A Proposal Optimizer for Sampling-Based Entity Resolution

Christan Earl Grant* Daisy Zhe Wang†

Abstract

Sampling-based entity resolution is used to identify mentions in large text corpora. Sampling with MCMC techniques guarantees converges and can jump out of any local optimum. When performing sampling-based entity resolution over streams of data, two main issues arise. First, because the sampling process is random, many samples are waisted trying to resolve unambiguous entities. Second, the quadratic runtime for scoring entities makes sampling the largest entities prohibitive. In this paper, we discuss the creation a proposal optimizer, in the spirit of a database optimizers. We motivate the use of compression techniques to reduce the amount of processing when scoring large entities. We also discuss statistical early-stopping techniques for scoring entities. We describe our initial progress over a large entity resolution data set and how an optimizer can improve performance when processing entity resolution streams.

1 Introduction

Entity resolution across text corpora is the task of identifying mentions within the documents that correspond to the same real-world entities. To construct knowledge bases or extract accurate information, entity resolution (ER) is a required step. This task is a notoriously computationally difficult problem. Sampling-based er techniques, most using Markov Chain Monte Carlo techniques, trade some raw performance for a flexible representation and guaranteed convergence [4, 8, 11]. Processing streaming text documents exacerbates two of the core difficulties of ER. First, the computation of large entities and second, excessive computation spent resolving unambiguous entities. Optimization that touches these critical portions is wholly understudied. In this paper, we argue that compression and approximation techniques can efficiently decrease the runtime of traditional ER systems.

In sampling-based entity resolution, entities are represented as clusters of mentions. A random proposal is made to move a mention from a source entity of a destination entity. The state of the proposed state is score and if it improves the state, the new state is accepted. This process is repeated until the state converges. The process of scoring the state of the entity cluster, through pairwise feature computation of the mention in a cluster, is $O(n^2)$. For entity clusters larger than 1000 mentions, calculating the score for each proposal can become prohibitively expensive.

Wick et al. present an entity resolution system that uses a tree structure to organize related entities to reduce the amount of work performed in each step [11]. During each proposal, this approach avoids the pairwise comparison by restricting model calculation to the top nodes of the hierarchy. This approach can avoid massive amounts of computation with only a small book keeping cost. This tree structure is a type of *compression*.

Singh et al. present a method of efficiently sampling factors to reduce the amount of work performed when computing features [10]. They observe that many factors are redundant and do not need to be computed when computing the feature score. The use statistical techniques to guess the computed feature scores with a user-specified confidence. This approach can be categorized as early stopping for feature computation.

There is no one size fits all for these sampling algorithms [7]; Each of these methods above has drawbacks. Compression may slow down insertion speed and requires extra book keeping to keep to organize the data structure. Early stopping is not always precise and adding an extra conditions in the sampling metropolis hastings looping structure may slow down computation. However, applying each technique at appropriate time will accelerate the entity resolution process.

In this paper, we discuss our initial work towards the design an optimizer that modifies the sampling-based entity resolution process to improve sampling performance. The optimizer, in the spirit of a database query optimizer, examines the current state of each proposal and choose an execution plan. We trained classifier to decide when the sampling process should use early stopping. Additionally, we use training data to decide when is the best time for a particular entity to be compressed. This is done with negligible book keeping and a small over head. We make the following contributions

- We identify several techniques to speed up sampling past a natural baseline.
- We create rules and techniques for an optimizer to choose parameters and methods at run time.

*Datascience Research Lab, University of Florida; cgrant@cise.ufl.edu

†Datascience Research Lab, University of Florida; daisyw@cise.ufl.edu

- We empirically evaluate these methods over a large data set.

The outline of the paper is as follows. TODO

2 Background

Factor graphs are a pairwise formalism for expressing arbitrarily complex relationships between random variables [3]. A factor graph $\mathcal{F} = \langle \mathbf{x}, \psi \rangle$, contains a set of random variables $\mathbf{x} = \{x_i\}_1^n$ and factors $\boldsymbol{\psi} = \{\psi_i\}_1^m$. Random variables are connected to each other through factors. Factors represent a mapping of the relationship to a real valued number.

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:

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

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

Exact inference over complex factors graphs is computationally expensive becouse it involves computing the normalizing constant. Therefore, it is popular for researchers to use Markov Chain Monte Carlo (MCMC) approximation techniques to estimate the probability of settings. In particular, for large and dense factor graphs MCMC Metropolis Hastings has been shown to be a scalable and efficive technique for inference calculation [8].

Cross-Document entity resolution, resolving entities across document borders, is usually several orders of magnitude smaller when compared to within document entity resolution. In large text corpora, the sizes of entities follows the power law [9]. For example, Figure 1 is a generated data set containing 40 million mentions and 3 million entities over 11 million web pages. As documents and mentions are incrementally streamed through, the scale problem must becomes a critical issue.

The mentions on disk can be represented as a large array of identifiers. Entities are a collection of mentions and can be represented as such. In the worst case there is an equal number of entities and mentions. This means each mention is its own individual entity. In the other extreme, all the mentions may be a part of the same entity. For streaming entity resolution, mentions within documents (mention chains) must be matched to the existing set of entities [6]. In this paper, we assume the initial assignment problem is performed by searching for the most similar mentions and assigning the new mention to the same entity.

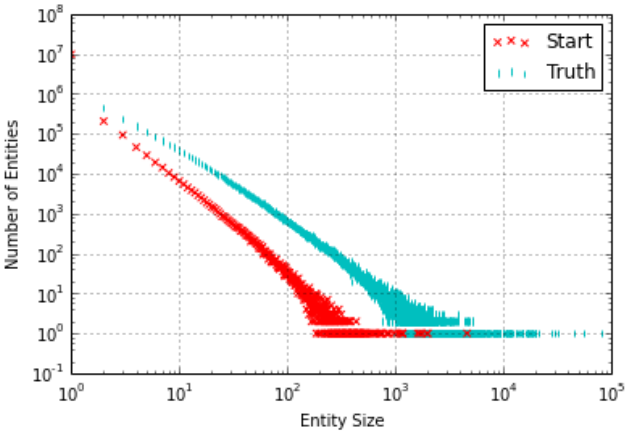

Figure 1: A distribution of entity sizes from the wikilinks corpus [9] with an initial start and the truth.

Doing pairwise comparison over clusters is $O(n^2)$. For clusters larger than 1000 mentions calculating scores of the model becomes extremely expensive. Performing sophisticated techniques over smaller clusters could also add extra over head. In this paper, we examine the trade-off of selecting techniques to accelerate the feature computation process.

3 Related Work

In this paper, we focus on sampling based ER over factor graphs. To compress and accelerate this data we observe two seminal methods for increasing sampling speed. First, is a method for approximating the score of factor counts and second is a compression method.

Singh et al. describe a method of approximating the factor computation during MCMC inference over factor graphs [8]. In this work, they note that sampling is usually inexpensive except in the case of many variable, or if scoring a sample is low. To this end they introduce Monte Carlo MCMC, this is a method of performing uniform and confident-based sampling over the computation of factors to avoid the full computation of features. This work shows significant speed up in a large-scale experiments.

Wick et al. perform hierarchal compression of entities during entity resolution [11]. This work defines a recursive data structure that allows succinct representations of millions of mentions. The hierarchical structure adds some book keeping to the sampling process but because large entity clusters are compress there are less comparison for each sampling iteration. This works displays orders of magnitude speedup when compared to a pairwise method.

Rao et al. create a system for streaming entity

resolution [6]. The use the doubling algorithm [1] to grow sets of clusters and a clever disk arrangement strategy [5] to retrieve growing entities on demand. However, this paper outlines no long-term strategy for managing growing entities. The same painpoints discussed early exist in this work.

4 Example

In this section, we discuss the acceleration in MCMC-MH sampling for entity resolution. We then motivate how we believe gains can be achieved given using compression, sampling acceleration methods and optimizers. We use a large real-world corpus to a motivating example.

The two issues we are investigating are as follows: First, given a source and destination entity (e_s, e_d) and the mention, how can we score the proposal in the *least amount of time*. Secondly, after the proposal is calculated, should we compress the entity structure? The optimizer will *decide* when to use each technique.

The total size of all entities in the traditional representation is:

$$\text{sizeof}(\mathcal{E}) = \sum_i c + (\text{sizeof(int)} * |e_i|), \tag{4.1}$$

where $sizeof$ is an abstract function to compute the size of the containing object, c is a class constant and $|e_i|$ is number of mentions in the entity.

There are many compression techniques, one being we only keep the mentions an entity that have a unique representation. That is, if any mention token is a duplicate, we remove it. This compressed total entity size size is:

$$\text{sizeof}(\mathcal{E}_{\text{compressed}}) = \sum_i c + (\text{sizeof}(int) * \#(e_i)), \tag{4.2}$$

where $\#(e_i)$ is the cardinality of the mention tokens in entity e_i. We note that when the $\#(e_i) \ll |e_i|$ it may be worth compressing this entity.

In Figure 1, we note that 45% percent of mentions are smaller that 100 mentions in size. Additionally, 82% percent of entities contain less than 1000 mentions. These number suggest that at times we we can take advantage of the redundancy within large entities and compress the entities. We investigate the Wiki links corpus further in Section 7.2.

In addition, Figure 1 shows that there is an order of magnitude difference between the sizes of initial entities and the true entity sizes. This difference gives us some intuition of the trends of the entity resolution process. The entities were initialized by exact string match, a common initialization scheme. This suggest that there are several unique representations of entities that during entity resolution entities sizes can expect to grow by an order of magnitude in size and the number of smaller entities will decrease. This is to be expected but we can use this property to track the growth and change of entity sizes over time to understand which camp a particular entity cluster belongs.

5 Algorithms

In this section, we will describe a simple algorithm for entity sampling and another simple compression technique. After introducing the compression and approximation techniques we discuss how an optimizer can be designed to improve the overall sampling time.

The baseline method performs pairwise comparisons by iterating over the mentions using the order on disk. The mentions ids are used to extract the contextual information of each mention from a database. This is the traditional method of computing the pairwise similarity of two clusters. This method results in simple code so modern compilers are able to perform extreme optimizations such as loop unrolling.

The confidence-base scoring method performs uniform samples of the mentions from the source and destination entities clusters. This method measures the confidence of the calculated pairwise samples and stops when the confidence of a score exceeds a threshold of 0.95. This is a simplified version of the sampling uniform sampling method described by Singh et al. [10].

The code to collect statistics is shown in Algorithm 2. The `add` function shows how and what statistics are recorded when each new mention is added. Notice `themax` and `themin` are variables in the Stats class that store the current maximum and minimum. The current sum, running mean are also updated with each new value added. The current implementation assumes the values from the pairwise factors follow a Gaussian distribution; the model in Singh et al. make the same assumption.

As entity sizes grow, we can expect to see many repeats of the same or very similar mentions. Reducing the entity size will shrink the effective memory footprint of entities. This is important for long running collection of entities. Run-length encoding is the simplest method for compressing entities. This method compresses the near duplicate mentions. A canonical mention is chosen along each exact duplicate and a counter map records the number of duplicates that are represented. For mention clusters with many duplicate the compression rates become large.

Technique	Compression	Early Stopping	Overhead
Baseline	No	No	None
Confidence-based [10]	No	Yes	Medium
Discriminative Tree [11]	Yes	No	Large
Run-Length Encoding	Yes	No	Small

Table 1: A table of the techniques to improve the sampling process and each is classified by how they affect sampling.

```
void Stats::add(long double x) {
  themax = MAX(themax,x);
  themin = MIN(themin,x);
  _sum += x;
  ++n;
  auto delta = x - mean;
  mean += (delta / n);
  M2 = M2 + delta * (x - mean);
}

double Stats::variance (void) const {
  if (n > 2)
  return M2 / (n-1);
  else
  return 0.0;
}
```

Figure 2: Sample code from the stats showing how running statistics are recorded and how the variance can be computed.

6 Optimizer

When before calculating the MCMC-MH proposal there are several decision we can make that will affect the runtime and accuracy of the algorithm. At each step we may: (1) approximate the calculation of the entity states; (2) update an entity structure to a compressed format; (3) skip the calculation of the proposal and directly accept or reject. These decisions can be made by observing several features of a source entity, destination entity and a source mention. We enumerate a small set of features that can yield information to help us decide how the entity structure should be changed.

The decision to compress an entity takes four main points into consideration. First, the time it takes to compress the entity (C_{time}). For example, if the time it takes to compress an entity is the same as the time it takes to reach an answer in the uncompressed format, then compression is superfluous. Second, it is important to consider the spaced saved in memory and the amount of additional entities that do not have to be fetched from disk and can now fit in memory (C_{space}). Third, we need to know how active an entity has been (C_{activity}). That is, how many additions or subtractions this entity has seen over a long period of time. This information is helpful in understanding the likelihood this entity will be requested for another addition or subtractions. Last, we retain the activity of an entity over a recent, short period of time (C_{velocity}). This information lets us know whether it is smart for this entity to take the time out to for compression while other mentions may be attempting an insertion or removal.

At each proposal step the decision made should maximize the *utility*. Utility of the decision is a numeric score to represent the gain performing the proposal calculation. The utility value is a real number ranged from $(-\infty, \infty)$. A formal model for utility is as follows:

$$U = C_{\text{time}} + C_{\text{space}} + C_{\text{activity}} + C_{\text{velocity}}$$

Collecting statistics to measure utility is can incur a significant overhead. Not every decision in the optimizer needs to be decided automatically. We can use some simple principles to estimate the utility at each point. In the next section we, examine an entity resolution data set and get some intuition for the development of the optimizer.

7 Implementation

In this section, we first describe the Wiki Link Data set we use for experiments. Next, we present a micro benchmark to validate our investigation of entity approximation and compression. We then discuss the implementation of the compression and approximation techniques over a large real-world cross-document coreference corpus.

7.1 Wiki Link Corpus The Wiki link corpus is the largest fully labeled cross-document coreference resolution data set to date [9]. When downloaded, the data set contains 40 million mentions and almost three million entities — it is a compressed 180 GBs of data. The wiki link corpus was created by crawling pages across the web and extracting anchor tags that referenced Wikipedia articles. Each page contains multiple multiple mentions of different types. The

Wikipedia articles act as the truth for each mention.

7.2 Micro benchmark To increase our intuition of early stopping techniques we simulated the MCMC proposal processes. We hypothesis that there is a clear range values where performing the baseline cluster sampling would be faster when compared to early stopping methods. We arrange entity clusters of increasing time and we compute the time (in clock ticks) each proposal takes to compute the arrangement of the clusters. The data in the clusters are distributed uniformly and for this experiment each cluster point was 5 dimensional. For the baseline cluster score computation we used a pairwise calculated of the average cosine distance with and without the mention. To compute early stopping we set a confidence threshold to 0.8 and the early stopping code stopped computation when the error predation was under 20%. There was no difference in the proposal choices between the baseline and the early sorting method.

The simulations were developed in `GNU C++11` and compiled with `g++ -O3`. The CPU was an 8 core Intel i7 with 3.2 GHz and 12 GBs of Memory. Each arrangement was run 5 times and results averages.

Early stopping or baseline. We first determine when early stopping approaches from proposal scoring is beneficial. For this result we compare the base like proposal evaluator with a confidence-based scorer for varying entity sizes. The result of this experiment is summarized in Figure 3. On the x-axis is the number of mentions in the source and destination cluster is for each proposal. The y-axsis is the number of clock ticks on a log scale.

We observe that for proposals with less than 100 and 1000 source and destination mentions, the performance of the baseline proposer is better than or almost equal to that of the more sorted early stopping method. For proposals that contain an entity cluster with 10000 mentions the early stopping method performs significantly better than the baseline method.

Surprisingly, the baseline proposals for for entities clusters containing $100K$ mentions performed over an order of magnitude better than the early stopping method.

The optimization found in predictable code paths make simple implementations like the baseline method attractive for small cluster sizes and very large clusters sizes. In addition, 82% of the entities in the truthed Wiki links data sets are less that 1000 mentions in size and 45% of the entities contain less than 100 mentions.

The results of the micro benchmark suggests that different proposal estimation techniques are useful at different times. Note that for these techniques a small constant amount of book keeping space is required to perform early stopping.

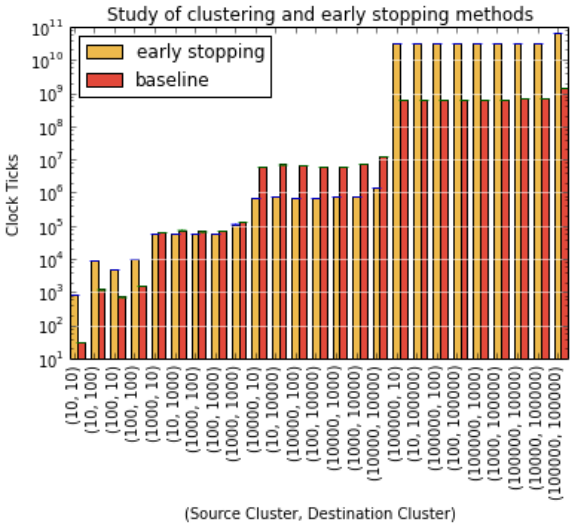

Figure 3: Comparison of baseline verses early stopping methods.

Insertion vs Compressions Time. Compressing an entity is an expensive operation. When compression and entity, this entity must be locked to prevent any concurrent access. In order to choose the best times to compress an entity cluster in this micro benchmark we look at the time to compression entity of different cardinalities and compare them to the time it takes to insert entities. Using a synthetic data set we generated entities of varying sizes and cardinality. This experiment is shown in Figure 4.

Cardinality number is a ratio of duplicates in the data set. For example, Cardinality 0.8 means 8 of 10 items in the data set are duplicates. The graph shows that in the time it take to compress entities of about 300K the sampler could make 100K samples. We can conclude from these result that compressing large entities is expensive should only be done if the cluster is prohibitively large and not popular.

Cardinality estimation for millions of entities is a significant overhead. Tracking cardinalities, in each entity, even using small probabilistic sketches such as Hyperloglog [2] become prohibitive at large sizes. By the time an entities cardinality needs to be monitored for possible compression, that entity might as well be compressed. We are continuing to look for lighter weight, cardinality estimators for millions of mentions.

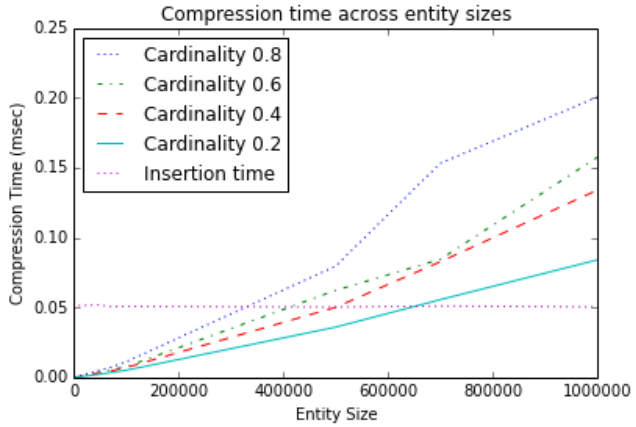

Figure 4: The time for compression for varying entity sizes and cardinalities.This is compared with line representing the time it take to make 100K insertions.

8 Summary

In this paper, we describe an initial approach for optimizing sampling for the entity resolution process. We begin to develop an optimizer that attacks two major limitations, the size of the entities and the redundant computation. This paper motivated the need for the optimizer and examined the feasibility of its treation. We plan to implement the full optimizer over a large, streaming corpus, with resolved entities. We hope to soon have a fully resolved TREC streamcorpus[1] and examine the performance of the optimizer of that large data set.

References

[1] M. Charikar, C. Chekuri, T. Feder, and R. Motwani. Incremental clustering and dynamic information retrieval. In *Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing*, STOC '97, pages 626–635, New York, NY, USA, 1997. ACM.

[2] P. Flajolet, É. Fusy, O. Gandouet, and F. Meunier. Hyperloglog: the analysis of a near-optimal cardinality estimation algorithm. *DMTCS Proceedings*, 0(1), 2008.

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

[4] A. Mccallum and B. Wellner. Toward conditional models of identity uncertainty with application to proper noun coreference. In *In NIPS*, pages 905–912. MIT Press, 2003.

[5] E. Omiecinski and P. Scheuermann. A global approach to record clustering and file reorganization. In *Proceedings of the 7th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '84, pages 201–219, Swinton, UK, UK, 1984. British Computer Society.

[6] D. Rao, P. McNamee, and M. Dredze. Streaming cross document entity coreference resolution. In *Proceedings of the 23rd International Conference on Computational Linguistics: Posters*, COLING '10, pages 1050–1058, Stroudsburg, PA, USA, 2010. Association for Computational Linguistics.

[7] D. Sculley and C. E. Brodley. Compression and machine learning: A new perspective on feature space vectors. In *Data Compression Conference, 2006. DCC 2006. Proceedings*, pages 332–341. IEEE, 2006.

[8] S. Singh, A. Subramanya, F. Pereira, and A. McCallum. Large-scale cross-document coreference using distributed inference and hierarchical models. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 793–803. Association for Computational Linguistics, 2011.

[9] 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, University of Massachusetts, Amherst, 2012.

[10] S. Singh, M. Wick, and A. McCallum. Monte carlo mcmc: efficient inference by approximate sampling. In *Proceedings of the 2012 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning*, pages 1104–1113. Association for Computational Linguistics, 2012.

[11] M. Wick, S. Singh, and A. McCallum. A discriminative hierarchical model for fast coreference at large scale. In *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics: Long Papers - Volume 1*, ACL '12, pages 379–388, Stroudsburg, PA, USA, 2012. Association for Computational Linguistics.

[1] `http://trec-kba.org/kba-stream-corpus-2014.shtml`