Entity Compression for Incremental Entity Resolution

Christan Earl Grant
University of Florida
Dept. of Computer Science
Gainesville, Florida, USA
cgrant@cise.ufl.edu

Daisy Zhe Wang
University of Florida
Dept. of Computer Science
Gainesville, Florida, USA
daisyw@cise.ufl.edu

ABSTRACT

1. INTRODUCTION

The storage of user generated content within systems has introduced vast amounts of data. To clean the dump an important task is entity resolution. Entity Resolution (ER) is the problem of resolving the records in a data set that correspond the same real world entity.

Entity resolution is a notoriously computationally difficult problem. Several efforts in different domains have made outstanding progress []. The main issues still affecting runtimes of ER systems are twofold, first, the computation of large entities and second, excessive computation spent resolving unambiguous entities. Optimization that touches these difficult points is wholly understudied. Amdahl's argument suggests that compression and approximation techniques can efficiently decrease the runtimes of traditional ER systems.

Some recently, researchers have suggest methods of compressing entities. Wick et al Heirchical ... Singh et all efficient facttoring http://people.cs.umass.edu/ sameer/files/mcmcmc-emnlp12-ppt.pdf

In this paper we propose aggressive compression methods to maximize resolution time. ...

We make the following contributions

In the paper we ...

2. BACKGROUND

Entity Resolution

Blocking

The distribution of entitiy sizes

3. RELATED WORK

Wick et al

Sameer Singh

Pay as you go ER

4. PROBLEM STATEMENT

In this section we formally define the compression problem for entity resolution.

We have two problems. First, Given an Entity $\mathcal{E}$ how can we create a a structure with minimal total size. Second, How can we incrementally add and remove items from the compressed structure. Third, how can we iterate over or sample from the compressed data structure to perform uncompression.

Show Distribution of Entity sizes in the data set

We can take advantage of the redundancy within large entities and compress the entities.

We look to perform add, remove and iterate operations over the compressed entity sets.

5. ALGORITHMS

In this section we will describe XXX algorithms for entity sampling. The main method we use to improve the computation speed is to reduce the number of comparisons we make for large entity clusters. We additionally discuss methods to improve the performance of algorithms over time by collecting statistics from the processes.

Naive Iterator. This method performs pairwise comparisons by iterating over the mentions using the order on disk. This is the traditional method of computing the pairwise similarity of two clusters.

SubSample Iterator. This 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 exceeds a threshold of 0.95.

Top-K SubSample Iterator. This method uses a priority queue to sort the mentions and only performs comparisons between the top M mentions.

Blocked Iterator. This method performs blocking on the mentions in the source and destination entities and takes the average gain from pairwise comparisons inside each block. Blocks are formed arbitrarily and are of fixed size. In database terms this method is a *block nested loop* comparison.

Blocked Subsample Iterator. This performs a block nested loop comparisom but it performes book keeping to only perform pairwise sampling until we reach a confidence threshold of 0.95.

Blocked Top-k Iterator. This method performs a block nested loop comparison except blocks are created by reading mentions from a priority queue.

Blocked Top-K Subsample Iterator. This method is a combination of the Block subsample iterator and the blocked top-k iterator.

Further, We examine active learning techniques to adjust threshold sizes based on statistics collected while running the algorithms.

We collect the following statistics from each training run:

Success and failure Data set size. Number of uniques tokens. Average pairwise score between mentions. Maximum pairwise score between mentions. Minimum pairwise score between mentions. Variance of the pairwise score between mentions. Mention Token tfidf scores. Cardinality of tokens in both entities. Generailze the feature explaination and move it to the implementation section — CEG

Using this information we train a decision tree classsifier to minimize the pairwise comparisons in the score function. The classifier is also constrained by the accuracy of the decision tree as approximate methods are less accurate. More formally . . .

The result of this classifier is two-fold. First, we have a classifier to choose the optimal algorithm for each proposal. Second, we have an active learning feed-back loop for algorithms with thresholds. We emperically study both of these outcomes.

6. IMPLEMENTATION

7. EVALUATION

8. CONCLUSION