Vertex Deduplication Based on String Similarity and Community Membership

Ryan McConville, Weiru Liu and Jun Hong

Abstract Entity resolution is a challenging problem with unresolved and duplicated entities common in many large real world datasets. New methods are required for addressing this problem as the use of graphs to model data continues to proliferate. In this paper we propose a general framework for the fast resolution of duplicate vertices in graphs. Our framework utilises locality sensitive hashing for the quick identification of potential duplicates based on string similarity. However it is clear that in many tasks string similarity alone is not enough to determine duplication. This motivates the second aspect of our method which discovers the community structure in the graph using an ensemble of community detection algorithms. These communities are then used as to augment the string similarity in the deduplication process. We evaluate our approach on a real world graph consisting of 620885 vertices and 1129986 edges and report a high accuracy score on a commercial real world graph.

1 Introduction

Entity Resolution is known by many names ranging from record linkage to data matching and deduplication. However, they all share a common goal which is to link or group different manifestations of the same real world entity in a dataset. Examples of this problem are plenty, such as the existence of different names for the same object (e.g., airplane, aeroplane, plane), or errors resulting from data entry (e.g. typos) and abbreviation and formatting differences (e.g. New York, NY, N. York).

Ryan McConville
University of Bristol, UK, e-mail: ryan.mcconville@bristol.ac.uk

Weiru Liu
University of Bristol, UK e-mail: weiru.liu@bristol.ac.uk

Jun Hong
University of the West of England, UK e-mail: Jun.Hong@uwe.ac.uk

Datasets found in the real world, unsurprisingly, tend to suffer from these problems. Furthermore, since the onset of the 'Big Data' era there is a drive to collect and store as much information as possible, the entity resolution problem becomes even more important as the increase in amount of data typically correlates with an increase in data quality issues. In addition to the increase in the amount of data being generated one must also consider that much of this data is being generated by separate sources and processes which raises challenges, for e.g. in record linkage. As a result one can not always proceed with the analysis or data mining without attempting to improve the data quality. Data collection, particularly when it is carried out by humans, is a messy process which introduces missing, ambiguous and incomplete values. Further it is typical that the data may not be well structured.

Much attention previously in entity resolution has focused on traditional databases but increasingly data is being modeled as networks and graphs meaning previous solutions may no longer be applicable. The field of graph mining has opened many opportunities for discovering knowledge from the connections between objects. For example, users within a social network are not just represented via their own profile, but also by whom they are connected to via friendship, etc. If the aim is to detect duplicate users within the social network it is not enough to just analyse names as they do not uniquely identify individuals, and vice versa. However, if one also takes into account the structural information of each user in the graph this can improve the entity resolution process. While it is possible to attempt resolving each entity by examining only each individual vertex, graph data provides extra information in the form of the edges which connect vertices together, and explicitly models the relationship between different objects.

To clarify, in this work, when a reference is made to resolving entities, it refers to detecting two entities that are duplicates of each other. That is, given two objects o_i and o_j that are dissimilar in some way, but actually refer to the same entity, the objective is to resolve them to a single object.

As typical datasets become larger it becomes important to consider scalability and efficiency. The social network example above may involve hundreds of millions of vertices and billions of edges. A naive approach to entity resolution would compute the pairwise similarity of all object pairs in the dataset. This has an unsatisfactory complexity of $O(n^2)$ where n is the number of objects. There have been a number of approaches in the literature proposed specifically to reduce the time complexity. A popular approach is the use of 'blocking' techniques [17]. These techniques are based on the fast assignment of objects to blocks, where each block consists of potentially similar objects, and each block is relatively small. By only performing the pairwise comparisons to objects within the same block, the number of pairwise comparisons is reduced from $O(n^2)$ to $O(b^2)$ where b is the number of objects in the block and $b \ll n$.

Assuming it is possible to efficiently discover the candidate sets for entity resolution, the next challenge is effectively discovering the objects in the candidate sets which are true duplicates of each other. In the literature there is a number of techniques for deduplication and entity resolution, outlined by Getoor et al. in [12]. Often specific measures are chosen for specific applications. One example is the use of approximate string matching methods to determine if two objects are duplicates of each other. For improving performance these methods are sometimes enhanced with domain specific rules that have been created for the specific task. A major downside of this approach is that it requires domain experts to create and maintain these rules, as well as modifications for new data sources and applications.

Our approach is to use the min-wise independent permutations locality sensitive hashing scheme (MinHash) [6], which is a locality sensitive hashing algorithm that quickly estimates the Jaccard Similarity between sets of objects. In this method each object to be processed by MinHash will be the vertex label. These sets of labels will be quickly processed by the MinHash algorithm to produce a candidate set which were found to be similar based on the label string. However, it alone is not sufficient to accurately solve the problem of resolving duplicate entities in large graphs.

In this work we therefore propose to tackle two of the above outlined challenges to develop a general purpose algorithm for entity resolution of duplicated vertices in graphs. We are proposing a novel method which employs ensembles of community detection algorithms, together with the fast string based similarity locality sensitive hashing function MinHash, to quickly and accurately discover duplicate vertices.

The combination of both methods are important, and one without the other is substantially weaker, or of no use. We expect to find that for most large graphs, the communities will consist of many vertices, and the vast majority of those vertices will not be duplicates of each other. Therefore, to check the string similarity of each and every vertex in the community with the others would be time consuming and inefficient. Alternatively if we were just to use MinHash, we would receive many false positive matches due to the legitimate high string similarity of distinct entities (for e.g. ‘10 Main Street, Dublin, Ireland’ and ‘11 Main Street, Dublin, Ireland’). MinHash alone would detect these as duplicates due to their high string similarity, but by taking into account the structural connectivity of these vertices in the graph we hope to determine that it was a false positive.

In order to evaluate our method, we will use a large real world heterogeneous graph consisting of people, businesses and their recorded postal addresses, email addresses and phone numbers.

To summarise, the main contributions are as follows:

- The proposal of a general method of detecting duplicates based on string similarity and community structure in graphs.
- The description of an algorithm for detecting duplicates based on string similarity using locality sensitive hashing and ensembles of community detection algorithms.
- The empirical evaluation of the method on a real world graph of 620885 vertices and 1129986 edges. It is demonstrated how this method can achieve high levels of precision, which is a key measure in a commercial setting.

2 Related Work

Entity resolution is an important and well established area of research with significant importance to the data quality issues which are frequent in real world data. Entity resolution has been applied in numerous areas ranging from web mirror detection [7], data warehousing [5], social network analysis [3] and continues to be an active area of research [13].

Often entity resolution algorithms employ blocking as a means of reducing the computational complexity of the task, and to increase efficiency [1] [13]. However, many of these techniques don't always take into account the structural information often associated with real world data. For example, the SPAN algorithm only considers the similarity of strings, and not the relationship of the each entity to others. This can be crucial in situations where entity resolution has to deal with data that may be similar (i.e. a similar string) but the reason for the difference is unknown.

Fu et al. [11] and Fan et al. [10] both propose approaches for graph based deduplication. However, both of these methods are domain specific, designed for either linking household census data or name disambiguation.

Graph partitioning methods more generally have been considered in the literature, such as Conditional Random Field based methods [9] which creates models of equivalence between relations. In contrast, in the approach of this work, the entire dataset will be modeled as graph, and edges will represent the natural relationship between vertices, that is, for an address, the edge of that address will connect to the person which the address belongs to.

Linked data has been studied in [2] which focuses on deduplication of linked data, as well as group detection. They formulate their problem differently to us, in the form of references obtained from graph objects being mapped to entities via *homogeneous* links of references. In the case of this work, the method proposed utilises the natural heterogeneous graph structure in the form of a single large graph, i.e. there may be no method of creating discrete links.

In summary, to the best of knowledge, this is the first general purpose entity resolution algorithm for vertex deduplication in graphs using locality sensitive hashing and ensembles of community detection algorithms.

3 Our Method

In order to describe our framework we must first introduce some preliminaries.

3.1 MinHash

The MinHash[6] algorithm as briefly introduced above is a locality sensitive hashing algorithm for quickly estimating the Jaccard Similarity between sets. Generally, locality sensitive hashing algorithms aim to hash objects in such a way that similar objects are more likely to collide into the same bucket. It is in these buckets that we will find the potential duplicates which will form the input to the second stage of our algorithm. Given a graph containing vertices which may contain duplicate vertices we first extract all the vertex labels. In a homogeneous graph this will be the labels of all vertices. In a heterogenous graph we will extract only the specific type of vertices we wish to deduplicate, as we can reasonably assume that an address label will never be the duplicate of a phone number label. By extracting the labels we will have the string information we can first use to determine potential duplicates. These labels should consist of the unique representation of the vertex, for e.g. the name of a product from a customer purchasing graph. Rather than operate on the string as a whole, or the individual characters of it, we break it down into subsequences. This retains structure of the labels which may be very informative. For example this can capture common reasons for the deduplication, as in the case of typos or misplaced characters. This is done via a process called shingling [6].

We define a k-shingle to be a substring of length k, such that the the original vertex label becomes a set of k-shingles. Next, as part of the MinHash algorithm, we convert each k-shingle into a signature which is representative of the original label. The detailed description of the idea of generating MinHash signatures can be found in Broder [6]. These signatures are only part of the solution for quickly estimating the similarity between objects. The next step is to further subdivide the signature produced above into rows(r) and bands(b). Each band will consist of r hash-values, which are input to another hash function that maps the band to a new bucket. Importantly there will be b sets of buckets to map to, one set for each band so no overlapping between bands can occur. Thus we can now say that if a band from each of the two documents map to the same bucket, they are candidate pairs based on their string similarity.

MinHash is known to approximate the Jaccard Similarity between sets, and the choice of r and b have significance in that they determine the probability that two documents with a Jaccard Similarity of s become candidate pairs. The other parameter of importance is the k value for extracting the shingles.

Specifically, the probability that the signatures are identical in at least one band, therefore making them a candidate pair, is $1-(1-s^r)^b$. We can exploit this in order to choose appropriate values for r and b prior to the clustering. The similarity s at which there is a 50% chance of two items becoming candidate pairs, at which the rise of the s curve is the steepest, is a function of b and r: $(1/b)^{1/r}$.

After the process of transforming input vertex labels into candidate pairs using MinHash an acknowledgement of a trade-off made should be noted. As this is a general method of deduplication it is important to limit the amount of datatype or domain specific cleaning that occurs in the preprocessing step. For example, for the deduplication of addresses, it seems sensible to extract ZIP codes and street addresses and process those independently. While one can reasonably assume that this will increase the effectiveness of the deduplication process, it would also constrain the ability to generalise it to a wider variety of datatypes and domains. For this reason this method chooses to assume that each label is a simple text string.

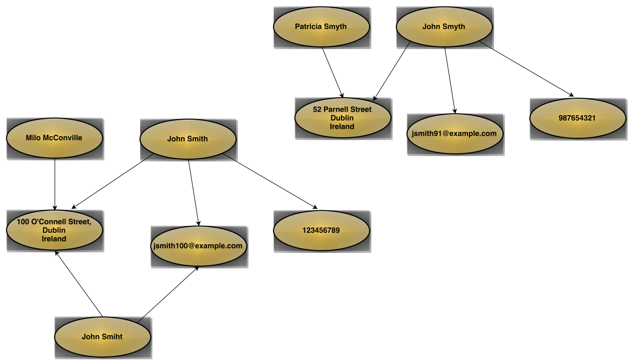

Fig. 1 An example illustrating how we can utilise structural features to augment string similarity.

In summary, each vertex label will be shingled and MinHashed as described above. The output of the process will be a set of candidate pairs. This candidate pair set is formed by taking each pair of vertex labels that were hashed to the same bucket by the MinHash algorithm.

3.2 Structural Features.

If one considers the MinHash step as a blocking technique, it would be successful in that it reduces the number of comparisons to be made from the entire set of vertices, to a smaller subset of it. However, there is still a requirement to further compare the similarity of items within the same 'block' or 'bucket' with each other. In doing so there is a challenge previously discussed; determining if two vertices which may have a high string similarity refer truly to the same entity.

Rather than devise domain specific rules or similarity functions, we propose a method that captures the natural structural properties of the data and therefore increases its generality. Thus, for each pair of potential duplicate entities in the MinHash candidate set (block), we should check if they are similar with regard to some structural connectivity which we will describe soon.

To illustrate, let's consider two examples (from Figure 1):

1. John Smith and John Smyth.
2. John Smiht and John Smith.

In Example 1 we intuitively know that Smith and Smyth are potentially two distinct and relatively well-known variations of the same name. This knowledge is lost in a naive string comparison. Whereas in Example 2, we know that Smiht is not a common surname, and is likely an error in the recording or input of the data. Our challenge is therefore to devise a method of differentiating between true duplicates (i.e., Example 2) and false positives (i.e., Example 1).

In graph data we propose that such information is contained within the structural information of the graph and by utilising this we can achieve our objective. We expect that distance between the two true duplicates (Example 2) would usually be closer than those of the false positive duplicates (Example 1), as the incorrect duplicate would be connected to vertices in the neighbourhood of their duplicated entity. One graph theoretic measure we could use is the shortest path between each pair of potential duplicates. While determining the shortest path between two vertices is relatively simple and fast, it has one major drawback. If we were to use the shortest path between two vertices, we would have to set a specific threshold at which we determine any two vertices within it to be true duplicates. This does not take into account the number, or strength of connections between the vertices. For example, given two strongly connected communities with each connected to the other via a single inter-community edge, we propose that two vertices within a single community (i.e., connected by an intra-community edge) are much closer than any two vertices connected by the inter-community edge. Further setting a threshold may require expert domain knowledge, as well as coupling the threshold to a specific graph model.

3.2.1 Community Detection

Instead, we propose utilising community detection algorithms as a means of filtering out false positives such as Example 1.

In an effort to make our method as general as possible, we propose the use of an ensemble of community detection algorithms. Ensemble methods are common in the supervised learning setting, where a set of classifiers are constructed and the combined models are used to improve the performance, for example by taking an average of their predictions. Ensembles of community detection algorithms have been previously explored [14], but not in the domain of entity resolution.

We will show how ensembles of community detection algorithms can effectively improve the performance of our task in deduplication. We will choose a set of community detection algorithms and each will be applied to a graph containing the vertices we wish to deduplicate. For ensemble methods to be effective it is generally expected that each of the decision makers would exhibit diversity. As each algorithm will have its own criteria and method for the detection of communities we hope to discover diverse sets of communities. We will provide an analysis of the diversity in our experimental evaluation in Section 4.1.

Now that we have introduced and justified two of the major components of our technique, MinHash and ensembles of community detection algorithms, we will continue with the description of our method. Following the discovery of the sets of candidate pairs by MinHash, we will test each candidate pair by querying their existence in each algorithms set of communities. For each algorithms communities, we will check if the candidate pair exist in the same community. If so, we consider this a vote by the algorithm that the candidate pair is a true duplicate. By checking the candidate pair in each algorithms set of communities, we hope that these ‘votes’ will leave the true positive duplicates, while removing the false positives.

Typically in ensemble based algorithms there is a combination step where the results of each of the models is evaluated before a final judgement is made. Voting based methods are popular where each method outputs a decision, typically a class or a label, and the majority (or weighted majority) label is chosen as the final label. We can consider this an estimate of the confidence of the decision. In our method, each algorithm will produce a set of communities, and if the candidate pair are in the same community, this is regarded as a vote in favour of them being duplicate entities. The more of the algorithms that have a duplicate pair within the same community, the higher the confidence we can have that they are in fact true duplicates. As discussed later in Section 4 the level of confidence required can vary depending on application and thus can be considered a threshold.

The high level steps of our algorithm *MCER* are as follows:

1. First, apply each of the community detection algorithms in the ensemble to the entire graph and store the resulting communities.
2. For a heterogeneous graph, extract all vertex labels into a set that are of the type to deduplicate. For a homogeneous graph extract all vertices into a set of labels.
3. Apply the MinHash process, including shingling, to every label in the set. The output of this will be a candidate set consisting of all potential duplicate vertex label pairs.
4. For each pair of potential duplicate vertex labels, query the community they belong to in the communities produced by the ensemble in Step 1. Each time a pair belongs to the same community consider this a vote in favour of the candidate pair being true duplicates.
5. For each pair of candidates that received enough votes to meet the confidence threshold, merge them in the original graph.

4 Experimental Evaluation

While our method is general and can be applied to any graph where string similarity helps in deduplication, we will provide an empirical evaluation of it on a large real world dataset for the purposes of address deduplication. The experimentation was carried out on a heterogeneous graph consisting of 620885 vertices and 1129986 edges. As the graph was heterogeneous, there were multiple vertex types in this graph, for e.g. there were vertices representing clients (for e.g. person or business), as well as vertices for addresses, phone numbers and email addresses.

A description of some known common patterns in the graph will be provided in order to shed some insight into the data. Usually most people and businesses would only be associated with a single address, phone number and email address. However, there may be legitimate and common cases where more than one client would be associated with an address or phone number. For example, two people who live in the same household, or two people who work at the same company.

In rare cases, there may be a large number of entities sharing the same address or phone number. There are known cases where addresses are duplicated due to numerous data entry and business process reasons. For many reasons it is important to ensure addresses are correct. For mailing purposes problems arise when mail is not delivered due to typos or data quality issues, or costs increase when mail is sent multiple times due to duplication. For fraud detection duplicated addresses (intentional or unintentional) may conceal key connections between entities that are not linked due to the duplication.

In this specific experiment a single person or business entity may be connected to any of the other vertex types. The label for each entity vertex was a unique ID from the original dataset. The label for each address consists of what would be considered a fairly standard method of representing an address (building number, street address, city, state, ZIP). The email and phone number vertex labels were the email address and phone number, respectively.

It is known that duplicate addresses exist in this graph, with some insight into the reasons why. The data was collected via a number of separate processes providing numerous opportunities for data to be entered incorrectly into the system. For example, an address can be entered in inconsistent formats using various abbreviations dependent on the person entering the address into the system. It is not uncommon for ‘Road’ to also be entered as ‘RD”, ‘Street’ as ‘ST”, ‘Avenue’ as ‘AV’, ‘Beach’ as ‘BCH’. The same problem can exist for city and state names. Furthermore, ZIP codes may also be entered in their full format, or in shorthand. Further, any of these may further contain typos or misspellings. The extent and variations of addresses are beyond enumeration and thus requires an unsupervised method of detecting duplications.

The method was evaluated with a number of parameter configurations. The first key parameter that can be varied is the MinHash threshold t which determines the estimated Jaccard Similarity required for two labels to be considered as a candidate pair. Naturally it would be expected that lower thresholds would result in a larger set of addresses in the candidate set and thus an improved recall. However, setting the threshold too low may result in increased false positives which is undesirable. In the analysis the efficiency aspect of this threshold is reported. In the MinHash stage it took 19 minutes with both a threshold of 0.7 and 0.85, with 3048157 and 131439 candidate pairs found respectively.

Algorithm	Communities	Size of Largest	Run Time (m)
Leading Eigenvector[15]	71065	305952	4
Louvain[4]	71897	85346	9
Label Propagation[16]	459422	5095	32
Fast-Greedy[8]	96547	280423	1290

Table 1 The performance of the algorithms in the ensemble on the commercial graph.

The other key parameter that can be varied is the number of votes required in order to declare a candidate pair a true duplicate pair. As can be seen in Figure 2 the majority of candidates had 1 or less votes in favor of them being potential duplicates. This may indicate the discriminating power of the ensemble in removing false positives. Further it serves as a form of evidence that an ensemble method is more useful over any single independent use of the algorithms and increases the generality of the method for different variations of communities that may exist across different graphs.

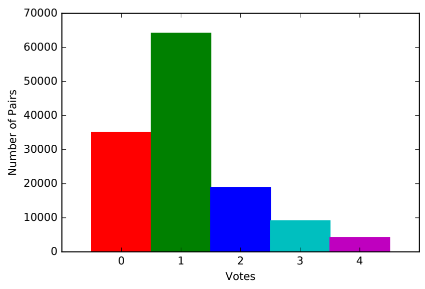

Fig. 2 The number of votes from the ensemble for the candidate pairs.

4.1 Result Analysis

After experimental evaluation, a minimum string similarity threshold *t* of 0.85 was chosen as it represented the best compromise for the business use case. In this scenario a much higher precision was preferred, rather than recall given the business requirements. That is, it is better for some addresses to remain duplicated rather than to incorrectly merge two addresses and create a connection between people where no connection should exist. For the same reason a confidence threshold of 100% was chosen, or in other words *all* of the algorithms in the ensemble placed the candidate pairs in the same community.

Calculating a F1 measure, for example, would be difficult as the large amount of data is completely unlabeled and to do so would be a considerable manual task. For the over 4000 duplicates that all of the algorithms in the ensemble had agreed belonged to the same community, a manual check of a sample of 500 was performed to check if they were true duplicates. An accuracy of 88% accuracy was achieved. A large number of failed cases were due to very poorly input addresses, of which it would have been very challenging for a human analyst to deduplicate.

4.1.1 Ensemble Community Diversity

Diversity in the ensemble is an important property for a successful ensemble, so an analysis of the resulting communities found by each of the algorithms used in the ensemble is provided.

As can be seen from Table 1 significant variance exists in both the amount of communities discovered by the ensemble, ranging from 71065 communities with the Leading Eigenvector[15] algorithm, to 459422 communities with the Label Propagation [16] method. Another property that is interesting to note is the size of the largest communities discovered for each of the algorithms in the ensemble. These also vary from 5095 vertices in the largest of one algorithm to 305952 vertices in the largest of another. This is an indicator of the refinement ability of using these community detection algorithms in an ensemble. The Label Propagation method, with the largest number of communities and also smallest largest community, provides the strongest refinement.

We note the diversity in the community sizes. The Louvain [4] algorithm has a small number of communities less than 20 vertices in size, and this is consistent across the log scale until the latter quarter of 10^3, where there is an increase in the occurrence of these larger communities. A number of these larger communities consisting of over 5000 vertices begin appearing. Note, naturally the Louvain algorithm produces multiple levels of clustering, and it could be possible to selectively cut at a specific level to influence the histograms. However, for this, and other algorithms as appropriate, the optimal communities as determined by the algorithms were used. In stark contrast to this, the Leading Eigenvector algorithm, while discovering a similar number of communities less than 100 vertices in size, discovers only two communities larger than this, at over one hundred thousand and three hundred thousand vertices in size. The Fast-Greedy [8] algorithm also found large numbers of smaller communities, although with more communities in the range of hundreds of vertices than the Leading Eigenvector and Louvain algorithms. It then consistently detects communities distributed between one and three hundred thousand vertices in size. The Label Propagation algorithm detects smaller communities, almost consistently falling in frequency until around one thousand vertices. After passing 1000 vertices in sizes, a small number of communities are detected up to around five thousand vertices in size.

5 Conclusion and Future Work

In this work we proposed a novel method for entity resolution of duplicate vertices based on string similarity and community structure in graphs. In practice MinHash reduced the number of duplicate checks from a complexity of $O(n^2)$ which was hundreds of billions of comparisons to less than one hundred and fifty thousand. Due to the coarse nature of this fast method, the false positive rate is extremely high, particularly when two entities can legitimately have very high similarity but remain distinct entities. To solve this, we proposed using an ensemble of community detection algorithms in order to discover diverse sets of communities. We performed an empirical evaluation of our proposed method on a real world dataset consisting of 620885 vertices and 1129986 edges. In our scenario we had a preference for accuracy, as incorrectly resolving two distinct entities was worse than missing two duplicate entities. Therefore, we focused solely on precision as an evaluation metric and manually confirmed that we could achieve 88% accuracy.

A direction we suggest for further work would be continued experimentation on other datasets. Of particular interest those which also have a gold standard set of entities accurately labeled as duplicates. This would facilitate better empirical evaluation of the approaches effectiveness as it would permit the use of measures such as the F1 score which takes into account both precision and recall.

References

1. Baxter, R., Christen, P., Churches, T.: A Comparison of Fast Blocking Methods for Record Linkage. International Conference on Knowledge Discovery and Data Mining, Workshop pp. 25–27 (2003)
2. Bhattacharya, I., Getoor, L.: Deduplication and Group Detection using Links. LinkKDD pp. 2–11 (2004)
3. Bilgic, M., Licamele, L., Getoor, L., Shneiderman, B.: D-dupe: An interactive tool for entity resolution in social networks. IEEE VAST 2006 pp. 43–50 (2006)
4. Blondel, V., Guillaume, J.: Fast unfolding of communities in large networks. Journal of Statistical Mechanics: Theory and Experiment p. P10008 (2008)
5. Brizan, D.G., Tansel, A.U.: A Survey of Entity Resolution and Record Linkage Methodologies. Communications of the IIMA **6**(3), 41–50 (2006)
6. a.Z. Broder: On the resemblance and containment of documents. Compression and Complexity of SEQUENCES 1997 pp. 21–29
7. Cho, J., Shivakumar, N., Garcia-Molina, H.: Finding replicated Web collections. ACM SIGMOD Record **29**(2), 355–366 (2000)
8. Clauset, A., Newman, M., Moore, C.: Finding community structure in very large networks. Physical Review E pp. 1–6 (2004)
9. Domingos, P.: Multi-Relational Record Linkage. Proceedings of the 3rd KDD Workshop on Multi-Relational Data Mining pp. 31–48 (2004)
10. Fan, X., Wang, J., Pu, X., Zhou, L., Lv, B.: On Graph-Based Name Disambiguation. Journal of Data and Information Quality **2**(2), 1–23 (2011)
11. Fu, Z., Christen, P., Zhou, J.: A graph matching method for historical census household linkage. Lecture Notes in Computer Science (PART 1), 485–496 (2014)
12. Getoor, L.: Entity Resolution for Big Data. KDD p. 204 (2013)
13. Getoor, L., Machanavajjhala, A.: Entity resolution: tutorial. VLDB (2012)
14. Kanawati, R.: YASCA: An Ensemble-Based Approach for Community Detection in Complex Networks. Computing and Combinatorics, Cocoon 2014 pp. 657–666 (2014)
15. Newman, M.E.J.: Finding community structure in networks using the eigenvectors of matrices.pdf. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics **74**(3), 36,104 (2006)
16. Raghavan, U.N., Albert, R., Kumara, S.: Near linear time algorithm to detect community structures in large-scale networks. Physical Review E - Statistical, Nonlinear, and Soft Matter Physics **76**(3), 1–12 (2007)
17. Whang, S.E., Menestrina, D., Koutrika, G., Theobald, M., Garcia-Molina, H.: Entity resolution with iterative blocking. SIGMOD p. 219 (2009)