entity and then randomly selecting a source node to merge into the entity of the target query node. This focus on building the query entity in this type of importance sampling means the query entity should be resolved faster than if we sampling each entity uniformly.

A query-driven ER algorithm that only selects the query-node as the target entity during sampling will create errors because such an algorithm is unable to remove erroneous mentions from the query entity. To prevent these errors, we allow the algorithm to occasionally back out of poor decisions, that is, it makes non-query specific samples. Shown in Algorithm 2, target-fixed entity resolution adapts Algorithm 1 but it allows parameters to specify the proportion of time the different sampling methods are selected.

In addition to the input mentions $\mathcal{E}$ from Algorithm 1, target-fixed entity resolution takes as input a query node q. The output of the algorithm is a resolved query entity and other partially resolved entities.

For each sampling iteration the algorithm can make two decisions. The sampler may propose to merge a random source node that is not already a member of the query entity into the target query entity. Alternatively, the algorithm merges a random node with a random entity.

Algorithm 2 Target-fixed entity resolution algorithm

```
Input:   A query node $q$.
         A set of entities $\mathcal{E}$ each with one mention $m$.
         A positive integer $samples$.
Output:  A set of resolved entities $\mathcal{E}'$.i

 1: $\mathcal{E}' \leftarrow \mathcal{E} \cup q$
 2: while $samples$-- $> 0$ do
 3:     if RANDOM() $< \tau_\alpha$ then
 4:         $e_i \sim_u \mathcal{E}'$
 5:         $e_j \leftarrow q.entity$
 6:         $m \sim_u e_i$
 7:     else
 8:         $e_j \leftarrow \{e | \exists e, e \in \mathcal{E}', e \neq q.\text{entity}\}$
 9:         $e_i \leftarrow \{e | \exists e, e \in \mathcal{E}', e \neq e_j\}$
10:         $m \sim_u e_i$
11:     end if
12:     $\mathcal{E}'' \leftarrow$MOVE$(\mathcal{E}', m, e_j)$
13:     if SCORE$(\mathcal{E}') <$ SCORE$(\mathcal{E}'')$ then
14:         $\mathcal{E}' \leftarrow \mathcal{E}''$
15:     end if
16: end while
        return $\mathcal{E}'$
```

On lines 3 to 6 the algorithm takes a uniform sample from the list of entities. If the sampled entity is the same as the query entity it tries again and samples a distinct entity. A node is drawn from this entity. The probability of this block being entered is τ_α. Lines 7 to 10 are entered with a probability $(1 - \tau_\alpha)$. This block performs a random entity assignment in the same manner as Algorithm 1. This block offsets the aggressive nature of the target-fixed algorithm by probabilistically backing out of any bad merges. Finally, the block starting from line 12 to line 15 scores the new arrangement and accepts if this improves the model score. We discuss parameter settings in Section 4.4.

Table 1: Mentions sets $\mathcal{M}$ from a corpus

id	Mention	...
m_1	NY Giants	...
m_2	Bronx Bombers	...
m_3	New York Giants	...
m_4	Yankees	...
m_5	Brooklyn Dodgers	...
m_6	The Yanks	...

Example Take the synthetic mention set $\mathcal{M}$ shown in Table 1 and a query node q, the baseball team 'New York Yankees', in Table 2. This is the result of the approximate match of query q over a larger data set (blocking). The mentions of $\mathcal{M}$ may be initialized by assigning each mention to its own entity. After a successful run of traditional entity resolution the set of entities clusters are

$$\{\langle q, m_2, m_4, m_6\rangle, \langle m_1, m_3\rangle, \langle m_5\rangle\}.$$

For query-driven scenario the only entity we are interested in is $\langle q, m_2, m_4, m_6\rangle$. Each mention in this query entity is an alias for the 'New York Yankees' baseball team. The other two mentions represent the 'New York Giants' football team and the 'Brooklyn Dodgers' baseball team respectively.

The target-fixed algorithm attempts to merge nodes with the query entity one mention at a time and the merge is accepted if it improves the score of the overall model. We can see in the example that a merge of m_1 and m_3 may improve the overall model because they have similar keywords but one refers to the query entity and the other to different football team. The target-fixed algorithm can correct this type of error by probabilistically backing out of errors by moving mentions in the query node to a new entity as show in line 7 to line 10 of Algorithm 2.

3.2. Multi-query ER A user may want to resolve more than one query entity, that is, she may be interested in resolving a watch list of entities over the data set. To support multiple queries, first merge the canopies of each query node in the watch list to obtain a subset of the full graphical model containing only the nodes similar to query nodes. To resolve the entities we can use query-proportional methods iteratively over each query node. We define two classes of schedules, namely, static and dynamic.

Table 2: Example query node q

id	Mention	...
q	New York Yankees	...

Static schedules are formulated before sampling while dynamic schedules are updated in response to estimated convergence. The two static schedules we develop are random and selectivity-based. In random scheduling each query node from the watch list is selected in a round robin style. Selectivity-based scheduling is a method of ordering multi-query samples to schedule proposals in proportion to the selectivity of the query node. Selectivity, in this case, is defined as the number of mentions retrieved using an approximate match of the data set, or the query node's contribution to the total new graphical model. For example, the selectivity of our query node q in Table 2 the selectivity is simply the size of $\mathcal{M}$, shown in Table 1.

Random-based scheduling method performs well if all query nodes come from similar selectivity. Otherwise, if the selectivity of each query node vary, one query node may require more sampling compared to the others. If one query node needs a lot of samples to converge, it may take the whole process a long time to complete and cycles may be wasted on other nodes that have already converged.

In addition to scheduling samples in proportion to their selectivity, we can schedule samples dynamically, depending on the progress of each query entity. To perform dynamic scheduling we need to know how each query entity is progressing towards convergence. To estimate the running convergence we do not use standard techniques in literature because scheduling needs to occur before the model is close to convergence [5]. Instead, we estimate the convergence by measuring the fraction of accepted samples over the last N samples of each query in the watch list. The two dynamic scheduling algorithms are closest-first and sampling the farthest-first. In closest-first we queue up the query node that has the lowest positive average number of accepted nodes over the last N proposals. This scheduling method performs inference for the node that is closest to being resolved so it can move on to other nodes. Alternatively, the farthest-first algorithm schedules the node that has the highest convergence rate. This scheduling algorithm makes each query entity progress evenly.

4. Optimization of Query-Driven ER

The previous ER techniques aggressively attempt to resolve the query entity. However, if the query node is not representative of the query items performance of target-fixed ER can lead to undesirable results. We do not explore this trade-off; we assume users can select representative query nodes. In this section, we introduce optimizations to create approximate query-driven samples based on the query node. We first discuss the influence function that is used to make query-driven proposals. We then discuss the attract and repel versions of the influence function followed by two new algorithms. We end with implementation details and a summary of our query-driven algorithms.

4.1. Influence Function: Attract and Repel To retrieve nodes from a graphical model that is similar to a query node we employ the notion of influence. Our assumption is that nodes that are similar have a high probability of being coreferent. An influence trail score between two nodes in a graphical model can be computed as the product of factors along their active trail as defined in literature [16]. For a node $m_i \in \mathcal{M}$ and the query node $q \in \mathcal{M}$ the influence of m_i on the query node is defined as:

$$\mathcal{I}(m_i, q) = \sum_{j \in \mathcal{F}} w_j \psi_j(m_i, q)$$

where $\mathcal{F}$ is the world of pairwise features and the feature weight and log-linear function are, respectively, w_j and ψ_j. The influence function $\mathcal{I}$ is an implementation of this trail score.

The influence function takes a set of entities — or the equivalent GM — and a query node q as parameters. The parameters to an influence function can be over the whole database or a canopy. Over several invocations of the function, $\mathcal{I}$ returns mentions from the graphical model with a frequency proportionate to their influence on q. If a mention has little or no influence, the influence acts as a blocking function, infrequently returning the mention. Recall influence is the distance active trail distance to query node. To implement the influence function we build a data structure based on an algorithm by Vose [13], hereafter referred to as a Vose structure.

The attract method initializes each mention in the canopy in its own entity, and then mentions are merged until the convergence. The target-fixed algorithm discussed in Section 3.1 is explained using this method. The attract method works well for low quality canopies, or canopies that require a small number or items to merge. Conversely, the repel method works well with high quality canopies or when most items in a canopy belong to the query entity.

The repel method initializes each mention in the canopy into a single entity. Then proposals are made to remove mentions from the entity so we are left with only the nodes in the query entity. We discuss this method using the hybrid algorithm in Section 4.3. To build an influence function for the repel method we can use the same method and we only need to normalize and invert the influence scores. We refer to this as co-influence or $\bar{\mathcal{I}}$.

4.2. Query-proportional ER In the query-proportional sampling algorithm, on every iteration, the source mention and target entity are selected in proportion to its distance to the query entity. Instead of focusing solely on the query entity, this algorithm prioritizes samples using a measure that represents probability of a mention being coreferent with the query entity.

Algorithm 3 Query-proportional algorithm

```
Input:   A query node $q$ to drive computation.
         A set of entities $\mathcal{E}$ each with one mention $m$.
         A positive integer $samples$.
         A function $\mathcal{I}$ that samples from nodes entities according to its
         influence on a mention.
Output:  A set of resolved entities $\mathcal{E}'$.

1: $\mathcal{E}' \leftarrow \mathcal{E} \cup q$
2: while $samples$-- $> 0$ do
3:     $m_1 \leftarrow \mathcal{I}(\mathcal{E}', q)$
4:     $m_2 \leftarrow \mathcal{I}(\mathcal{E}', q)$
5:     $\mathcal{E}'' \leftarrow$MOVE$(\mathcal{E}', m_1, m_2.entity)$
6:     if SCORE$(\mathcal{E}')$ $<$ SCORE$(\mathcal{E}'')$ then
7:         $\mathcal{E}' \leftarrow \mathcal{E}''$
8:     end if
9: end while
   return $\mathcal{E}'$
```

That is, each node p in the graphical model $\mathcal{G}$ is selected on the active trail between itself and the query node q. This algorithm merges nodes that are similar to the query node with an increased frequency.

Prior to query-proportional sampling, a Vose structure ($\mathcal{I}$,§ 4.1) is created. The $\mathcal{I}$ influence structure takes a query node q and the global graphical model $\mathcal{E}$ then returns a sampled mention. Queries over $\mathcal{I}$ are made multiple times, the distribution of the nodes returned is proportional to their influence. Algorithm 3 describes the query-proportional algorithm.

For each iteration, the algorithm selects mentions using the influence function (line 3 and line 4). Then, one mention m_1 is moved into the entity of m_2. Mentions m_1 and m_2 have a higher probability of being coreferent and therefore a higher probability of a merge occurring in the query entity compared to random selections as in Algorithm 1. As a corollary, the influence sampling property creates many small entities that are similar to the query entity.

During query-proportional sampling more entities that are similar to the query node are created. Some of the mentions created in intermediate entities during query-proportional sampling will move to the query entity. This is a big advantage when performing entity-to-entity merges (as opposed to mention to entity merges). In this paper, we do not investigate this extension to the algorithm.

4.3. Hybrid ER The best of both the target-fixed and query-proportional algorithms can be combined to create a hybrid algorithm. Like the target-fixed algorithm, the hybrid method aggressively fixes the target as the query entity. The hybrid method also chooses its source node using the influence function in the same manner as the query-proportional algorithm.

Algorithm 4 shows the hybrid algorithm using the repel method. With probability τ_α the algorithm chooses a mention using the repel method ($\bar{\mathcal{I}}$) and moves it to an

Table 3: Summary of algorithms and their most common methods for proposal jumps

	source	target
Baseline	random	random
Target-Fixed	random	fixed
Query-Proportional	proportional	proportional
Hybrid	proportional	fixed

entity that is not the query node. This is the opposite of merging a node into the query entity. Pseudocode is listed on lines 3 to line 5.

Algorithm 4 Hybrid-Repel algorithm

```
Input:   A set of entities $\mathcal{E}$, where one contains all the mentions $m$ and the
         others are empty.
         A positive integer $samples$.
         A query node $q$.
         A function $\overline{\mathcal{I}}$ that samples from nodes entities according to its
         influence on a mention.
Output:  A set of resolved entities $\mathcal{E}'$.

 1: $\mathcal{E}' \leftarrow \mathcal{E} \cup q$
 2: while $samples$-- $> 0$ do
 3:     if RANDOM() $< \tau_\alpha$ then
 4:         $m \leftarrow \overline{\mathcal{I}}(\mathcal{E}', q)$
 5:         $e_i \leftarrow \{e | \exists e, e \in \mathcal{E}', e \neq q$.entity$\}$
 6:     else
 7:         $e_i \sim_u \mathcal{E}'$
 8:         $e_j \leftarrow \{e | \exists e, e \in \mathcal{E}', e \neq e_i\}$
 9:         $m \sim_u e_j$
10:     end if
11:     $\mathcal{E}'' \leftarrow$MOVE$(\mathcal{E}', m, e_i)$
12:     if SCORE $(\mathcal{E}')$ $<$ SCORE$(\mathcal{E}'')$ then
13:         $\mathcal{E}' \leftarrow \mathcal{E}''$
14:     end if
15: end while
    return $\mathcal{E}'$
```

Also, the sampling over the query nodes for each algorithm can also be perform in parallel. In our method, a thread selects a query node using a random schedule as described in Section 3.2. The system will use the Vose structure associated with the query node to set up a proposal move. The system attempts to obtain a locks for both entities involved in the proposal. If the system is unable to obtain a lock on either of the two entities the system will back out and resample new entities. When the number of query nodes is small the query-driven algorithms experience lot of contention at the entities containing the query nodes. In these circumstances, the system will back out and either restart the proposal process or attempt a baseline proposal. This avoids waiting for locked entities and keeps the sampling process active. In Section 5.5 we demonstrate the parallel hybrid method over a large data set.

4.4. Algorithms Summary Discussion Algorithms 2, 3 and 4 are modifications of proposal jumps found in the baseline Algorithm 1. Table 3 describes the proposal process for each algorithm by its preferred jump method.

The target-fixed algorithm builds the query entity by aggressively proposing random samples to merge into the query entity. The query-proportional algorithm uses an influence function to ensure its samples are mostly related to the query node. The hybrid algorithm mixes the aggressiveness of the target-fixed with the intelligent selecting of the source node found in the query proportional method.

After choosing the correct algorithm, a user needs to have a well trained model with several features. An advantage of using query-proportional techniques, because so little sampling is required, is that we can interactively test query accuracy. We can and also add context or keywords that were discovered from a previous run of the algorithm. This interactive querying workflow will help improve accuracy, which we experimentally verify in Section 5.

5. Experiments

In this section, we describe the implementation details, the data sets and our experimental setup. Next, we discuss our hypotheses and four corresponding experiments. We then finish with a discussion of the results.

Implementation We developed the algorithms described in Section 3 in Scala 2.9.1 using the Factorie package. Factorie is a toolkit for building imperatively defined factor graphs [10]. This framework allows a templated definition of the factor graph to avoid fully materializing the structure. The training algorithms are also developed using Factorie. The algorithms for canopy building and approximate string matching are developed as inside of PostgreSQL 9.1 and Greenplum 4.1 using SQL, PL/pgSQL and PL/Python. Inference is performed in-memory on an Intel Core i7 processors with 3.2GHz, 8 cores and 12GB of RAM. The approximate string matching on Greenplum is performed on a AMD Opteron 6272 32-core machine with 64 GB.

The parallel experiments were developed entirely in a parallel database, DataPath [1]. DataPath is installed on a 48-core machine with 256 GBs.

Data sets. The experiments use three data sets, the first is the English newswire articles from the Gigaword Corpus, we refer to this as the NYT Corpus [6]. The second is a smaller but fully-labeled Rexa data set. [3] Because it is fully-labeled it allows us to run the more detailed micro benchmarks. The NYT corpus contains 1,655,279 articles and 29,866,129 paragraphs from the years 1994 to 2006. We extracted a total of 71,433,375 mentions using the natural language toolkit named entity extraction parser [4]. Additionally, we compute general statistics about the corpus including the term and document frequency and tf-idf scores for all terms. We manually labeled mentions for each query over the NYT data set.

[3]http://cs.iit.edu/c̃ulotta/data/rexa.html

The second data set, Rexa, is citation data from a publication search engine named Rexa. This data set contains 2454 citations and 9399 authors of which 1972 are labeled. We perform experiments on the Rexa corpus because it is fully labeled unlike the NYT Corpus. The Rexa corpus is smaller in total size but it has average sized canopies.

The third data set is the Wikilinks Corpus [12] largest labeled corpus for entity resolution that we could find at the time of development. It contains 40 million mentions and 3 million entities that were extracted from the web and truthed based on web anchor links to Wikipedia pages. We loaded a million mentions onto DataPath to demonstrate the parallel capabilities.

5.1. Experiment Setup

Models. Features on the NYT and Wikilinks data sets were manually tuned and the features for the Rexa data set were trained using sample rank [17] with confidence weighted updates. We manually tune some of the weights in the NYT corpus to make up for the lack the complete training data.

Evaluation metrics. Convergence of MCMC algorithms is difficult to measure as describe in a review by Cowles and Carlin [5]. We estimate the convergence progress by calculating the $f1$ score of the query node's entity ($f1_q$). We create this new measure because we are primarily concerned with the query entity. Other measures include B^3 for entity resolution and several others for general MCMC models [2,5].

The query-specific $f1$ score is the harmonic mean of the query-specific recall R_q and query-specific precision P_q. To accurately determine the P_q and R_q of each query in this experiment we label each correct query node. Query-specific precision is defined as $P_q = \frac{|\{\text{relevant}(\mathcal{M})\} \cap \{\text{retrieved}(\mathcal{M})\}|}{|\{\text{retrieved}(\mathcal{M})\}|}$ and query-specific recall $R_q = \frac{|\{\text{relevant}(\mathcal{M})\} \cap \{\text{retrieved}(\mathcal{M})\}|}{|\{\text{relevant}(\mathcal{M})\}|}$. The $f1$ score for the query node's entity q is defined as:

$$f1_q = 2\frac{R_q P_q}{R_q + P_q}.$$

The $f1_q$ score is a good indicator of entity and answer quality. For multi-query experiments we calculate the average $f1_q$ scores for each query node. The run of each non-parallel algorithm is averaged over 3 to 10 runs.

5.2. Realtime Query-Driven ER Over NYT

In this experiment we show that query-driven entity resolution techniques allow us to obtain near realtime[4] results on large data sets such as the NYT corpus.

[4]We define realtime as only contributing a small loss when this process is part of an external execution pipeline such as an IE pipeline.

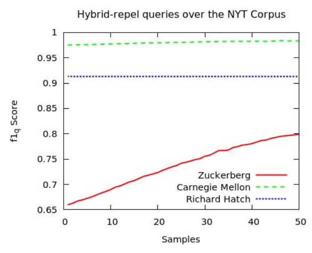


Figure 2: Hybrid-repel performance for the first 50 samples for three queries. Each result is averaged over 6 runs

Figure 2 shows the $f1_q$ score of the hybrid ER algorithms with three single-query ER queries. The graph shows performance over the first 50 proposals.

Recall, a canopy is first generated using an approximate match over the mention set. We use the repel inference function and all the mentions are initialized in one large entity. The 'Richard Hatch' and 'Carnegie Mellon' queries start at an $f1_q$ score of .92 and .97, respectively. The 'Zuckerberg' query starts above .65 and improves to an $f1_q$ score over .8.

These experiments show the repel method removing mismatches from the query entity. The co-influence function is used to quickly identify the mentions that do not belong in the entity and they are proposed to be removed. When a hybrid move is proposed, a mention from the large entity moved from a large entity group to a new, possibly empty, entity. This method relies on the good repulsion features and correct weights.

In Table 4 we show the performance of three queries. In addition to the query token we add four columns: blocking time in seconds, canopy size, inference time in seconds and the total compute time. Total time is the complete time taken by each run, this includes building of the influence data structure and result writing. The values in Table 4 show that fast performance of query-driven ER over a large database of mentions.

5.3 Single-query ER In this experiment we show a performance comparison between the single-query algorithms summarized in Sections 3 and 4. We run the query-driven algorithms over queries with different selectivity levels and show the accuracy over time. Each algorithm uses the attract method, so each mention in the canopy starts in its own entity.

Figure 3a shows the run time of all four algorithms on the Rexa data set with the query 'Nemo Semret', an author with a selectivity of 11. The performance for the baseline entity resolution does not get a correct proposal until about 500 seconds. The baseline algorithm takes a long time to accept the first proposal because it is randomly trying to insert mentions into an existing entity. Target-fixed immediately begins to make correct proposals. Hybrid and query-proportional have the best performance and resolve the entity almost instantaneously. The hybrid chooses the most likely nodes to merge into the query entity. As the first couple of proposals are correct merges, hybrid quickly converges to a high accuracy. Due to imperfect features, among the 10 averaged runs a few runs get stuck at local optimum and causing suboptimal results.

Figure 3b shows the run time of four algorithms for query node id 'A. A. Lazar' with selectivity of 46. The baseline algorithm progresses the slowest. The hybrid algorithm quickly reaches a perfect $f1_q$ score. Query-proportional algorithm lags slightly behind the hybrid method but still reaches a perfect value. The target-fixed algorithm gradually increases to a perfect $f1_q$ score about 60 seconds after hybrid and query-proportional.

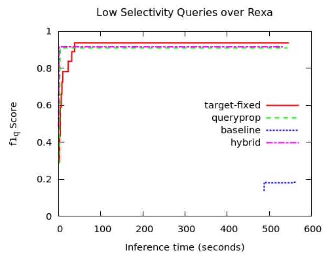
Figure 3c shows the run time of four algorithms with a query 'Michael Jordan' of selectivity 130. The baseline slowly increases over the 100 seconds. The hybrid algorithm again quickly achieves a perfect $f1_q$ score followed by query-proportional and then target-fixed. The time gap between each of the algorithms increases with the increase in selectivity, hybrid achieves the best performance.

We look deeper at how selectivity affects the rate of convergence. In Figure 4 we show the time it takes for each algorithm to reach an $f1_q$ score of 0.95 over increasing selectivity. We choose five query nodes of increasing selectivity but with the same canopy sizes. The hybrid algorithm runtime increased with the increase in selectivity but only slightly steeper than constant. Target-fixed increased for the first three queries but did not last more than 50 seconds. Query-proportional has only a slight increase in time till convergence for the first three queries. The highest two selectivity queries are expensive for query-proportional and we observe an exponential increase in runtime. These results are consistent with the exponentially large increase in the number of random comparisons needed to find a match for a query entity. The query-proportional algorithm does not focus on the query entity as aggressively as target-fixed and hybrid algorithms. Recall that the target-fixed and the hybrid algorithm focus on moving correct nodes into the query entity. Query-proportional selects candidate nodes using the influence function but it does not fix the target entity. With the target entity not fixed, the chance of correct node for the query entity decrease exponentially. This shows that selectivity of nodes affects the runtime performance of each algorithm. When performing join-driven ER it is important to take the relative selectivity of nodes into account for choosing best scheduling algorithms.

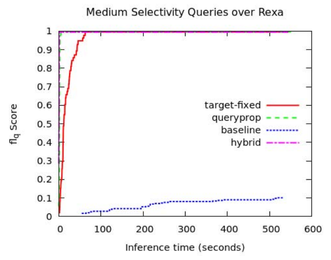
5.4. Multi-query ER In this experiment we study performance of our different scheduling algorithms for join-driven ER queries. We choose ten query nodes of different

Table 4: The performance of the hybrid-repel ER algorithm for queries over the NYT corpus for the first 50 samples. Total time includes the time to build the $\bar{\mathcal{I}}$ data structure and result output. The NYT Corpus contains over 71 million mentions, a large amount for the entity resolution problems.

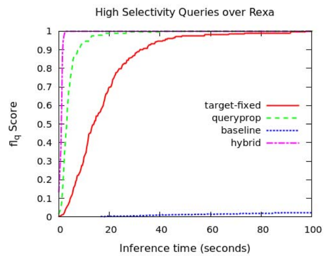
Query	Blocking	Mentions	Inference	Total time
Zuckerberg	24.4 s	103	2 s	37 s
Richard Hatch	28.3 s	226	18.5 s	59 s
Carnegie Mellon	25.9 s	1302	68 s	124 s



(a) A comparison of single-query algorithms on a query with selectivity of 11.



(b) A comparison of single-query algorithms with a query node of selectivity 46.



(c) A comparison of selection-driven algorithms with a query node of selectivity 130.

Figure 3: The results of each graph is averaged over three runs.

selectivity and run the join queries scheduling algorithms described in Section 3.2. Consider a list of query nodes with selectivities of {130, 63, 68, 7, 12, 12, 301, 11, 46}. The four algorithms, random, closest-first, farthest-first and selectivity-based are shown in Figure 5. The selectivity-based method out performs the other three algorithms in terms of convergence rate. The jumps in accuracy on the graph correspond to the scheduling algorithms choosing new query nodes and accepting new proposals. It has a high jump when it starts sampling the seventh, and highest selectivity nodes. The farthest-first algorithm rises the slowest out of the scheduling algorithms because it tries to stop sampling the high performing query entity and makes proposals for the slowest growing. Selectivity-based method performs well early because the high selectivity queries are sampled first. The high selectivity query makes up a large proportion of the total $f1_q$ score. The large jump in the random method is when it reaches the node with selectivity 301. Notice, closest-first reaches its peak $f1_q$ score the fastest because it tries to get the most out of every query node.

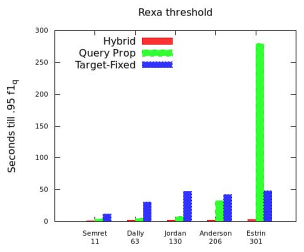


Figure 4: The time until an $f1_q$ score of 0.95 for five queries of increasing selectivities; averaged over three runs

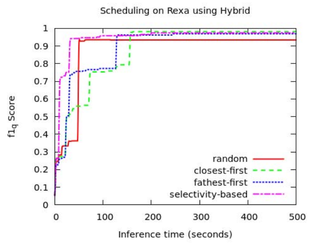


Figure 5: The progress of the hybrid algorithm across for multiple query nodes using difference scheduling algorithms. Each result is averaged over three runs

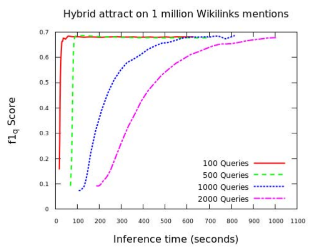


Figure 6: Hybrid-attract algorithm with random queries run over the Wikilinks corpus. Each plot starts after the Vose structures are constructed

5.5. Parallel Hybrid ER In this experiment has two objectives, first how does the hybrid algorithm perform in a canopy size of 1 million queries and what is the effect of increasing the number of queries nodes. In Figure 6 the Hybrid algorithm is able to resolve entities in a short amount of time. The creation time of the Vose structure is about linear in the number of queries. The ratio of queries to entities increases the performance benefit of the hybrid-attract method decreases. With more query nodes the construction time increases and the benefits of the algorithm decrease and become no better than the baseline method.

Experiment Summary Each of the query-driven methods outperform the baseline methods in terms of runtime while not losing out on accuracy. Across different data set sizes hybrid algorithms have the most consistent performance. If a system has a quality blocking function then it is better to use the co-influence entity resolution method. With multiple query nodes, selectivity-based is the most consistent performing algorithm. More accurate estimation of MCMC convergence performance could allow the dynamic scheduling algorithms closest-first and farthest-first to achieve higher accuracy. The more contextual information that can be added to query nodes at query time causes higher accuracy of the entity resolution algorithms. Parallel query-driven sampling is an effective way to get speed up in an ER data set when the ratio of mentions to entities is low.

6. Summary

In this paper, we propose new approaches for accelerating large-scale entity resolution in the common case that the user is interested in one or a watch list of entities. These techniques can be integrated into existing data processing pipelines or used as a tool for exploratory data analysis. We showed three single-node ER algorithms and three scheduling algorithms for multi-query ER and show experimentally how their runtime performance is several orders of magnitude better than the baseline. Further experiments and discussion is available in our extended write up [7].

An extended version of the paper is available here on arXiv: `http://arxiv.org/abs/1508.03116`.

Acknowledgements We would like to thank Kun Li and Alin Dobra for their help in the implementation of our prototype.

References

[1] S. Arumugam, A. Dobra, C. M. Jermaine, N. Pansare, and L. Perez. The datapath system: A data-centric analytic processing engine for large data warehouses. In *Proc. of the 2010 SIGMOD*, pages 519–530, NY, USA, 2010. ACM.

[2] A. Bagga and B. Baldwin. Entity-based cross-document coreferencing using the vector space model. In *17th ACL*, pages 79–85. ACL, 1998.

[3] I. Bhattacharya and L. Getoor. Collective entity resolution in relational data. *ACM Trans. KDD*, (1), 2007.

[4] S. Bird, E. Loper, and E. Klein. Natural language processing with python. In *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[5] M. Cowles and B. Carlin. Markov chain monte carlo convergence diagnostics: a comparative review. *Journal of AmStat*, 91(434):883–904, 1996.

[6] D. Graff. Ldc2007t07: English gigaword corpus, 2007.

[7] C. E. Grant, D. Z. Wang, and M. L. Wick. Query-driven sampling for collective entity resolution. *CoRR*, abs/1508.03116, 2015.

[8] D. Koller and N. Friedman. *Probabilistic graphical models: principles and techniques*. MIT press, 2009.

[9] J. Madhavan, S. Jeffery, S. Cohen, X. Dong, D. Ko, C. Yu, and A. Halevy. Web-scale data integration: You can only afford to pay as you go. In *Proc. of CIDR*, pages 342–350, 2007.

[10] A. McCallum, K. Schultz, and S. Singh. FACTORIE: Probabilistic programming via imperatively defined factor graphs. In *NIPS*, pages 1426–1427, 2009.

[11] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *ACL-HLT*, pages 793–803. ACL, 2011.

[12] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Wikilinks: A large-scale cross-document coreference corpus labeled via links to Wikipedia. Technical Report UM-CS-2012-015, 2012.

[13] M. D. Vose. A linear algorithm for generating random numbers with a given distribution. *IEEE Trans. Softw. Eng.*, 17(9):972–975, Sept. 1991.

[14] S. E. Whang, D. Menestrina, G. Koutrika, M. Theobald, and H. Garcia-Molina. Entity resolution with iterative blocking. In *2009 ACM SIGMOD*, pages 219–232. ACM, 2009.

[15] M. Wick, S. Singh, and A. McCallum. A discriminative hierarchical model for fast coreference at large scale. In *Proceedings of the 50th ACL*, ACL '12, pages 379–388, 2012.

[16] M. L. Wick and A. McCallum. Query-aware mcmc. In J. Shawe-Taylor, R. Zemel, P. Bartlett, F. Pereira, and K. Weinberger, editors, *Advances in NIPS 24*, pages 2564–2572, 2011.

[17] M. L. Wick, K. Rohanimanesh, K. Bellare, A. Culotta, A. McCallum, and A. McCallum. Sample rank: Training factor graphs with atomic gradients. In *ICML*, pages 777–784, 2011.