e-ISSN: 2582-5208
International Research Journal of Modernization in Engineering Technology and Science
(Peer-Reviewed, Open Access, Fully Refereed International Journal)
Volume:05/Issue:10/October-2023 Impact Factor- 7.868 www.irjmets.com

ENTITY RESOLUTION IN LARGE GRAPHS

Apurva Kumar*1, Shilpa Priyadarshini*2

*1Principal Software Engineer, Walmart Labs, Sunnyvale, CA, USA.

*2Principal Product Manager, Silicon Valley Bank, Santa Clara, CA, USA.

DOI : https://www.doi.org/10.56726/IRJMETS45298

ABSTRACT

Entity resolution is the process of linking data references that refer to the same real-world entity. It is a problem that occurs in many large-scale processes and applications. The ambiguity in references comes from various networks such as social networks, biological networks, citation graphs, and many others. Ambiguity in references not only leads to data redundancy but also inaccuracies in knowledge representation, extraction, and query processing. Entity resolution is the solution to this problem. There have been many approaches such as pair-wise similarity over attributes of references, a parallel approach for morphing the graph data on to a cluster of nodes (P-Swoosh) [2], and relational clustering that makes use of relational information in addition to the attribute similarity. In this paper, we make use of a relational clustering algorithm to resolve author name ambiguities in a subset of a real-world dataset: a US patent network consisting of more than 650,000 author references. We evaluated this algorithm using both attribute and neighborhood similarity and we achieved significantly high precision (~92%) if just rely on attribute similarity the precision is much lower (~67%). Thus, in large graphs to resolve references to real-world entities, using the neighborhood similarity in addition to attribute similarity leads to higher precision resolution.

Keywords: Entity Resolution, Name Disambiguation, Machine Learning, Relational Clustering, Large Network Graphs.

I. INTRODUCTION

Entity Resolution is a generic problem that currently is embedded in various networks such as biological network, spread across multiple social networks such as Facebook, TikTok, LinkedIn, Twitter (X.com), Instagram; even in search results such as Google, Bing and Yahoo! we can see multiple results for the same search topic. Other examples can be an author search in Google Scholar, DPLP, and numerous other citation networks. In all these applications, there are a variety of ways of referring to the same underlying object. Given a collection of objects, we want to a) determine the collection of 'true' underlying entities and b) correctly map the object references in the collection to these entities. This problem comes up is inherent throughout computer science. Examples include computer vision, where we need to figure out when regions in two different images refer to the same underlying object (also known as the correspondence problem); natural language processing when we would like to determine which noun phrases refer to the same underlying entity (co-reference resolution); and databases, where, when merging two databases or cleaning a database, we would like to determine when two records are referring to the same underlying individual (deduplication). In digital libraries, a big challenge is to automatically group bibliographic references that refer to the same publication. This problem is known as citation matching. Not only publications, but also authors, venues, etc. are often referred to by different representations and need to be disambiguated. For example, the dataset might describe publications written by two people, Johny Chao and Jane Chao, but refer to both of them as J. Chao, leading to ambiguity. This is a more general disambiguation challenge. It is also known as fuzzy grouping [10] and object consolidation.

To formalize entity resolution problem, let's consider a dataset D describes a set of entities E = e1, e2, ..., em and the relationships in which they participate. Entities can be of different types, such as publications, authors, search topics and venues etc. Entities in D are represented as a set of instantiated attributes R = r1, r2, ..., rn referred to as entity representations or references. The goal is to correctly group the representations in R that co-refer, that is, refer to the same entity. Entity resolution (ER) is the process of identifying and merging records judged to represent the same real-world entity.

In this paper we are trying to solve the author's name disambiguation problem in U.S. patent citation network [25] originally consisting of 3,774,768 author references. We extracted 651,877 author references from it that constitutes all patents filed by US based authors from 1975 to 1999 which only have first name and last name (301,877), as well as 350,000 references for authors which have middle name in addition to first and last name. We restricted our input to such numbers because the jobs were memory bound and can approximately handle these many references at a time on a single machine. To solve this problem, we considered numerous approaches as explained in Section II; Section II also explains the entity resolution as a graphical based approach and an unsupervised clustering algorithm [2] to solve it. Section III explains the dataset, experiments and results. We used graph based unsupervised relational clustering algorithm. This algorithm predicted good results as can be seen in Section IV.

II. METHODOLOGY

In this section, let's consider an example that illustrates collective resolution of author references. Figure 1 shows four paper references on similar topic say Topic modelling in machine learning, each with its own author references, with an optional institution attribute for each author. For instance, figure 1a shows paper P1 has two author references J Ullman and Andrew Ng. Andrew Ng has an attribute as institution and the value of the institution is Stanford. Figure 1b shows paper P2 with author references J.D. Ullman and Andrew Ng. Figure 1c shows four author references as K.Chen, Andrew Ng, J.D. Ullman, Di Mario. Each of the three authors except J.D. Ullman have their institution attribute as Stanford. Figure 1d shows the fourth paper which has three author references J.Dean, J.Ullman and Rajat Monga. Two of its authors J.Dean and J. Ullman has its institution attribute value Google.

In all we have seven unique author references that have five unique names - Andrew Ng, K. Chen, Di Mario, Rajat Monga and J.Dean, so all these five references can be reduced to five entities. Note that among these seven references, three references correspond to Andrew Ng, who after gathering evidence from figure 1a, 1b, 1c can be reduced to a single entity Andrew Ng with the value of institution attribute as Stanford. Out of eleven, four remaining references are J.Ullman (1a), J.D. Ullman (1b), J.D. Ullman (1c) and J. Ullman (1d). Just from Figure 1a and 1b, it is not clear if J. Ullman and J.D. Ullman are same entity or not. But since Andrew Ng wrote two papers P3 and P4 with same author J.D. Ullman, and also Andrew Ng and the other two authors K. Chen and Di Mario have same institution attribute value as Stanford in (c), J.D. Ullman seems to work with authors from Stanford. On the other hand, J. Ullman has worked on paper P4 with author references J. Dean, Rajat Monga who have same institution attribute value as Google. So based on the cumulative evidences from all figures 1a, 1b ,1c and 1d its clear from that the two ambiguous references J.Ullman and J.D.Ullman are not same but two distinct entities. Thus, entity resolution in this example made use of both attribute similarity and relational (neighborhood) similarity to resolve the author's name disambiguity.

2.1 Entity Resolution using Relationships

2.1.1 Problem

In this section we formally define the notation that we will be using to represent the entity resolution graphs to solve the entity resolution problem. In the entity resolution problem, we are given a set of references $R = \{r_i\}$, where each reference r has attributes $r.A_1, r.A_2, ..., r.A_k$. The references correspond to some set of unknown entities $E = \{e_i\}$. We introduce the notation r.E to refer to the entity to which reference r corresponds. The problem is to recover the hidden set of entities $E = e_i$ and the entity labels r.E for individual references given the observed attributes of the references. In addition to the attributes, we assume that the references are not observed independently, but that they co-occur. We describe the co-occurrence with a set of hyper-edges $H = \{h_i\}$.

Each hyper-edge h may have attributes as well, which we denote $h.A_1, h.A_2, ..., h.A_k$, and we use h.R to denote the set of references that it connects. A reference r can belong to zero or more hyper-edges and we use r.H to denote the set of hyper-edges in which r participates. In this paper, we only consider entity resolution when each reference is associated with zero or one hyper-edge, but in other domains it is possible for multiple hyper-edges to share references. For example, if we have paper, author and venue references, then a paper reference may be connected to multiple author references and also to a venue reference. Let us now illustrate how.

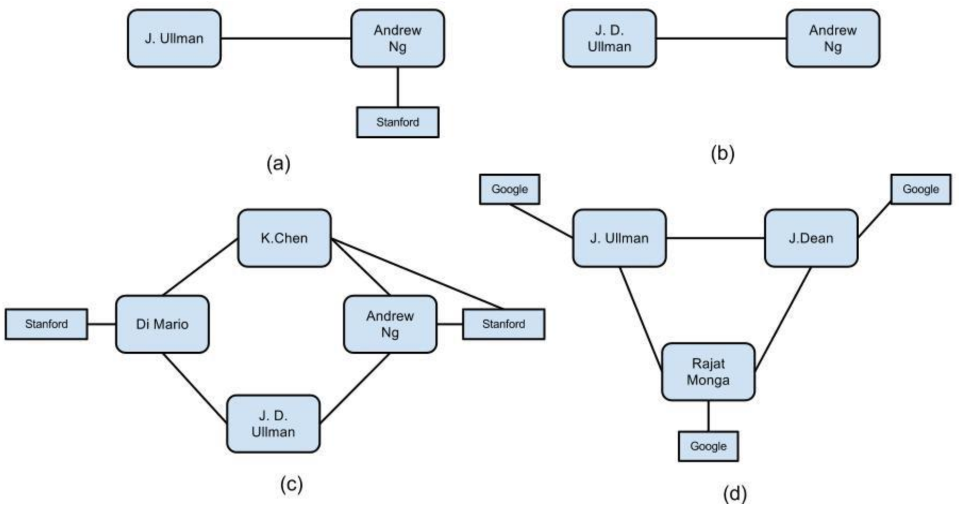

Figure 1. (a), (b), (c), (d) corresponds to four papers P1, P2, P3 and P4 on Topic Modelling.

Each paper has some author references, some are unique for e.g. K.Chen in (c), but some author references occur in more than one paper, for eg. Andrew Ng occurs in (a), (b) and (c), but he can be resolved to a single entity based on attribute similarity (Stanford) from (a) and (c) and relational similarity from (b) and (c), sharing co-author J.D.Ullman in both. Some author references are ambiguous, for instance from (a) and (b) it is not clear if J. Ullman and J.D. Ullman refer to same author. Only after gathering relational evidence from Andrew Ng and unique neighborhood match for J. D. Ulman in (c) and J. Ullman (d) can these two references be resolved to two distinct entities.

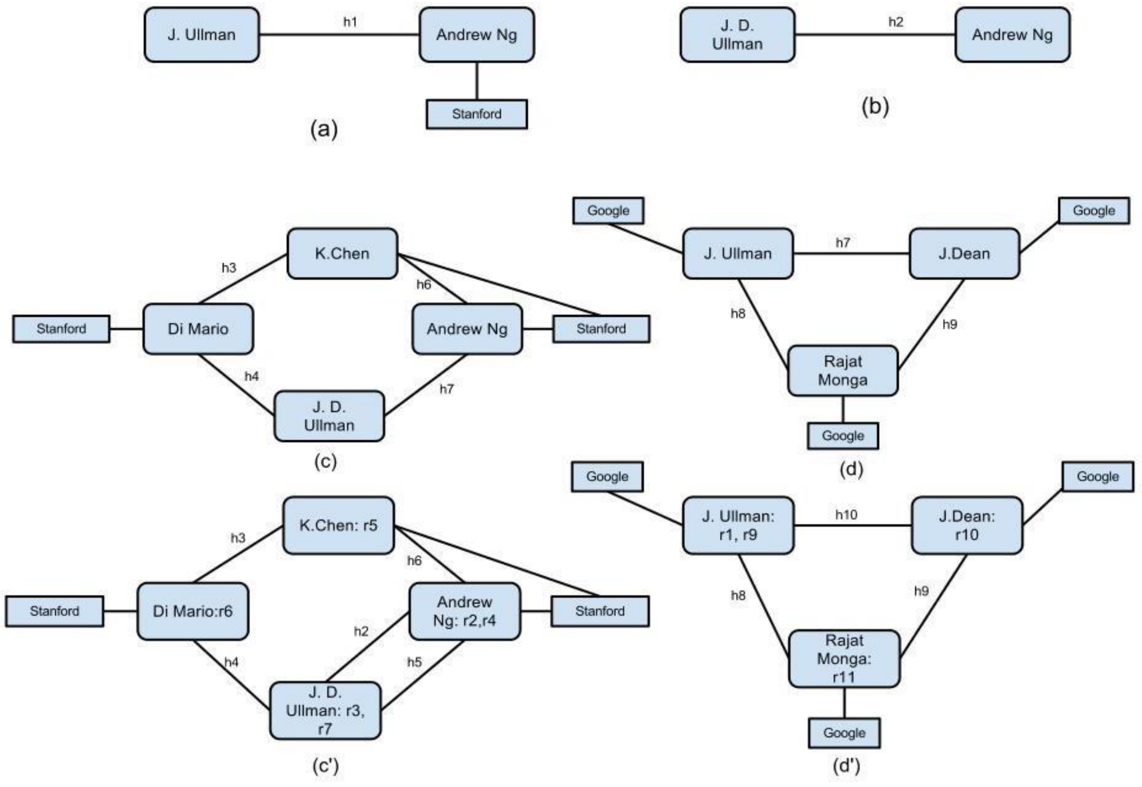

Figure 2. 2(a), (b), (c) and (d) are same as corresponding figures in 1.

Each edge between two author references have been labeled. Fig (c’) and (d’) represent the reduced entities of the author references in (a) - (d). Each node in (c’) and (d’) corresponds to the different reference labels that each author represented in the papers P1 - P4 and also an edge between two authors also contains edges from other references representing a hyper-edge. For eg. in fig (c’) Andrew Ng and J.D. Ullman have edges h2 and h5. our running example is represented in this notation.

Figure 2 shows the references and hyper-edges corresponding to Figure 1. Each observed author name corresponds to a reference, so there are eleven references r1 through r11. In this case, the names are the only attributes of the references, so for example r1.A is D. Ullman, r2.A is Andrew Ng and r3.A is J.D.Ullman, r5.A is K. Chen and so on. The set of true entities E = {Andrew Ng, K. Chen, Di Mario, Rajat Monga, J. Dean, J. Ullman, J.D. Ullman} as shown in Figure 2 (c , d). References r2 and r4 correspond to Andrew Ng, so that r2.E = r4.E = Andrew Ng. Similarly, r1.E = r9.E = J. Ullman and so on. There are also the hyper-edges H = {H1, H2, H3, H4}, where H1, = {h1}, H2 = {h2}, H3 = {h3, h4, h5, h6} and H4 = {h7, h8, h9}.

The attributes of the hyper-edges in this domain are the paper titles; for example, H1.A1 = "Unsupervised topic modeling". The references r2, r3, r4, r7 are associated with hyper-edge h2 and h5, since they are the observed author references (Andrew Ng and J.D. Ullman) in P2 and P3. This is represented as H1.R = {r1, r2}. Similarly, references also hold the values of the hyperedges they belong to. This process is done for all the references in a similar way.

2.1.2 Collective Relational Entity Resolution

The goal of collective relational entity resolution (CR) is to make resolutions using a dependency model so that one resolution decision affects other resolutions via hyper-edges. We now motivate entity resolution as a clustering problem and propose a relational clustering algorithm for collective relational entity resolution. Given any similarity measure between pairs of references, entity resolution can be posed as a clustering problem where the goal is to cluster the references so that only those that correspond to the same entity are assigned to the same cluster. We use a greedy agglomerative clustering algorithm, where at any stage, the current set C = {ci} of entity clusters rejects the current belief about the mapping of the references to entities. We use r.C to denote the current cluster label for a reference; references that have the same cluster label correspond to the same entity. So far, we have discussed similarity measures for references; for the clustering approach to entity resolution, we need to define similarities between clusters of references. For collective entity resolution, we define the similarity of two clusters c_i and c_j as:

$$sim(c_i, c_j) = (1 - \alpha) \times sim_A(c_i, c_j) + \alpha \times sim_R(c_i, c_j), \text{ where} 0 \leq \alpha \geq 1$$

where $sim_A()$ is the similarity of the attributes and $sim_R()$ is the relational similarity between the references in the two entity clusters. From this the equation, we can see that it reduces to attribute-based similarity for α = 0. Also, the relational aspect of the similarity measures includes the relational similarity measures the attribute similarity of the related references that are connected through hyper-edge and also the labels of related clusters that represent entities. This similarity is dynamic in nature, which is one of the most important and interesting aspects of the collective approach. For attribute-based and naive relational resolution, the similarity between two references is fixed [2].

In contrast, for collective resolution, the similarity of two clusters depends on the current cluster labels of their neighbors, and therefore changes as their labels are updated. The references in each cluster c are connected to other references via hyper-edges. For collective entity resolution, relational similarity considers the cluster labels of all these connected references. Recall that each reference r is associated with one or more hyper-edges in H. Therefore, the set of hyper-edges c.H that we need to consider for an entity cluster c is defined as

$$c.H = \cup\{h | h \in H \wedge r \in h.R\}, \text{ where } r \in R \wedge r.C = c$$

The hyper-edges connect c to other clusters. The relational similarity for two clusters needs to compare their connectivity patterns to other clusters. For any cluster c, the set of other clusters to which c is connected via its hyper edge set c.H form the neighborhood

Nbr(c) of cluster c:

$$Nbr(c) = \cup\{c_j | c_j = r.C\}, \text{ where } h \in c.H, r \in h.R$$

This defines the neighborhood as a set of related clusters, but the neighborhood can also be defined as a bag or multi-set, in which the multiplicity of the different neighboring clusters is preserved. We will use NbrB(ci) to denote the bag of neighboring clusters. In our example in Figure 2, the neighborhood of the cluster for J. Ullman consists of the clusters for Andrew Ng, J. Dean and Rajat Monga Note that we do not constrain the definition of the neighborhood of a cluster to exclude the cluster itself. For measuring the attribute similarity, we are using

Levenshtein distance [23], for neighborhood similarity, we are using Jaccard Coefficient with frequency [24] For the relational similarity between two clusters, we look for commonness in their neighborhoods.

2.2 Relational clustering algorithm

Given the similarity measure for a pair of reference clusters, we use a greedy agglomerative clustering algorithm [2] that finds the closest cluster pair at each iteration and merges them. Figure 3 is the pseudo-code for the algorithm that is implemented in detail in this section. We also discuss several important implementation and performance issues regarding relational clustering algorithm.

```
1.   Find similar references using blocking
2.   Initialize clusters using bootstrapping

3.   For clusters $c_i, c_j$ such that similar($c_i, c_j$)
4.       Insert $\langle sim(c_i, c_j), c_j, c_j \rangle$ into priority queue

5.   While priority queue not empty
6.       Extract $\langle sim(c_i, c_j), c_i, c_j \rangle$ from queue
7.       If $sim(c_i, c_j)$ less than threshold, then stop
8.       Merge $c_i$ and $c_j$ to new cluster $c_{ij}$
9.       Remove entries for $c_i$ and $c_j$ from queue
10.      For each cluster $c_k$ such that similar($c_{ij}, c_k$)
11.          Insert $\langle sim(c_{ij}, c_k), c_{ij}, c_k \rangle$ into queue
12.      For each cluster $c_n$ neighbor of $c_{ij}$
13.          For $c_k$ such that similar($c_k, c_n$)
14.              Update $sim(c_k, c_n)$ in queue
```

Figure 3. Pseudo-code for the relational clustering algorithm used in this paper.

2.2.1 Blocking

Since comparing all the references is costly O(n^2), unless the dataset is small, blocking techniques [Hernandez and Stolfo 1995; Monge and Elkan 1997; McCallum et al. 2000][2] are used to reduce the number of reference pairs that will result in a non-match. On a single machine with 4-8 GB of RAM, it is impractical to consider all possible pairs as potential candidates for merging. Apart from the scaling limitation, most pairs checked by an O(n^2) approach will be rejected since usually only about 1% of all pairs are true matches [2]. The blocking technique used is to separate references into possibly overlapping buckets and only pairs of references within each bucket are considered as potential matches. The relational clustering algorithm uses the blocking method as a black-box and any method that can quickly identify potential matches minimizing false negatives can be used. We use a variant of an algorithm proposed by McCallum et al. [2000] that we briefly describe below.

This algorithm just makes a single pass over the list of references and assigns them to buckets using an attribute similarity measure. To find the best potential bucket for a reference, each bucket has a representative reference that is the most similar to all references currently in the bucket. For assigning any reference, it is compared to the representative for each bucket. It is assigned to all buckets for which the similarity is above a threshold. If no similar bucket is found, a new bucket is created for this reference. A naive implementation yields a O(n(b + f)) algorithm for n references and b buckets and when a reference is assigned to at most f buckets. This can be improved by maintaining an inverted index over buckets. This implementation detail was adapted from the implementation details in [2]. For example, when dealing with names, for each character we maintain the list of buckets storing last names starting with that character. This helps in finding the right potential set of buckets in O(1) time for each reference leading to a O(nf) algorithm.

2.2.2 Bootstrapping

This phase of the relative clustering algorithm is an iterative loop that utilizes clustering decisions made in previous iterations to make new decisions. It is done by measuring the shared neighborhood for similar clusters. The problem here is that if we begin with each reference in a distinct cluster, then initially there are no shared neighbors for references that belong to different hyper-edges. So, the initial iterations of the algorithm have no relational evidence to depend on. Therefore, the relational component of the similarity between clusters would be zero and merges would occur based on attribute similarity alone. Many of such initial merges can be inaccurate, particularly for the references with ambiguous attribute values. To avoid this, we need to bootstrap the clustering algorithm such that each reference is not assigned to a distinct cluster. Specifically, if we are confident that some reference pair is co-referent, then they should be assigned to the same initial cluster.

However, precision is crucial for the bootstrap process, since our algorithm cannot undo any of these initial merge operations. In this subsection, we describe our bootstrapping scheme for relational clustering that makes use of the hyper-edges for improved bootstrap performance. The basic idea is very similar to the naive relational approach [2], with the difference that we use exact matches instead of similarity for attributes. To determine if any two references should be assigned to the same initial cluster, we first check if their attributes match exactly. For references with ambiguous attributes, we also check if the attributes of their related references match. In-depth coverage of this approach can be found below.

The bootstrap scheme goes over each reference pair that is potentially co-referent (as determined by blocking) and determines if it is a bootstrap candidate. First, consider the simple bootstrap scheme that looks only at the attributes of two references. References with ambiguous attribute values are assigned to distinct clusters. Any reference pair whose attribute values match and are not ambiguous is considered to be a bootstrap candidate. The problem with this simple approach that recall can be very poor for datasets with large ambiguous references if it assigns all references with ambiguous attributes to distinct clusters. When hyper-edges are available, they can be used as further evidence for bootstrapping ambiguous references.

A pair of ambiguous references form a bootstrap candidate if their hyper-edges match. Two hyper-edges h_1 and h_2 are said to have a k-exact-match if there are at least k pairs of references (r_i, r_j), $r_i \in H_1.R$, $r_j \in H_2.R$ with exactly matching attributes, i.e. $r_i.A = r_j.A$. Two references r_1 and r_2 are bootstrap candidates if any pair of their hyper-edges have a k-exact-match. Referring to our example from Figure 1 and 2, two references with name J. Ullman bootstrap candidate, will not be merged during bootstrapping on the basis of the name alone. However, if the first Ullman reference has co-authors 'A. Ng' and 'K. Chen', and the second Ullman has a coauthor 'A. Ng' in some paper, then they have a 1-exact-match and, depending on a threshold for k, they would be merged. The value of k for the hyper-edge test depends on the ambiguity of the domain. A higher value of k should be used for domains with high ambiguity. Other attributes of the references, and also of the hyper-edges, when available, can be used to further constrain bootstrap candidates.

Two references are considered only if these other attributes do not conflict. In the bibliographic domain, author references from two different papers can be merged only if their institutions and correspondence addresses match. After the bootstrap candidates are identified, the initial clusters are created using the union-find approach so that any two references that are bootstrap candidates are assigned to the same initial cluster. In addition to improving accuracy of the relational clustering algorithm, bootstrapping reduces execution time by significantly lowering the initial number of clusters without having to find the most similar cluster-pairs or perform expensive similarity computations.

2.2.3 Iterative cluster merge and evidence updates

Once the similar clusters have been identified and bootstrapping has been performed, the algorithm iteratively merges the most similar cluster pair and updates similarities until the similarity drops below some specified threshold. This is shown in lines 5-14 of the pseudo-code in Figure 3. The similarity update steps for related clusters in lines 12-14 are the key steps for the collective relational clustering. In order to perform the update steps efficiently, indexes need to maintained for each cluster.

In this section, we describe the data structure that which was used for this purpose. In addition to its list of references, we maintain three additional lists with each cluster. First, we maintain the list of similar clusters for each cluster. The second list keeps track of all neighboring clusters. Finally, we keep track of all the queue entries that involve this cluster. For a cluster that has a single reference r, the similar clusters are those that contain references in the same bucket as r, after blocking. Also, the neighbors for this cluster are the clusters containing references that share a hyper-edge with r. Then, as two clusters merge to form a new cluster, all of these lists can be constructed locally for the new cluster from those of its parents. All of the update operations from lines 9-14 can be performed efficiently using these lists. For example, updates for related clusters are done by first accessing the neighbor list and then traversing the similar list for each of them.

2.2.4 Time complexity

Having described each component of the relational clustering algorithm, in this section discuss about the running time of each component. First, we look at how the number of similarity computations required in lines 3-4 of Figure 5 is reduced by the blocking method. We first consider the worst-case scenario where the bootstrapping approach does not reduce the number of clusters at all. We need to compare every pair of references within each bucket. Suppose we have n references that are assigned to b buckets with each reference being assigned to at most f buckets. Then using an optimistic estimate, we have (nf/b) references in each bucket, leading to $O((nf/b)^2)$ comparisons per bucket and a total of $O((n^2f^2/b))$ comparisons. We have assumed that the number of buckets is proportional to the number of references, i.e., b is of the order of O(n). Additionally, assuming that f is a small constant independent of n, we have O(n) computations. It should be noted that this is not a worst-case analysis for the bucketing. A bad bucketing algorithm that assigns O(n) references to any bucket will lead to $O(n^2)$ comparisons. Now, let us consider the time taken by each iteration of the algorithm.

To analyze how many update or insert operations are required, we assume that for each bucket that is affected by a merge operation, so all the $O((nf/b)^2)$ computations need to be redone. Then we need to find out how many buckets may be affected by a merge operation. We say that two buckets are connected if any hyper-edge connects two references in the two buckets. Then if any bucket is connected to k other buckets, each merge operation leads to $O(k(nf/b)^2)$ update/insert operations. Note that this is still only O(k) operations when f is a constant independent of n and b is O(n). Using a binary-heap implementation for the priority queue, the extract-max and each insert and update operation take O(log q) time, where q is the number of entries in the queue. So the total cost of each iteration of the algorithm is O(k log q).

Next, we count the total number of iterations that our algorithm may require. In the worst case, the algorithm may have to exhaust the priority queue before the similarity falls below the threshold. So, we need to consider the number of merge operations that are required to exhaust a queue that has q entries. If the merge tree is perfectly balanced, then the size of each cluster is doubled by each merge operation and as few as O(log q) merges are required. However, in the worst case, the merge tree may be q deep requiring as many as O(q) merges. With each merge operation requiring O(k log q) time, the total cost of the iterative process is O(qk log q).

Finally, in order to put a bound on the priority queue's initial size q, we again consider the worst-case scenario where bootstrapping does not reduce the number of initial clusters. This results in $O(n^2f^2/b)$ entries in the queue as shown earlier. Since this is again O(n), hence the total cost of the algorithm can be bounded by O(nk log n). Let us consider the cost of bootstrapping. We can analyze the bootstrapping by considering it as a sequence of cluster merge operations that do not require any updates or inserts to the priority queue. Then the worst-case analysis of the number of iterations accounts for the bootstrapping as well. To compare this with the attribute and naive relational baselines, observe that they need to take a decision for each pair of references in a bucket. This leads to a worst-case analysis of O(n) using the same assumptions as before. However, each similarity computation is more expensive for the naive relational approach than the attribute-based approach because the former only requires a pair-wise match to be computed between the two hyper-edges.

III. MODELING AND ANALYSIS

We evaluated our relational entity resolution algorithm on a real-world dataset - U.S. patent records. This dataset is maintained by the National Bureau of Economic Research (NBER). The data set spans 37 years (January 1, 1963 to December 30, 1999), and includes all the utility patents granted during that period, totaling 3,923,922 patents (author references). The citation graph includes all citations made by patents granted between 1975 and 1999,1,803,511 nodes for which there is no information with NBER about their citations (they only have in-links to the records). The machine used for this experiment is a quad-core Intel(R) Xeon X3210, 2.13GHz machine with 4GB RAM.

3.1 Dataset

Since the relational clustering algorithm that we implemented is a sequential and not parallel, we took a fraction (1˜5% approx.) of the records for entire US patent dataset for the period 1975 to 1999.he total records D that we ran our algorithm was 651,877. D constitutes of two parts Dataset 1 (D1) and Dataset 2 (D2). D1 contains all the records for the authors who belong to U.S. and have no middle name i.e. they have first name, last name and other remaining attributes (such as address, country, company and author sequence number in the patent etc.). D1 holds 301,877 records. D2 holds the first 350,000 records for all the authors who belong to U.S. and they also have middle name in addition to first name, last name and all the remaining attributes. These numbers were selected by sampling records from D and then running clustering algorithm over it to see if they can successfully run on our machine. We also tried to run the entire 4 million records of D on our matching but it was making system run out of memory. The reason for running out of memory is because the binary heap used in bootstrapping phase is consuming O(n) memory; and hence the memory scales linearly with the number of records, thus putting an upper bound on the input dataset size.

3.2 Experiments

In this section we discuss about the different experiments we did to test our algorithm. The goal of first experiment was to disambiguate maximum number of records that can fit in machine's RAM. Since our system only had 4 GB of RAM, so running algorithm on D1 and D2 combined is not feasible as the dataset would not fit in the memory. To encounter it we made use of a statistical technique. We first ran the relational clustering algorithm on D1 and build the buckets for D1 using blocking technique (Section II) and then ran the algorithm on D2 using these clusters. Since all the assignments to the cluster during blocking occurs based on attribute similarity, the run on D2 on these clusters used for D1 can be used as added evidence to disambiguate D2. Let's discuss this approach in more detail: we first ran the clustering algorithm on D1, did the blocking and bootstrapping as explained in section II. It built the buckets where all the author references that are similar based on attribute similarity measures are clustered together into separate buckets; thus, running relational clustering on D1 builds evidences clusters for D1, then we ran the clustering algorithm on D2 using the clusters build for D1. In presence of more evidences i.e. we already have attribute similarity measures for the records in D1 and in presence of records that reach cluster of D2 based on attribute similarity we already have previous evidence and we can use that to disambiguate records in D2 using both already built attribute similarity and relational similarity during iterative phase (section II).

Thus, this statistical evidence-based reduction allowed us two advantages: Firstly, it allowed us to run a greater number of records that the memory can hold in the first place and also it provided us more evidence for the records in D2 to get better precision. The second experiment we ran was used to test the accuracy of the algorithm on a labeled dataset and validate the number of entities obtained by running the algorithm against the labeled count of entities known beforehand. We explain the experiment methodology and results in the following subsections:

IV. RESULTS AND DISCUSSION

The results and discussion may be combined into a common section or obtainable separately. They may also be broken into subsets with short, revealing captions. An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it. This section should be typed in character size 10pt Times New Roman.

4.1 Ambiguous names

For each run of relational clustering algorithm on D1 and D2, we computed the total number of authors that constitute the dataset and also the number of ambiguous names found. Dataset D1 had 92,585 total authors and 11,560 ambiguous names. On the other hand, dataset D2 had 140,589 total authors and 24,072 ambiguous names. Bar-chart in Figure 4 displays this information. Ambiguous names are a measure of the quality of a dataset. If the dataset is noisy, there will be high fraction for ambiguous names w.r.t total author names and vice versa for a good quality data set.

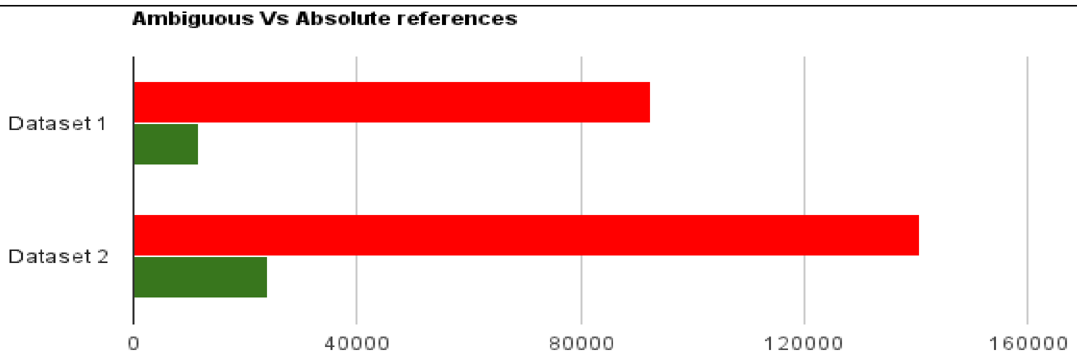

Figure 4. A bar chart showing the number of true names against number of ambiguous names found in dataset 1 and dataset 2 after blocking phase. Red bar represents the true names and green bar represents the ambiguous name.

4.2 Precision and Recall

Since the dataset that we are using is unlabeled, we used $_{\tau s}$successful pair-wise counts among all the pairs matched to get the precision. Figure 5 and 6 shows how precision varies as the algorithm iterates over all pairs during the pair matching process in the priority queue. This is an approximate measure as we do not have any prior information about the quality of the dataset. The initial true pair count τt is made at the end of blocking phase by setting it to the count of similar matches done at the end of blocking. Then as the iterative process begin, we compare the actual number of pair-wise similarity matches τa that were made. Precision P is then given as $P = {}^{\tau t}/_{\tau a}$. Note, because we are using unlabeled dataset, recall has been assumed to be one just to measure the accuracy of similarity measure of relational clustering process. Figure 5 shows the precision vs recall graph for the complete run of the algorithm on D1. The precision ranges from 0.918627 to 1. Figure 6 shows the precision vs recall graph for the run on D2. In this case the precision ranges from 0.663406 to 0.690025. The precision results are not so good for run on D2, suggesting that attribute similarity is not the best similarity measure to integrate evidences. This integration seems to have created lots of false positives match pairs and hence precision suffered as a result.

4.3 F1 score and α

Since α is a coefficient that tunes the relational clustering and decide the weight of attribute similarity and neighborhood similarity, we varied the values of α from 0.0 to 1.0 and measured F1 value for both datasets D1 and D2. Figure 7 shows how F1 score varies with α. Note that this measurement also makes the same assumption regarding the recall. As we can see D1 has higher F1 score for all the values of α as compared to D2. But F1 score for D2 increases as α value increase, suggesting that as α increases, relational similarity factor tends to increase (according to sim(ci, cj) and leading to increase in F1 score for D2 run. On the other hand, for D1, F1 drops slightly around 0.4 and then remain stable around that mark signifying lesser impact of α.

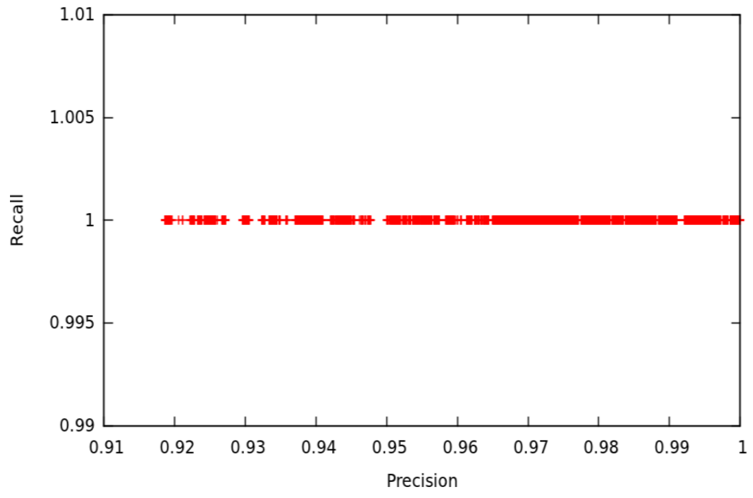

Figure 5. This figure shows precision vs recall in a run of relational clustering algorithm on dataset D1.

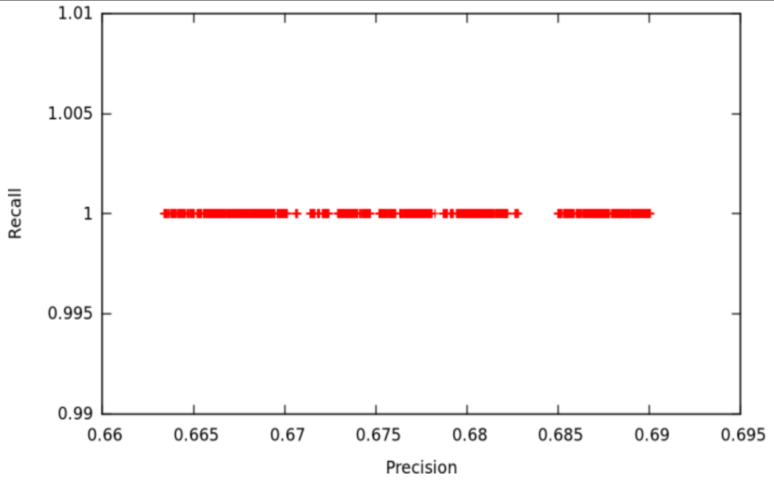

Figure 6. This figure shows precision vs recall in a run of relational clustering algorithm on dataset D2. Precision has dropped sharply on D2 because D2 was run on evidence clusters generated during run on D1, so attribute similarity-based matching has resulted in false records in each cluster that was reduced in D2.

4.4 CPU Usage

Figure 8 shows the running time of relational clustering algorithm against the total number of record references. It is evident that as the number of references increases the running time too increases in a slightly linear curve, confirming with the O(n log n) algorithm.

4.5 Memory usage

Figure 9 shows the peak memory usage of the algorithm against the total number of references. It's quite evident that as the number of references increase, memory scales linearly too, suggesting a linear time peak memory usage.

4.6 Validation

We tested the accuracy of the relational clustering algorithm on a selected set of labeled records from D2, say S. The author references in S were manually disambiguated using Google patents [26] and also using the attributes and relational similarity w.r.t. coauthors present in S. The size of S was 250 references which has in total 86 author entities. Some authors also had co-authors reference records in S. So, both attribute similarity and relational similarity were used to validate the clustering algorithm. Denote the total number of labeled author entities as T, number of entities identified by the algorithm as A and number of correctly identified entities as C respectively. We use precision P = C/A, recall R = C/T and F-score F1= (2PR)/(P+R) to measure the performance of the algorithm.

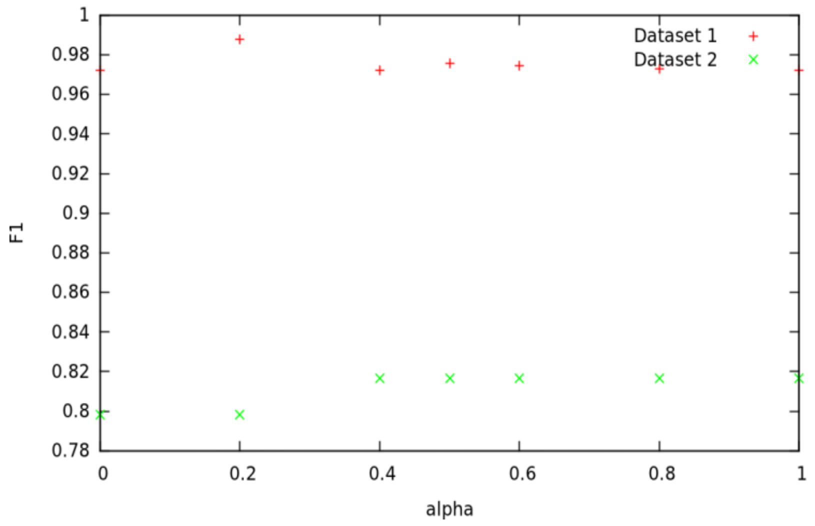

Figure 7. This figure shows F1 score versus our tuning parameter alpha on D1 and D2.

Because of lower precision for D2 (Figure 6), F1 score has dropped for D2 compared to D1 but shows improvement as we scale alpha from 0 to 1. D1 on the other hand seems to have dropped on 0.4 then shows steady behavior as alpha increases.

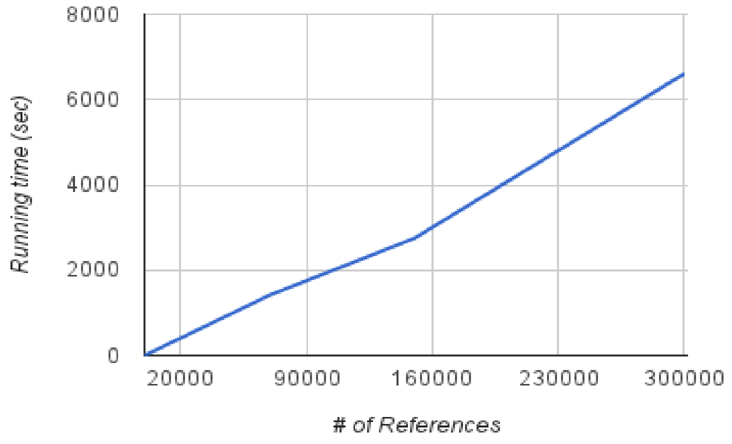

Figure 8. A CPU run time vs number of records, it can be seen that both scale slightly linear w.r.t. to each other, validating a O(nk log n) algorithm.

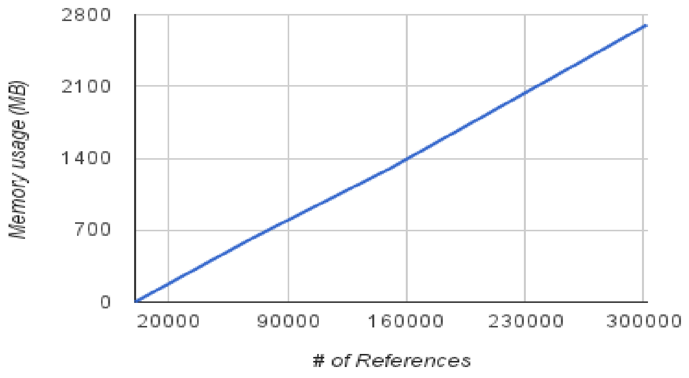

Figure 9. This shows the peak memory usage of the algorithm against the total number of references. It's can be seen that as the number of references increase, memory scales linearly too, suggesting a O(n) peak memory usage.

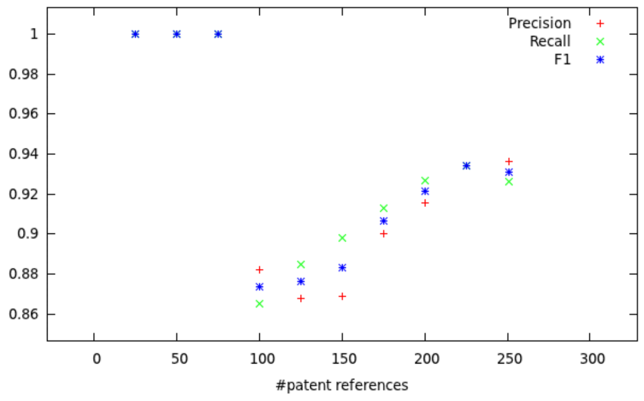

Figure 10. This graph shows the precision, recall and F1 score of the algorithm against top k labeled reference records, where k=25, 50, 75,...,250.

All these three parameters have high values initially, then there is a drop around k =75, suggesting lack of evidence for some sequence of author references leading to mispredictions, but as the dataset size increases and algorithm gathers more evidence, prediction, recall and F1 score keep improving. Performance and accuracy parameters where calculated for every run of the algorithm on top k records of S, where k = 25, 50, 75,. . . , 250. The precision, recall and F-1 measurements are shown in Figure 10. From these results, we can see that as the number of reference records increase and the more relational evidence merges with the existing evidence, the iterative algorithm does a good job over larger sets of top-k data and makes better predictions.

In this section we discuss the various performance trade-offs that were made and also discuss the error analysis done during the course of implementing and running the algorithm. The first optimization that was done was adding a strong evidence attribute field called assignee from the pat63 99.txt file [25] to the patent references. This attribute was given a higher value as compared to other fields such as zipcode or state. The results obtained from this tweak suggest in favor of this step. Second performance efficiency step, that helped during blocking phase was using a normalized author name (lname fname) or (lname fname mname) as patent records if are pre-processed by sorting on the basis of normalized name will lead to faster blocking as same author names can create pairs quickly. Another technique was to measure approximate precision was made in Section II, by setting recall to one, we just wanted to check how precision varied if we only trust that bootstrapping alone can create good potential pairs. Figure 5 shows good results alone for this approximate precision measurement experiment. Some other lessons learned was that rather than relying on a purely deterministic approach of running the entire datasets, it's a good idea to reduce the dataset, and use statistical approach to build evidence for doing next iteration on new data. This approach is useful for sequential algorithms like relational clustering, for which entire dataset cannot fit in main memory.

V. CONCLUSION

In this paper we implemented a relational clustering algorithm [2] to disambiguate author references in a real-world patent citation dataset. To get good results, our similarity function made use of both attribute similarity as well as neighborhood similarity. Two different types of experiments were performed to validate the efficiency of the algorithm. The first method was a statistical approach to add more evidence to a previous efficient run of the algorithm, but since our blocking technique relied on attribute similarity, second run led to lots of false negatives and hence lead to moderate precision in the iterative phase. The second experiment made use of a manually labeled dataset and the results obtained validated the efficiency of the algorithm. To summarize, to resolve data references in large graphs such as social networks, biological networks, web corpus etc. it is important to use both attribute and neighborhood similarity in order to resolve entities to their real-world ones.

VI. REFERENCES

[1] Lisa Getoor, Indrajit Bhattacharya, Entity resolution in graphs, Mining Graph Data, Wiley publication, Chapter 13.

[2] Lisa Getoor, Indrajit Bhattacharya, Collective entity resolution in relational data, ACM Transactions on Knowledge Discovery from Data (TKDD), Volume 1 Issue 1, March 2007.

[3] Kawai, H., Garcia-Molina, H., Benjelloun, O., Menestrina, D., Whang, E., Gong, H.: PSwoosh: Parallel algorithm for generic entity resolution. Tech. Rep. 2006-19, Department of Computer Science, Stanford University (2006)

[4] Vetle I. Torvik, Neil R. Smalheiser. Author name disambiguation in MEDLINE. ACM Transactions on Knowledge Discovery from Data (TKDD) Volume 3 Issue 3, July 2009.

[5] MARostami, ASaeedi, E Peukert, E Rahm, Interactive Visualization of Large Similarity Graphs and Entity Resolution Clusters. EDBT, 2018 SNAP http://snap.stanford.edu/data/cit-Patents.html

[6] I. Fellegi and A. Sunter. A theory for record linkage. Journal of Amer. Statistical Association 1969.

[7] M. Hernandez and S. Stolfo. The merge/purge problem for large databases. In SIGMOD, 1995.

[8] R. Ananthakrishna, S. Chaudhuri, and V. Ganti. Eliminating fuzzy duplicates in data warehouses. In VLDB, 2002.

[9] I. Bhattacharya and L. Getoor. Iterative record linkage for cleaning and integration. In DMKD Workshop, 2004.

[10] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In SIGMOD, 2005

[11] D. V. Kalashnikov, S. Mehrotra, and Z. Chen. Exploiting relationships for domain-independent data cleaning. In SIAM Data Mining (SDM), 2005.

[12] B. Malin. Unsupervised name disambiguation via social network similarity. In Workshop on Link Analysis, Counterterrorism, and Security, 2005.

[13] Z. Chen, D.V. Kalashnikov and S. Mehrotra, Adaptive Graphical Approach to Entity Resolution, Proc. ACM IEEE Joint Conf. Digital Libraries (JCDL), 2007.

[14] I. G. Councill, H. Li, Z. Zhuang, S. Debnath, L. Bolelli, W. C. Lee, A. Sivasubramaniam, and C. L. Giles. Learning metadata from the evidence in an on-line citation matching scheme. In JCDL, 2006.

[15] X. Dong, A. Y. Halevy, and J. Madhavan. Reference reconciliation in complex information spaces. In SIGMOD, 2005

[16] H. Han, L. Giles, H. Zha, C. Li, and K. Tsioutsiouliklis. Two supervised learning approaches for name disambiguation in author citations. In JCDL, 2004.

[17] A. McCallum and B. Wellner. Conditional models of identity uncertainty with application to noun coreference. In NIPS, 2004.

[18] H. Pasula, B. Marthi, B. Milch, S. Russell, and I. Shpitser. Identity uncertainty and citation matching. In NIPS, 2002.

[19] P. Singla and P. Domingos. Multi-relational record linkage. In MRDM Workshop, 2004.

[20] R. Bekkerman and A. McCallum. Disambiguating web appearances of people in a social network. In WWW, 2005.

[21] Z. Chen, D.V. Kalashnikov and S. Mehrotra, Adaptive Graphical Approach to Entity Resolution, Proc. ACM IEEE Joint Conf. Digital Libraries (JCDL), 2007.

[22] Levenshtein distance. http://en.wikipedia.org/wiki/Levenshtein distance

[23] Jaccard index.http://en.wikipedia.org/wiki/Jaccard index

[24] The NBER U.S. Patent Citations Data File: Lessons, Insights, and Methodological Tools.
http//data.nber.org/patents/Google Patents. www.google.com/patents.