Large Scale Entity Resolution of Quiz Bowl Questions

Tim Destan
University of Maryland, College Park
destan@cs.umd.edu

Abstract

Entity resolution is the task of determining which references in a data set refer to distinct real world entities. Many entity resolution techniques have been applied successfully to resolve references with predictable structure. We present a framework for performing entity resolution on clues from a quiz bowl competition to identify clues with the same answer. The clues are natural language text and have very little structure aside from being well formed English sentences. We explore two different blocking algorithms, canopies and iterative blocking, to facilitate scaling to large datasets.

1 Introduction

Entity resolution is the task of resolving a set of references to an underlying set of real-world entities. Because data is noisy, the references will often be ambiguous, with syntactically different references that refer to the same entity, or syntactically identical references in different contexts that refer to different entities.

In this paper, we apply entity resolution techniques to find ”duplicate” clues from a quiz bowl setting. Each clue has an associated answer, which is the name of some real-world entity (people, places, events, etc.). If we assume that each answer is relatively unambiguous, we can model this as an entity resolution problem where the answers are the entities, and the text of each clue (a short paragraph of natural language) is a ”reference” to that entity. We then formulate the problem as a clustering one, to find and group together all clues that reference a single answer (considering the answers to be unobserved). Because the set of possible answers is too large to practically enumerate (and no training data can possibly contain all of the potential answers), the clustering must be unsupervised as a matter of practicality. The data set used is described in more detail in Section 3.

The problem of entity resolution has been widely studied, with early approaches framing the problem as a pairwise matching problem. In one early paper (Fellegi and Sunter, 1969), the authors consider the problem of merging the records from two database files where there may be duplicates between the two files. They use a feature vector to represent each record and define a similarity function that quantifies the differences between two records. They then use these differences to classify each pair of records as a match or a non-match.

A recurring problem encountered in entity resolution is scaling the algorithms to handle large data sets. For any sufficiently large set of references, the problem of evaluating the pairwise distance function on every possible pair of references (which is quadratic in the number of references) is problematic.

Many previous approaches have made some effort to reduce the number of comparisons made by applying a preprocessing step to partition the set of references into subsets and then running their algorithm on each subset. Ideally, we would like to know that the quality of the results obtained by applying the entity resolution technique to each subset should be comparable to that of running the full pairwise entity resolution. This is obviously the case if the preprocessing step does not place matching references into different subsets. If the subsets may overlap, then the quality of solution may be preserved under less restrictive conditions, discussed in more detail later.

A few researchers have proposed meta-algorithms to formalize this preprocessing step. These meta-algorithms view the base entity resolution technique as a black box that takes as input a set of references and produces as output a clustering of those references into sets, where the references within each set refer to the same entity (the exact constraints placed on the underlying entity resolution technique vary). We choose two such techniques to evaluate: canopies and iterative blocking. The details of these algorithms are described in Section 2.

The data sets that entity resolution algorithms are typically evaluated on share many characteristics in common. Namely, the references are either structured or semi-structured, and the textual components of the references are quite short. Early approaches focused on de-duplicating references distinguishing a person, such as a customer database or census data (Winkler, 2006). Data sets of bibliographic citations are also commonly used (Cohen et al., 2003).

The census-style data sets in which each reference identifies a person make for useful test cases because the references are very structured. They have specific field names (e.g., first name, last name, address), and many common variations within these fields often follow predictable patterns (e.g. ”Street” shortened to ”St.”). The bibliographic citation databases can be less structured, as the references are textual strings. Conversely, the bibliographic citations are interesting because there are multiple related classes of entities (papers, authors, venues), and many sophisticated entity resolution techniques can take into account the relationships between these entities to help decide which references co-refer (Kalashnikov et al., 2005) (Bhattacharya and Getoor, 2007).

Where structure exists in data, it makes sense to exploit it to improve the quality of the results. However, it is also interesting to consider data sets that do not have an inherent structure, at least not the same degree of structure. Unlike a census entry or a bibliographic citation, the clues from the quiz bowl dataset we are using will not all share the same structure, other than being well-formed sentences in English that perhaps share some common patterns. Because of this, we will need to consider different feature representations than those commonly used when evaluating entity resolution algorithms. In Section 4, we formulate a feature representation and comparison mechanism for this dataset and demonstrate how to apply the aforementioned pre-partitioning schemes to references using this representation. Finally, in Section 5, we will evaluate the speed and effectiveness of the methods we have developed for this dataset.

2 Partitioning Methods Evaluated

We consider two methods of pre-partitioning the set of input references, described below.

2.1 Canopies

The canopies method uses a cheap distance metric to partition the set of input references into overlapping subsets called *canopies* (McCallum et al., 2000). Another, more expensive distance metric, namely the base entity resolution algorithm, is then used to cluster the references within each canopy, and transitive closure is performed on the results.

To make use of the canopies algorithm, two distance thresholds, T_1 and T_2 ($T_1 > T_2$), must be chosen, either by the user or by cross-validation. The algorithm begins with a list $\mathcal{L}$ containing all the references. Until this list is empty, choose a point $p \in \mathcal{L}$ and measure its distance (as computed by the cheap distance metric) to every other point in $\mathcal{L}$. Form a new canopy that includes all points $p' \in \mathcal{L}$ such that $d_{cheap}(p', p) \leq T_1$. Then remove from $\mathcal{L}$ all points p' such that $d_{cheap}(p', p) \leq T_2$. This allows overlap in the canopies because all points whose distance to p falls between T_2 and T_1 will be added to the canopy formed around p, but will remain in the list, eligible to be included in future canopies.

The authors show that under certain circumstances, the canopies method will perfectly preserve the results of the underlying entity resolution technique. In particular, they show that for K-means, Expectation-Maximization, and Greedy Agglomerative Clustering, provided that for every cluster produced by the technique without canopies, there exists some canopy containing all the elements of that cluster, then the clustering accuracy will be preserved. The authors do not, however, provide any way to determine whether this precondition holds for any given cheap distance function and data set.

Within each canopy, all references still must be compared, but with a well chosen cheap distance function, T_1, and T_2, the size of each individual canopy will be much smaller than the total set of references. Indeed, in the author's experiments, their canopies method improves the run time by an order of magnitude, while impacting accuracy only very slightly (and in some cases even improving it, unexpectedly).

2.2 Iterative Blocking

The second pre-processing method we consider is an iterative blocking method (Whang et al., 2009). The iterative blocking framework views the base entity resolution technique as a black box that transforms one partition of the references into another partition of references. The initial input is represented as a trivial partitioning where every reference is in its own partition. The interpretation of a partition, similar to the canopies approach, is that all references within a set in the partition refer to the same real world entity. To use the terminology of the paper, each set of "base records" within the partition becomes a "composite record" that has been judged to refer to the same entity.

Iterative blocking uses multiple blocking criteria to partition the input references. Each individual blocking criterion is some algorithm that (quickly) partitions the set of input records into subsets called blocks. The total set of blocks used by the algorithm is the union of all the blocks created by all the blocking criteria. Note that because there is no necessary relation between any of the various blocking criteria, there may be substantial overlap in the blocks.

The key feature of iterative blocking is that entity resolution results from one block are propagated to all the other blocks. Because of this, if the first block determines that references r_1 and r_3 refer to the same entity, any subsequent blocks that contain these records will learn of this and start with these records pre-merged.

The algorithm uses a queue to track the blocks to process. Until the queue is empty, it removes a block from the queue and runs the base entity resolution technique on that block. Each newly merged set of records from the output is propagated to any other blocks that contain any of the references that were merged in that result. Also, any such block is reinserted into the queue if it is not there already. There are many ways to choose the order in which blocks in the queue are processed, but the authors found that the best performing way is to track the number of "hits" each block receives and process the block with the most hits first. A block is hit whenever a base record contained in it is merged with other records.

The paper presents two implementations of iterative blocking: Lego, an algorithm that tracks the references each base reference is clustered with using an in-memory hash table, and Duplo, a scalable disk-based algorithm that tracks this same information in a log file. Duplo uses the same principles as Lego, but is intended for the use case in which the data set is so large that a hash that tracks all of it cannot be fit into main memory. For simplicity, we use Lego for the purposes of this evaluation.

3 Quiz Bowl Dataset

The data set used for this evaluation is taken from the context of a quiz bowl tournament. In such tournaments, two or more teams attempt to answer the same clue. A clue (or question) in this context is a short paragraph that gives a series of facts about the desired answer. A team may "buzz in" at any time, which stops the reading of the clue. At this point, the team that buzzed in must attempt to answer. If the answer given is correct, the team is granted points. If the answer is wrong, any opposing teams who have not yet attempted an answer for the current clue may listen as the remainder of the clue is read and attempt to answer it.

Note that this setup is different from Jeopardy, the TV quiz show that is the subject of IBM's Watson (Ferrucci et al., 2010). In Jeopardy, each clue is read in full before all contestants are given the chance to answer. In fact, buzzing in before the clue is finished is penalized. There is another difference that is more pertinent to our analysis. Jeopardy occasionally has clues that are difficult or impossible to answer without knowing their category (because all the clues in the category follow a theme). An ex-

One version of this quantity represents the sum of the predicted values of data. Other versions are given by the Bekenstein-Hawking formula and Sackur-Tetrode equation, which give this state variable for black holes and ideal gasses. It is proportional to a constant times the natural log of the number of microstates, the constant of proportionality being Boltzmann's; that formula appears on his grave. Introduced by Clausius, it is at a minimum when a system approaches absolute zero. For ten points, name this measure of disorder, that, for an isolated system, does not decrease with time according to the second law of thermodynamics.

Figure 1: An example clue from the quiz bowl dataset (Answer: Entropy).

ample could be a category called "Literary Settings" in which the clue only describes the work of literature, and only by knowing the name of the category does the player know that they are expected to answer with the setting. In contrast, each question in this data set is made to be understandable without any external information.

The answers to the clues in quiz bowl are well known entities such as historic events, scientific laws, literary characters, battles, etc. In particular, there are no open-ended or subjective questions (e.g. "What were the primary causes of the French Revolution?") that could have multiple correct answers. There may be multiple ways to refer to the entity designated by the answer, but there is never more than one acceptable answer for any clue.

3.1 Entity resolution formulation

One way to formulate the entity resolution problem for this dataset is to view the clues as references and the answers as entities. The problem then becomes one of finding the clues that refer to the same answer. The dataset includes a canonical answer for each clue, which we will not use in our clustering approach but will use to evaluate the accuracy of our approach. A small subset of the canonical answer labels were corrected manually because the clues clearly referred to different entities of the same name, so not correcting them would have penalized the framework for correctly placing such clues in different clusters. For example, there were clues that referred to both entropy, the statistical property, and "Entropy," the 1960 short story by Thomas Pynchon, and the labels for the canonical answers did not distinguish the two.

Although this is only one possible entity resolution formulation of many that could potentially be studied for this dataset, there are reasons why this particular one may be of interest. When selecting clues from a dataset like this for a tournament, it is important not to select the same clue twice. Also, one could attempt to create a question answering system that, given a new clue, constructs the feature representation for that clue and attempts to find one or more "duplicate" clues in the dataset and return the answer or answers from the duplicate(s).

These and other potential applications will not be explored directly but are described to provide motivation for why an efficient solution to this problem would be useful.

4 Implementation

To implement a system to cluster the quiz bowl data set, we first need to select a base entity resolution technique that will partition a set of clues into subsets, where the clues within each subset refer to the same entity. The algorithm must be unsupervised, as we do not assume that we have access to labeled training data. Additionally, the number of clusters is unknown, since we do not know the number of underlying entities. To clarify, the set of potential entities may be enumerable but is very large (This set would contain all well-known people, places, events, concepts, etc.). However, the subset of these entities actually referred to in our data will be quite small by comparison, and impossible to determine a priori.

We choose to use a greedy agglomerative clustering algorithm. Greedy agglomerative clustering begins with each reference in its own cluster. At each step, the algorithm greedily merges the two "closest" clues, as determined by some distance or similarity function. This continues until all clues have been merged into a single cluster or until some other stopping condition is reached. Obviously the former is not desirable, We will therefore need to provide such an alternative stopping criterion.

4.1 Feature representation

The choice of features used to represent each clue in the data set is of prime importance, as is the choice of the function used to compare these features. As the clues contain a large amount of text, it makes sense to have features for each of the words present in the clue, with stopwords removed. A natural way of comparing two clues using these features would be to construct an inverted index using the TF-IDF cosine similarity measure, treating each clue as a separate "document."

It is also possible to extract named entities from the clues using shallow parsing techniques. Named entities are useful, because clues referring to the same entity are likely to refer to the same named entities. For example, two clues whose answer is "Abraham Lincoln" are likely to both refer to the Emancipation Proclamation, the Civil War, John Wilkes Booth, etc. Conversely, if two clues have similar named entities, this could be evidence that the two clues refer to the same entity.

An observation about the clues leads to another potentially useful comparison. Clues frequently must refer to their underlying entities, but cannot do so by name. There are a number of ways they may do this, but one of the most common is via a noun phrase beginning with the word "this" (e.g. "this Italian painter"). Regular expression-based shallow parsing can be used to efficiently extract noun phrases from the clues, and from there, it is possible to collect the base nouns from those phrases that begin with "this."

This collection of nouns, which we call **referers**, is typically quite small for a given clue. A naive way to compare two clues using these referers may be to look for matching referers. But this coarse approximation could easily miss related words. For example, one clue could refer to Bertrand Russell as a "thinker," another calling him a "philosopher," and another calling him a "logician." We would like to score these clues as similar because, although the referers are not exact matches, they are highly related.

We can use the Jiang-Conrath similarity measure to quantify the semantic similarity between a single pair of nouns (Jiang and Conrath, 1997). This measure requires a semantic taxonomy of word senses, for which we use Wordnet (Miller, 1995). It also requires information content values, which were computed using the British National Corpus. Since the number of referers in each clue is small, we can simply compute the similarity measure between all possible senses of all the referers in the two clues, and choose the maximum similarity to represent the score.

Our score function is then a weighted combination of these three factors, and represents the similarity, rather than the distance, between two clues.

4.2 Canopies Implementation

To use the canopies method, we must choose our cheap distance metric, as well as our two thresholds. A TF-IDF cosine measure seems like the best choice for the cheap metric, since with it, we can measure the similarity from one clue to all other clues without having to spend time linear in the number of clues. A TF-IDF using all the words from the clues performed abysmally, so a TF-IDF using just the named entities was used instead.

Note that since the cosine scores from the TF-IDF measure similarity, not distance, the loose threshold T_1 actually must be *less* than T_2, since the description of these thresholds in Section 2 assumes a function that measures distances.

Our implementation sets T_1 equal to 0 and scales T_2 based on the size of the dataset. We experiment with various formulas for T_2.

4.3 Iterative Blocking Implementation

Like in the original paper, we will use minhash signatures on the sets of named entities extracted from the clues to determine the blocks. Some hash function h is computed for each element in the set, and the block for that set is chosen based on the minimum hash value among the elements in the set:

$$minhash(A) = \min_{a \in A} h(a)$$

With a minhash signature, the probability that sets A and B share a block is proportional to the Jaccard similarity of the two sets:

$$similarity_{Jaccard}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

Using a family of related hashes, we can generate as many hash functions, and thus as many blocking criteria, as we want.

It is possible to vary the approximate average size of the blocks generated by varying the length of the minhash signature used. If you only use a certain number of bits from the hash to determine which block to assign a clue to. The average block size will be inversely proportional to the number of bits used in the minhash signature. This should be fairly intuitive, if we assume that the hash function does a good job of separating the clues. One bit allows for at most two possible blocks per criterion, two bits allows up to four possible blocks, and so on.

In practice, it was necessary to add an additional blocking criterion using the categories from the data set. For each category, the iterative blocker looks at the named entities for each clue in that category and blocks as it normally would, but uses a smaller number of bits from the hash function to compute the minhash signature, since the fact that they are restricted to a single category already reduces the size of the blocks that will be generated.

5 Evaluation

The code discussed in Section 4 was written in the Python programming language using the Natural Language Toolkit[1]. All experiments were run on a desktop computer with a Intel Core 2 Quad Q6600 2.4 GHz processor with 8 gigabytes of RAM running 64 bit Windows 7. The Python 2.7 interpreter (CPython) was used to run the code, and the NLTK version used was 2.01.

For each method (the full pair-wise entity resolution solution, the canopies blocking, and the iterative blocking solution), we are interested in two things: the quality of the solution and the time taken. To measure the quality of the solution, we compare the clustering to the clustering created by grouping all clues with the same canonical answer together, called the "Gold Standard" clustering.

Given a Gold Standard clustering G and an experimentally generated clustering E, there are many competing metrics for evaluating the results (Menestrina et al., 2010). We choose to report the pairwise precision (the fraction of pairs present in E that are also in G) and pairwise recall (the fraction of pairs present in G that are also in E). Using these quantities, we can also calculate the pairwise F_1 measure:

$$F_{1,pairwise} = \frac{2 \cdot P_{pairwise} \cdot R_{pairwise}}{P_{pairwise} + R_{pairwise}}$$

Arguably, this measure captures the information we are interested in, and is easily interpretable. There are competing precision and recall measures that count the number of clusters in E present in G and vice versa, respectively. However, these seem inappropriate for this context. It is not so much the groups of clues themselves we care about so much as the fact that two given clues are related, so it seems incorrect to give no credit to a cluster that, for example, correctly groups 10 out of 11 related clues but misses the last one.

5.1 Clustering results

For greedy agglomerative clustering, we must select the method to use to measure the similarity between two clusters C_1 and C_2. Three choices could be:

- The distance between the closest 2 points in the 2 clusters:

$$\max_{c_1 \in C_1, c_2 \in C_2} similarity(c_1, c_2)$$

- The distance between the furthest 2 points in the 2 clusters:

$$\min_{c_1 \in C_1, c_2 \in C_2} similarity(c_1, c_2)$$

- The average distance between all pairs of points in the 2 clusters:

$$\frac{\sum_{c_1 \in C_1, c_2 \in C_2} similarity(c_1, c_2)}{|C_1 \times C_2|}$$

  where $C_1 \times C_2$ is the Cartesian product of the two sets.

To evaluate these methods, we tested all three methods (using the full entity resolution with no blocking) on a subset of the clues of size 250. The results are shown in Table 1. The average distance method achieves high precision and recall values at a feature similarity threshold of 2.0. Accordingly, throughout the other experiments, we use average distance and a feature similarity threshold of 2.0.

[1] http://nltk.org/

Method	Threshold	F_1	Prec.	Rec.
Closest	1.0	0.106	0.056	1.000
Closest	1.5	0.106	0.056	0.992
Closest	2.0	0.106	0.056	0.992
Closest	2.5	0.172	0.094	0.959
Closest	3.0	0.232	0.132	0.951
Closest	3.5	0.648	0.522	0.854
Average	1.0	0.106	0.056	1.000
Average	1.5	0.577	0.413	0.959
Average	2.0	0.954	0.990	0.921
Average	2.5	0.904	0.999	0.825
Average	3.0	0.808	0.998	0.678
Average	3.5	0.712	1.000	0.553
Furthest	1.0	0.106	0.056	1.000
Furthest	1.5	0.786	0.964	0.664
Furthest	2.0	0.786	0.964	0.664
Furthest	2.5	0.506	0.998	0.339
Furthest	3.0	0.506	0.998	0.339
Furthest	3.5	0.325	1.000	0.194

Table 1: Clustering method results for various thresholds.

5.2 Feature evaluation

In order to evaluate our choice of feature similarity function, we ran the full entity resolution using greedy agglomerative clustering, allowing it to run to completion, creating the entire dendrogram. Then, for each possible cutoff point, we took the resulting clusters and computed the pairwise F_1 score for that clustering, and report the maximum such score. This shows a kind of theoretical maximum F_1 for a greedy agglomerative clustering algorithm using the feature set being evaluated, assuming the algorithm had an oracle to tell it the best possible place to stop.

Table 2 shows the results of this experiment. Again, this was run on a subset of 250 clues. In the table, ”TF-IDF” denotes the cosine similarity between words in the clues, ”Named Entities” denotes the overlap in the sets of named entities, and ”Referers” denotes the semantic similarity between the nouns in noun phrases beginning with ”this” from the clues (see Section 4.1 for more details). We also include a simple baseline using the categories from the dataset that compares two clues only by determining whether or not they share the same category.

As expected, the category baseline did extremely poorly. Even alone, the named entities criterion achieves an impressive 0.9104 F_1 score. This appreciably outperformed the TF-IDF measure, which suggests that filtering down the words being considered to just named entities does provide a more precise criterion for comparison. The semantic similarity of referers performed the worst individually, which makes intuitive sense. Even if two clues refer to ”this inventor,” there could be many inventors, so we would expect this feature to identify many false matches without help from other features. The combination of all three criteria nets us an F_1 score of 97.31 for all three combined [2].

The most interesting result of this experiment is how the ”referers” feature interacts with the ”TF-IDF” feature. The combination of these two features actually results in a *lower* F_1 score than the ”TF-IDF” feature alone. If we looked at each combination of two features in isolation, this could lead us to believe that the ”referers” feature is always harmful to the overall result, but the F_1 score for the algorithm that incorporates all three actually does improve upon the one that only incorporates the other two features. Hence, all three features do contribute (although the comparison of named entities contributes the most), so all three are used throughout the other experiments.

Features Used	Maximum F_1
TF-IDF, Referers, Named entities	0.9731
TF-IDF, Named entities	0.9695
Referers, Named entities	0.9325
Named entities	0.9104
TF-IDF	0.6790
TF-IDF, Referers	0.4756
Referers	0.3831
Categories (baseline)	0.1057

Table 2: Maximum F_1 values for various feature combinations.

[2] It is worth mentioning that adding in the ”shares a category” feature from the baseline to this combination of the other three results in no additional improvement in F_1 measure.

5.3 Runtime Results

To test the scalability of each of the three algorithms (the full resolution, canopies, and iterative blocking), we ran each one on subsets of the full dataset of clues. The subsets were of sizes 10, 40, 100, 250, and 500. Each algorithm was run 4 times to obtain an average runtime. Only one blocking criterion was used for iterative blocking on top of the blocking by category. The results are shown in Figure 2.

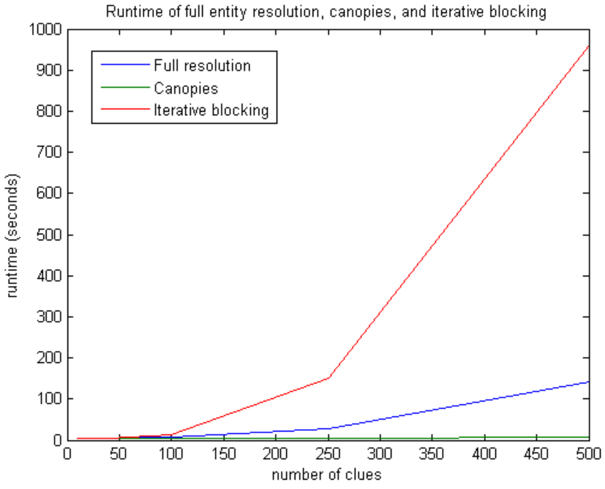

Figure 2: Runtime results.

The first thing to notice is how quickly the canopies clustering process runs. The total runtime is shown by the green line that is barely visible, and is always less than 10 seconds, even on 500 references.

The full resolution shows a quadratic increase in runtime with the size of the data set, as expected, but it is easy to miss this next to how poorly iterative blocking scales. Iterative blocking should be slower than canopies, as whenever two references are merged, any blocks that originally contained any of those references are reinserted into the queue to be processed again in light of the new information. This process is guaranteed to converge eventually, but this convergence can be somewhat slow.

Another potential reason why iterative blocking is so slow may be that we implemented the code assuming that the clusterer used by iterative blocking (or canopies) is a black box that merely provides the clustering. The iterative blocker may waste a substantial amount of computation time recomputing similarities between clusters that it could already know. If the greedy agglomerative clustering algorithms used on each block used globally unique identifiers to track their clusters, we could implement a caching mechanism to prevent some redundant recomputation.

5.4 Canopies Results

We evaluated three different plausible formulas for T_2. In each of them, T_2 is proportional to some function of the size of the data set, N. In all cases, T_2 is smaller for larger datasets. We tested one in which $T_2 = \frac{1}{N}$, designated ”Inverse.” Another has $T_2 = \frac{1}{log(N)}$, labeled as ”Inverse Log.” The third, labeled ”Inverse Sqrt,” has $T_2 = \frac{1}{sqrt(N)}$. The results are shown in Table 3.

Type	N	F_1	Precision	Recall
Inverse	10	1.000	1.000	1.000
Inverse Log	10	1.000	1.000	1.000
Inverse Sqrt	10	1.000	1.000	1.000
Inverse	40	1.000	1.000	1.000
Inverse Log	40	1.000	1.000	1.000
Inverse Sqrt	40	1.000	1.000	1.000
Inverse	100	0.903	1.000	0.824
Inverse Log	100	0.801	0.711	0.917
Inverse Sqrt	100	0.800	0.720	0.899
Inverse	250	0.782	0.939	0.670
Inverse Log	250	0.150	0.082	0.942
Inverse Sqrt	250	0.284	0.168	0.930
Inverse	500	0.681	0.748	0.624
Inverse Log	500	0.053	0.027	0.955
Inverse Sqrt	500	0.089	0.047	0.904

Table 3: Results for various threshold types for canopies clusterer.

For small datasets, the value for T_2 is inconsequential. However, we see that F_1 is maximized generally for the inverse relationship $\frac{1}{N}$. Hence, we use this value for T_2 in other experiments in which we test the canopies clusterer.

5.5 Iterative Blocking Results

The parameters that need to be tweaked for the Lego iterative blocking algorithm are the number of criteria and the number of bits used in the minhash signature. A larger minhash signature provides more but smaller blocks. A larger number of criteria increases the total number of blocks considered.

The results are shown in Table 4. These results used a subset of 250 clues. Since the hash functions used to create the blocks are randomly generated, each precision, recall, and F_1 value in the table is an average across 4 runs with the given number of criteria and minhash size. Each minhash was computed using the low order bits of a hash function, with the number of bits shown. Adding one bit should approximately double the number of blocks per criterion and approximately halve the average size of such blocks.

Criteria	Size (bits)	F_1	Precision	Recall
1	2	0.944	0.977	0.912
3	2	0.937	0.954	0.923
5	2	0.917	0.922	0.914
1	3	0.891	0.862	0.921
3	3	0.920	0.925	0.916
5	3	0.909	0.912	0.908
1	4	0.873	0.843	0.908
3	4	0.888	0.864	0.913
5	4	0.903	0.904	0.903

Table 4: Iterative blocking results, varying minhash signature size and number of criteria

Increasing the number of criteria tends to increase recall up to a point, then past that point degrades recall. 3 criteria maximizes the recall for every minhash signature length tested. Increasing the size of the minhash signature while holding the number of criteria constant generally degrades the recall, in all but one case (going from 2 bits to 3 in the case of 1 criterion). This makes perfect sense, because increasing the signature length makes for smaller blocks, which should increase the probability that pairs of clues that should be matched do not end up in a block together.

Increasing the number of criteria while holding the signature length constant decreases the precision for small minhash signatures, but increases the precision as the signatures grow larger. A plausible explanation for this could be that the precision is already maximized when the blocks are large, but as they get smaller, adding additional criteria can improve precision. Increasing the size of the minhash signature also degrades precision. The reason that smaller blocks result in more mistakes is not immediately obvious. However, recall that smaller blocks force the greedy agglomerative clusterer to make much more local decisions (it can only merge the closest clusters within the block, instead of globally). The falling off of precision could be seen as the penalty for making more bad local decisions.

The best overall F_1 scores are achieved with only one criterion and a low number of bits. However, using a low number of bits results in large blocks, which is at odds with our overall goal of scaling to large datasets. Note however, that the F_1 measure does not drop off dramatically even for large minhash sizes.

6 Future Work

There is still a large amount of work to do related to this problem. The quality of the solutions (as measured by F1 scores) is promising, but not so great that there is no room for improvement in the algorithm used. The feature representation and similarity function could still be customized further, possibly yielding better results.

Improving the runtime performance of iterative blocking is an obvious avenue of further research. The quality of its solutions was consistently better than those obtained using canopies, but the current run time is entirely too expensive a cost to pay. It is still unclear whether the iterative nature of the algorithm, some quirk of the implementation, or some unusual feature of this data set is responsible for its slow performance.

Conversely, while canopies has excellent runtime, the quality of the solutions it returns could be improved. Perhaps the strengths of both algorithms could be combined to somehow get the best of both worlds.

Assuming that these problems could be corrected, there are still usability improvements that could be made to our framework. The various parameters need to be finely tuned to produce good results. Finding a way to learn good values for these parameters in an unsupervised fashion would be quite useful (or more trivially and less ideally, using a small amount of labeled held-out data).

7 Conclusion

We have presented a framework for scaling entity resolution on datasets of quiz bowl clues, using canopies and iterative blocking as possible blocking algorithms. Much improvement can be made to the quality of the solutions from canopies and the runtime of iterative blocking, but the initial results are encouraging. Although we have not arrived at a definitive solution for a scalable entity resolution technique for large datasets that resemble this one, we have formulated a feature representation that results in quality solutions for the full resolution problem without blocking. Much progress was made even after the presentation, at the time of which, for example, neither iterative blocking nor canopies achieved even 0.50 F_1 for a data set of any size. Hopefully, the techniques outlined in this paper could provide a starting point for future exploration on entity resolution with more diverse datasets.

References

Indrajit Bhattacharya and Lise Getoor. 2007. Collective entity resolution in relational data. *ACM Transactions on Knowledge Discovery in Data*.

William W. Cohen, Pradeep Ravikumar, and Stephen E. Fienberg. 2003. A comparison of string distance metrics for matching names and records. *International Joint Conferences on Artificial Intelligence*.

Ivax P. Fellegi and Alan B. Sunter. 1969. A theory for record linkage. *Journal of the American Statistical Society*, 64:1183–1210.

David Ferrucci, Eric Brown, Jennifer Chu-Carroll, James Fan, David Gondek, Aditya A. Kalyanpur, Adam Lally, J. William Murdock, Eric Nyberg, John Prager, Nico Schlaefer, and Chris Welty. 2010. Building watson: An overview of the deepqa project. *AI Magazine*.

Jay J. Jiang and David W. Conrath. 1997. Semantic similarity based on corpus statistics and lexical taxonomy. *Proceedings of International Conference Research on Computational Linguistics*.

Dmitri V. Kalashnikov, Sharad Mehrotra, and Zhaoqi Chen. 2005. Exploiting relationships for domain-independent data cleaning. *SDM*.

Andrew McCallum, Kamal Nigam, and Kyle Ungar. 2000. Efficient clustering of high-dimensional data sets with application to reference matching. *KDD*.

David Menestrina, Steven Euijong Whang, and Hector Garcia-Molina. 2010. Evaluating entity resolution results. *Proceedings of the VLDB Endowment*.

George A. Miller. 1995. Wordnet: A lexical database for english. *Communications of the ACM*, 38:39–41.

Steven Euijong Whang, David Menestrina, Georgia Koutrika, Martin Theobald, and Hector Garcia-Molina. 2009. Entity resolution with iterative blocking. *SIGMOD*.

William E. Winkler. 2006. Automatically estimating record linkage false match rates. *Proceedings of the Section on Survey Research Methods, American Statistical Association*.