Using Link Features for Entity Clustering in Knowledge Graphs

Alieh Saeedi, Eric Peukert, Erhard Rahm

University of Leipzig & ScaDS Dresden/Leipzig

Abstract. Knowledge graphs holistically integrate information about entities from multiple sources. A key step in the construction and maintenance of knowledge graphs is the clustering of equivalent entities from different sources. Previous approaches for such an entity clustering suffer from several problems, e.g., the creation of overlapping clusters or the inclusion of several entities from the same source within clusters. We therefore propose a new entity clustering algorithm CLIP that can be applied both to create entity clusters and to repair entity clusters determined with another clustering scheme. In contrast to previous approaches, CLIP not only uses the similarity between entities for clustering but also further features of entity links such as the so-called link strength. To achieve a good scalability we provide a parallel implementation of CLIP based on Apache Flink. Our evaluation for different datasets shows that the new approach can achieve substantially higher cluster quality than previous approaches.

1 Introduction

Knowledge graphs physically integrate entities from multiple sources with their properties, relationships and concepts in a graph-like structure [16, 14]. Popular knowledge graphs in the Web of Data include DBpedia and Yago that combine information about millions of real-world entities (such as persons or locations) of different domains from Wikipedia and other sources. Web search engines such as Google or Bing also integrate information from web pages into their knowledge graphs and use this information to enhance the search results for web queries. The automatic construction and maintenance of such large knowledge graphs faces substantial challenges regarding data quality [5]. One main task is entity resolution to identify different representations referring to the same real-world entity in order to fuse the knowledge about such an entity within the knowledge graph.

While entity resolution and the corresponding problems such as link discovery are intensely investigated research topics (see related work), this problem is still not sufficiently solved for the large-scale integration of data from many sources as needed for knowledge graphs. For more than two sources a binary linking of entities is not sufficient but all matches of the same entity should be clustered together to derive a fused entity representation in the knowledge graph. There are several known approaches for such an entity clustering [8]. At the University of Leipzig, we have recently developed a scalable tool called FAMER (FAst Multi-source Entity Resolution system)[18] that integrates parallel implementations of six such clustering schemes, including Connected Components as the baseline, Correlation Clustering and Star clustering [1]. Clustering is applied on a *similarity graph* where entities are represented as vertices and edges link pairs of entities with a similarity above a predefined threshold. FAMER can construct such similarity graphs for entities of multiple sources based on different linking schemes; existing links from the Web of Data could also be used to build the similarity graph. The clustering schemes use this graph (with the similarity values) to determine groups of matching entities aiming at maximizing the similarity between entities within a cluster and minimizing the similarity between entities of different clusters. We have also developed a tool to visually analyze the similarity graphs and clusters determined by FAMER [17].

Analyzing the clusters determined by the different clustering schemes, we observed some common problems in particular overlapping clusters and so-called source-inconsistent clusters. Algorithms like Star clustering can associate entities to more than one cluster leading to cluster overlaps and thus wrong clusters. For cleaned data sources without duplicates as assumed in FAMER[1] each cluster should contain at most one entity per source. We call clusters violating this restriction *source-inconsistent*. While the similarity graphs determined by FAMER never link entities from the same source the transitive clustering of linked entities, e.g., with the baseline approach Connected Components, can easily lead to source-inconsistent clusters which should be avoided or repaired.

In this paper, we propose and evaluate new algorithms to create high-quality entity clusters or to repair clusters determined by other approaches so that the observed cluster problems are avoided. Specifically, we make the following contributions:

- We propose a new clustering approach called CLIP (Clustering based on LInk Priority) to determine high quality, overlap-free and source-consistent entity clusters. Its cluster decisions are based on a link prioritization considering not only link similarities but also the so-called link strength and link degree.
- We propose an approach called RLIP (cluster Repair based on LInk Priority) to repair entity clusters such that the overlap and source inconsistency problems are resolved. It includes a component to resolve overlapping clusters and uses CLIP to produce source-consistent clusters.
- We develop parallel implementations for both approaches based on Apache Flink and integrate them into the FAMER framework for multi-source entity resolution.
- We comprehensively evaluate the cluster quality and scalability of the new approaches for different datasets and compare them with previously proposed clustering schemes.

In the next section we discuss related work. Section 3 provides an overview of FAMER and the new approaches. In Section 4 we define concepts and describe the new algorithms. Section 5 is the evaluation. Finally, we conclude in Section 6.

2 Related Work

Entity resolution and link discovery have been the subject of much research and several surveys and books provide overviews about the main methods, e.g., [6, 4, 12]. Most

[1] If necessary, the individual sources could be deduplicated before the entity resolution with other sources.

of these previous approaches focus on finding duplicates in a single source or matching entities in only two data sources. Scalability is mainly supported by blocking techniques to reduce the search space for finding possible matches. Some tools also support parallel matching, but mainly for MapReduce (e.g., [10, 12]) and not yet for newer execution platforms based on Apache Spark or Apache Flink.

There is already a substantial number of entity clustering schemes, mainly to group duplicates in a single data source [4]. Hassanzadeh and colleagues comparatively evaluated several of these entity clustering algorithms in [7]. Our FAMER tool [18] includes distributed versions for a subset of the best algorithms from [7] and supports their use for clustering entities from multiple sources. While the problem of cluster overlaps was already observed in [7], considering multiple sources also leads to the problem of source-inconsistent clusters that we address in this paper. Clustering entities from multiple sources of course leads to increased difficulties to achieve high performance and effectiveness compared to considering only one or two data sources. We are aiming at supporting scalability and efficiency by applying both blocking and parallel processing. Furthermore we are proposing advanced clustering methods that avoid the problems of previous clustering schemes and achieve a better match and cluster quality. We are not aware of other tools supporting a parallel entity clustering for multiple sources.

Repair was already studied in pairwise ontology matching, e.g., to ensure that mappings only contain 1:1 matches (which can be achieved by computing stable marriages) or to correct other mapping-induced inconsistencies [2, 15]. However, repair for multiple source mappings was not covered. For entity resolution, [19] and [13] already investigated repair techniques mainly by exploiting the transitive closure of matches to add or remove match links. The repair of entity clusters proposed in [19] depends on manual user feedback which is difficult to provide for very large datasets. Ngonga et. al. are not concerned with entity clusters but focus on finding missing/wrong links and also try to repair entities replicated in different sources [13]. By contrast, CLIP and RLIP avoid/repair overlapping and source-inconsistent clusters in multi-source entity resolution utilizing different link features. Moreover, CLIP and RLIP are implemented as parallel algorithms on Apache Flink to allow for large-scale entity resolution.

3 Overview

Fig. 1 illustrates the main components of the FAMER framework for holistic entity clustering. The input are entities from multiple sources and the output is a set of clusters containing matching entities. The first part is a configurable component to generate a similarity graph where similar entities are linked pairwise with each other. As outlined in [18], this phase starts with blocking, e.g. using Standard Blocking on a specific property, so that only entities of the same block need to be compared with each other. Pairwise matching is typically based on the combined similarity of several properties and a threshold for the minimal similarity. A future version of FAMER will also support the use of learned models for binary match classification. In this paper, we focus on the second part of FAMER to use the similarity graph to determine entity clusters. In the initial version, we support six approaches that will also be considered for comparison in this paper: connected components, correlation clustering (CCPivot) [3], Center [8], Merge Center [8] and two variations of star clustering [1] that can lead to overlapping clusters.

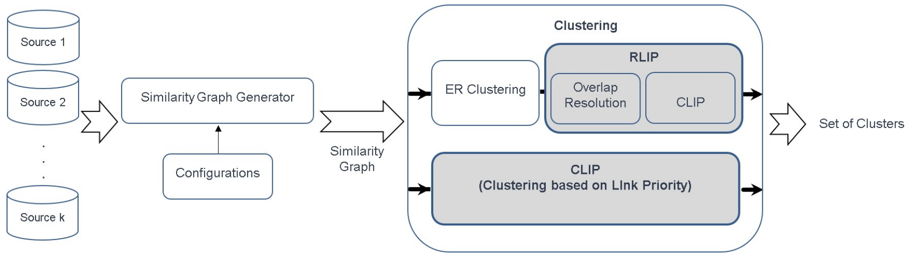


Fig. 1: FAMER workflow for multi-source entity clustering (new components in gray)

There is also a specific clustering method based on already existing links from the Web of Data as input [11] which is not further considered here. To this system we add the components shown in gray in Fig. 1. First, we provide a new clustering scheme CLIP as an alternative to the previous clustering schemes that avoids both overlapping and source-inconsistent clusters. Second, we provide the RLIP approach to repair clusters determined by one of the previous clustering schemes. RLIP first resolves overlapping clusters, if necessary, and then applies CLIP to eliminate source-inconsistent clusters.

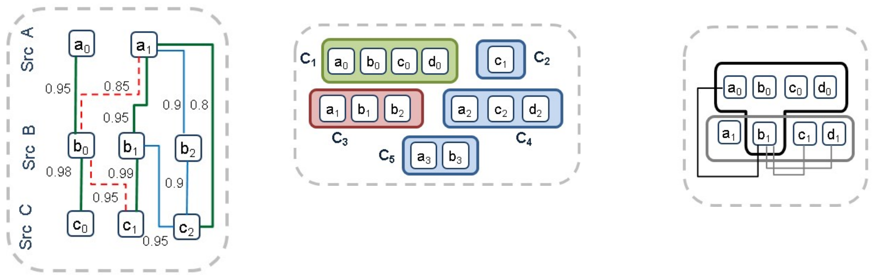
FAMER is implemented using Apache Flink so that the calculation of similarity graphs and the clustering approaches can be executed in parallel on clusters of variable size. For the implementation of the parallel clustering schemes we also use the Gelly library of Flink supporting a so-called vertex-centric programming of graph algorithms to iteratively execute a user-defined program in parallel over all vertices of a graph [9]. The vertex functions are executed by a configurable number of worker nodes among which the graph data is partitioned, e.g., according to a hash partitioning on vertex ids.

4 Approach

We first define the main concepts and then describe the CLIP and RLIP algorithms.

4.1 Concepts

Similarity graph: A similarity graph $SG = (V, E)$ is a graph in which vertices of V represent entities and edges of E are links between matching entities. There is no direct link between entities of the same source. Edges have a property for the similarity value (real number in the interval [0,1]) indicating the degree of similarity.
Cluster: A cluster groups a set of entities that are assumed to represent the same real-world entity. In our implementation, we also include the similarity links between cluster members from the originating similarity graph. Hence a cluster C_i is represented by a *cluster graph* $C_i = (V_i, E_i)$ with the clustered entities in V_i and intra-cluster similarity links in E_i.
Maximum link: An entity from a *source A* may have several links to entities of a *source B*. From these links, the one with the highest similarity value is called maximum link. For example, for entity a_1 in Fig. 2-a the maximum link with respect to source B is the one with similarity 0.95 to entity b_1.
Based on this concept we define the strength of links and classify links into *strong*, *normal*, and *weak* links. Considering a link ℓ between entity e_i from source A and entity e_j from another source *B* we define these link types as follows:



a) Link features b) Cluster Types c) Cluster association degree

Fig. 2: Clustering concepts

Strong link: Link ℓ is classified as a *strong* link, if it is the maximum link from both sides, i.e. for e_i to source B and for e_j to source A. In Fig. 2-a, entity a_1 from source A has a strong link, colored in green, to b_1 in source B. Note that an entity can have several strong links to different sources; e.g., a_1 is also strongly linked to c_2 from source C.

Normal link: Link ℓ is called a *normal* link, if it is the maximum link for only one of the two sides. In Fig. 2-a, the link between a_1 and b_2 is a normal link (colored in blue) as it is the maximum link from b_2 to source *A*, but not the maximum link from a_1 to source *B*.

Weak link: Link ℓ is a *weak* link, if it is not the maximum link for any of the two sides. In Fig. 2-a, the link between a_1 and b_0 is such a weak link and shown with a red dashed line.

Link Degree: The link degree is the minimum vertex degree of its two end point vertices. In Fig. 2-a, the vertex degree of a_1 is 4 and the vertex degree of b_1 is 3, so that the link degree between a_1 and b_1 is $min(4, 3) = 3$.

Link prioritization: Our clustering approach is based on the introduced link features to prioritize links based on their link similarity value, link strength (strong, normal, weak), and link degree. Links with higher similarity value, higher strength and lower degree have priority over links with lower similarity, lower strength and higher degree.

Source-consistent cluster: A cluster that contains at most one entity per source is called a *source-consistent* cluster. In Fig. 2-b, the red-colored cluster is source-inconsistent since it contains two entities (b_1 and b_2) from source *B*. The other clusters colored in blue and green are source-consistent.

Complete cluster: A source-consistent cluster that contains entities from all sources is called a *complete* cluster. The green-colored cluster in Fig. 2-b is a complete cluster for four sources *A*, *B*, *C*, and *D* as it contains one entity from each source.

Cluster association degree: An entity e that is shared between two or more clusters will be in some cases assigned to the cluster with the highest association degree. The association degree of e for cluster C of size k corresponds to the average similarity of e to the $k - 1$ other entities e_i in C, i.e., it is determined by the ratio of the sum of similarity values of the intra-cluster links involving e and $k - 1$. In Fig. 2-c, entity b_1

is member of the gray and black clusters of sizes 4 and 5, respectively. Assuming a link similarity of 1 for the shown links, the association degree for b_1 is $2/3$ for the gray cluster and $1/4$ for the black cluster. Hence, b_1 will be preferably assigned to the gray cluster.

4.2 Entity clustering with CLIP

The proposed CLIP algorithm favors strong links for finding clusters while weak links will be ignored. This helps to find good clusters even when the similarity graph contains many links with lower similarity values. The approach works in two main phases. In the first phase, CLIP determines all complete clusters based on strong links between entities from all sources. The second phase also considers normal links and iteratively clusters the remaining entities based on link priorities such that no source-inconsistent or overlapping clusters are generated.

The pseudocode of CLIP is shown in Algorithm 1. Its input is a similarity graph SG and the output is the cluster set CS. Fig. 3 illustrates the algorithm for entities from four data sources *A*, *B*, *C*, and *D*. Entities with the same index are assumed to belong to the same cluster, e.g. entity a_0 from source A and b_0 from source B. The sample similarity graph in the example already links most matching entities but also contains wrong links, e.g. (b_0, c_1). In phase 1, we start with determining the strength of all links (line 1 of Algorithm 1). Then we only use strong links to determine graph G' (line 2). We then apply $ConnectedComponents$ on G' to identify complete clusters and add these to the output (lines 3-4). In the example of Fig. 3, the second graph in the upper half differentiates between strong, normal, and weak links by showing them as green, blue and dashed lines, respectively. Focussing on strong links, we obtain four connected components in the example, one of which (for index 0) results in a complete cluster that is added to the output of phase 1.

For phase 2, we remove the vertices and edges from the complete clusters. Furthermore, we ignore weak links and only consider strong and normal links in the updated graph G' (lines 5-6 of Algorithm 1). Again we use $ConnectedComponents$ to consider the resulting connected components as possible clusters (line 7). Afterwards these components $Cluster_i$ are processed in parallel (line 8). If the cluster $Cluster_i$ is already a source-consistent cluster, it is directly added to the CLIP output (lines 9-10). Otherwise the component/cluster is source-inconsistent and will be iteratively processed as outlined below. In the example of Fig. 3, phase 2 is illustrated in the lower part which starts with a reduced similarity graph that has no longer the entities from the complete cluster determined in phase 1 and that only contains strong and normal links. We then obtain two connected components one of which (with index 3) is already a source-consistent cluster that is thus added to the ouput. The remaining source-inconsistent component/cluster needs further processing.

In the processing of source-inconsistent clusters/components we initially consider each entity of component $Cluster_i$ as a cluster C_i of its own (lines 12-13). We then iteratively process the intra-component links (lines 15-21) in the order of their maximal link priority and merge linked entity pairs from different sources into larger clusters such that no source inconsistency is created (line 17). For merged clusters, the cluster set

Algorithm 1: CLIP

```
Input  : $SG = (V, E)$
Output: $Cluster\ set\ CS = \{\}$
/* PHASE 1                                                    */
1  $DetermineLinkStrength(E)$
/* Links are classified so that $E = E_{Strong} \cup E_{Normal} \cup E_{Weak}$   */
2  $G' = (V, E_{Strong})$
3  $Cluster\ set\ CS' \leftarrow ConnectedComponents(G')$
4  $CS \leftarrow getCompleteClusters(CS')$
/* PHASE 2                                                    */
5  $V' \leftarrow V - V_{Complete},\ E' \leftarrow (E_{Strong} - E_{Complete}) \cup E_{Normal}$
6  $G' = (V', E')$
/* Vertices and links of the complete clusters are removed from the current
   graph G'                                                   */
7  $CS' \leftarrow ConnectedComponents(G')$
8  for $(Cluster_i \in CS')$ in Parallel do
9      if $(IsSourceConsistent(Cluster_i))$ then
10         $CS \leftarrow CS \cup Cluster_i$
11     else
12         $Assume\ v_t \in V_{Cluster_i}\ as\ a\ cluster\ t = 1, 2, ..., n$
13         $Cluster\ set\ CS_i \leftarrow \{C_1, C_2, ...., C_n\}$
14         $IntraLinks \leftarrow E_{Cluster_i}$
15         repeat
16             $\ell_{Ci,Cj} \leftarrow getMaxPriority(IntraLinks)$
17             $MergeClusters(C_i, C_j)$
18             $UpdateClusterSet(CS_i)$
19             $RemovedLinks \leftarrow RemoveConflictingLinks(IntraLinks)$
20             $IntraLinks \leftarrow IntraLinks - RemovedLinks$
21         until $(IntraLinks \neq \{\})$
22 $CS \leftarrow CS \cup \bigcup_{i=1}^{m} CS_i$
```

is updated accordingly (line 18). Links from the newly formed cluster to entities of the same sources already present in the formed cluster (conflicting links) are removed from the intra-component link set (lines 19-20). The process for each component terminates when the corresponding intra-component link set is empty (line 21). The union of all cluster sets CS_i determined in this way for the different components combined with the previously determined clusters in phase 1 form the final output of CLIP (line 22). In the example of Fig. 3, we start with the link between a_2 and b_2 in the third graph for phase 2 and merge these entities into a new cluster. The link to a_1 from this newly formed cluster is considered as a conflicting link and therefore removed. In the next iterations the link priorities are updated and a new link with maximum priority is selected and clusters are merged. In the example this leads to adding entities c_2 and d_2 to the previously determined cluster while the link of this cluster to entity a_1 is in conflict and will be

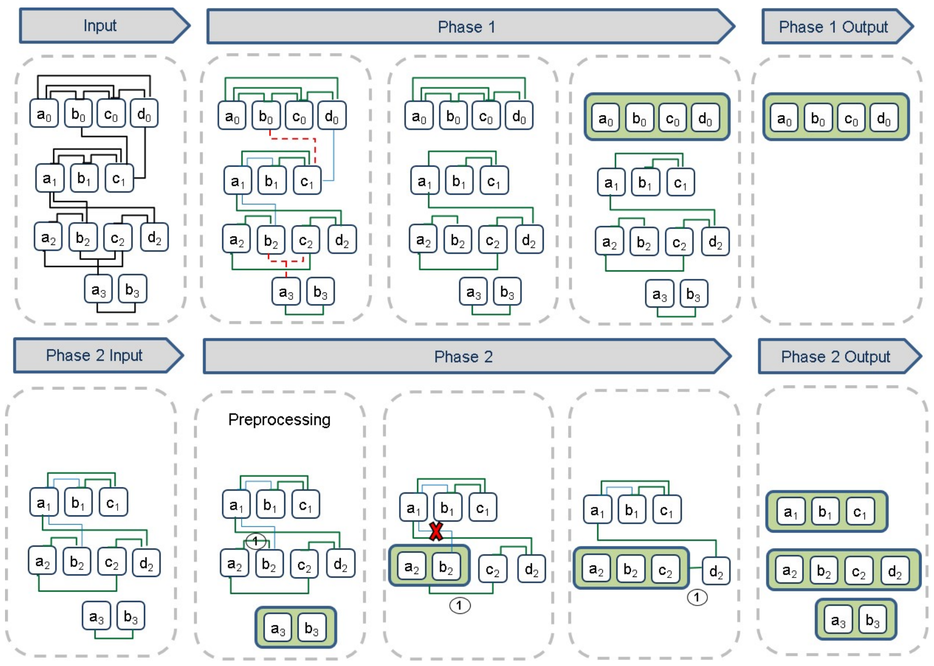


Fig. 3: CLIP example

removed. Similarly, the cluster with index 1 can be generated. Together with the output of phase 1, four clusters are found in the example.

CLIP creates disjoint clusters since it operates on connected components which are by definition disjoint. Furthermore, the iterative processing of source-inconsistent components adds each entity to at most one cluster thereby avoiding cluster overlaps.

4.3 Cluster repair with RLIP

As explained in Section 3, RLIP aims at repairing the output of clustering algorithms by first resolving overlapping clusters, if necessary, and second by using CLIP to repair source-inconsistent clusters. We thus focus on overlap resolution.

The RLIP approach to resolve overlapping clusters also uses the intra-cluster links between entities[2] and favors strong links to select the cluster to which an overlapped entity should be assigned. In particular, overlapped entities that have only strong links to one cluster are assigned to this cluster and for overlapped entities with strong links to several clusters we choose the cluster with the highest association degree for this entity. Overlapped entities with no strong link are kept as singletons. The cluster decision cannot be made directly if an overlapped entity is only strongly linked to another overlapped entity since the best result will depend on the cluster decision for the other overlapped entity. We therefore treat such cases in a second iteration of the algorithm. If in the second iteration the entity still is linked to only overlapped entities, all of them will become singletons.

[2] RLIP could also repair cluster results determined outside FAMER by computing the similarity links between entities within clusters beforehand.

Algorithm 2: Overlap Resolution

```
    Input  : $Cluster\ set\ CS = \bigcup_{i=1}^{m} CS_i(V_i, E_i)$
    Output: $Cluster\ set\ outputCS$
1   $outputCS \leftarrow DetermineLinkStrength(CS)$
    /* Links are classified so that $E_i = E_{i(Strong)} \cup E_{i(Normal)} \cup E_{i(Weak)}$ */
2   for $(iterationNo := 1$ to $2)$ do
3       $OV \leftarrow getOverlappedVertices(outputCS)$
        /* $OV$ : Vertices that belong to more than one cluster */
4       for $v \in OV$ in Parallel do
5           $adjacentVertices \leftarrow StronglyLinkedPairs(v)$
6           if $(adjacentVertices.Size() = 0)$ then
7               $UpdateClusterSet(outputCS, v)$
                /* remove $v$ and its associating links from all clusters. */
8               $outputClusterset \leftarrow outputClusterset \cup (newCluster(v))$
                /* $v$ is a singleton. */
9           else
10              $associatedClusters \leftarrow \{\}$
11              for $(v_n \in adjacentVertices)$ do
12                  if $(v_n \notin OV)$ then
13                      $associatedClusters \leftarrow associatedClusters \cup getCluster(v_n)$
14              if $(associatedClusters.Size() = 0 \wedge iterationNo > 1)$ then
15                  $UpdateClusterSet(v)$
16                  $outputClusterset \leftarrow outputClusterset \cup (newCluster(v))$
17              else
18                  $resolvedCluster \leftarrow \underset{associatedClusters}{\text{argmax}} (association(cluster_i, v))$
19                  $UpdateClusterSet(v)$
20      $iterationNo{+}{+}$
21  return $outputCS$
```

Algorithm 2 outlines overlap resolution in more detail. The input is a set of cluster graphs CS and the output is a set of disjoint clusters $outputCS$. The cluster graphs in the output can be merged into a similarity graph as input for the subsequent execution of CLIP. In line 1, we first determine and store the strength of links in the input cluster graphs. Then, we determine the overlapped entities and process them in parallel in one or two iterations. For overlapped entity v, we store all strongly linked entities in *adjacentVertices* (line 5). If there is no such entity, the overlapped entity is kept as a singleton (lines 6-8). Otherwise, the clusters of non-overlapped entities (line 11) of *adjacentVertices* are determined and stored in the set *associatedClusters* (lines 10-13). If there is no such cluster, i.e., all strongly linked entities are overlapped entities, and we are in the first iteration we wait and this entity will become a singleton in the second iteration (lines 14-16). Otherwise, the cluster association degree of the overlapped entity v to all members of *associatedClusters* is determined and v is assigned to the cluster with the maximal association degree (lines 17-19). Obviously, if there is only one element in *associatedClusters*, v will go to this cluster.

Fig.4 illustrates overlap resolution for four input clusters (C_1, C_2, C_3, and C_4) where entities a_0, a_5 and d_4 belong to two clusters and entity b_2 even to three clusters. The algorithm starts by determining the strength of the links. In the second box of Fig. 4, strong, normal and weak links are shown by green, blue and dashed red lines. The output of the first iteration (third box) shows that entity a_5 is considered as a singleton because it is not strongly linked to any other entity. Entity a_0 is assigned to cluster C_1 because of a higher association degree to C_1 than to C_2. Entity d_4 is strongly linked only to the overlapped entity b_2 so we do not decide about d_4 in this iteration. Entity b_2 has strong links to non-overlapped entities only to cluster C_3 so it is removed from clusters C_2 and C_4. In the second iteration (last box), the remaining overlapped entity d_4 is also resolved. It is linked to entity b_2 which has been assigned to cluster C_3 in the previous iteration, so d_4 is now also assigned to C_3 and removed from cluster C_4. We have thus resolved all overlaps although the resulting clusters are not necessarily source-consistent (e.g., cluster C_3 has two entities from source *D*). So the output of overlap resolution is then processed by CLIP to obtain both disjoint and source-consistent clusters.

5 Evaluation

We now evaluate the effectiveness and efficiency of the proposed clustering and repair algorithms CLIP and RLIP in comparison to the previous clustering schemes of FAMER. We first describe the used datasets from three domains. We then analyze comparatively the effectiveness of the proposed algorithms. Finally, we evaluate runtime performance and scalability.

5.1 Datasets and Similarity Graphs

We use the same evaluation datasets as in [18] to facilitate the comparison with the previously studied clustering schemes. Table 1 summarizes the main characteristics of these datasets as well as their properties and functions used for blocking and calculating the similarity links. The smallest dataset DS1 is a real dataset with geographical entities of type 'settlement' from four sources (DBpedia, Geonames, Freebase, NYTimes). The perfect clusters for DS1 are manually determined. The two larger datasets are based on the real data from the MusicBrainz database (DS2) and the North-Carolina voter registry (DS3) but apply synthetic data generators to create duplicates and corruptions of property values to make entity resolution harder. The DS2 dataset is heavily corrupted and consists of five sources with duplicates for 50% of the original records in two to five sources. The DS3 dataset consists of five sources with 1 million entities per source such that 50% of the entities are replicated in all sources without any corruption. Moreover, 25% of the entities are corrupted and replicated in all sources, and the remaining 25% are corrupted but present in only some sources. The degree of corruption is moderate so that matches are easier to find than for DS2. For determining the similarity graphs with FAMER, we first apply a standard blocking based on the prefix of a specific property (specified in Table 1) so that only entities with the same prefix value need to be compared. Pairwise similarity between entities is determined based on the string similarity

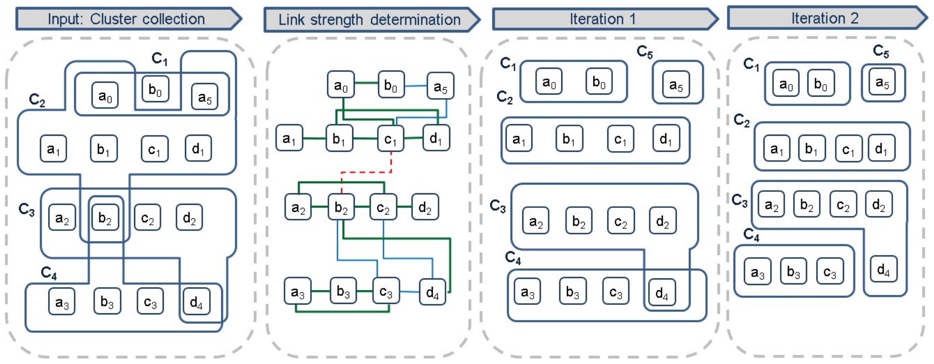


Fig. 4: Overlap resolution (example)

(JaroWinkler or trigram) of selected properties and, for DS1, the geographical distance between settlements and a variable minimal similarity threshold θ.

Table 1: Dataset characteristics and linking details

-	Data sets			Similarity graph configuration	
domain	#entities	#sources	#clusters	blocking key	sim functions
DS1 (geographical)	3,054	4	820	PreLen1 (label)	JW(label), distance
DS2 (music)	20,000	5	10,000	PreLen1(album)	trigram(title)
DS3 (person)	5,000,000	5	3,500,840	PreLen3 (surname)	JW(name),JW(surname), JW(suburb),JW(postcode)

5.2 Cluster Quality

CLIP quality: We first evaluate the cluster quality achieved with the new CLIP clustering scheme in comparison with six known clustering schemes for the three datasets. For this purpose we assume that all entities in the determined clusters match with each other and determine the precision, recall and F-measure compared to the matches of the perfect cluster result. Fig. 5 shows the achieved results for these metrics and different similarity thresholds θ for the seven clustering schemes (CLIP, Connected Components, CCPivot, Center, MergeCenter, Star-1, Star-2) as well as for the similarity graph used as input to the clustering schemes (although this graph has only links but no clusters). The results for the six previous approaches correspond to those reported in [18].

We observe that CLIP achieves an excellent quality result and outperforms all previous algorithms in terms of precision and F-measure for all three datasets. Recall is comparable to the best approaches and only slightly worse compared to approaches such as Connected Components achieving the best possible recall (albeit at the expense of the poorest precision). A closer inspection of the CLIP behavior showed that its good recall is already achieved by determining the connected components for finding complete clusters and source-consistent clusters involving only strong and normal links. By contrast, the CLIP iterations to split source-inconsistent components into several source-consistent clusters is primarily helpful to improve precision. The excellent precision of CLIP, even for lower similarity thresholds, is further due to the ignorance of weak links. This behavior is especially helpful for the relatively dirty dataset DS2 where we had to use very low similarity thresholds to achieve a sufficient recall. CLIP here achieves a F-Measure of 82% compared to only 65-75% for the other clustering schemes. Interestingly, the previous clustering schemes had even problems to outperform the link quality of the similarity graph due to wrong clustering decisions while CLIP achieves clear improvements compared to the similarity graphs by correctly clustering matching entities and removing wrong links between non-matching entities.

RLIP quality: We now study the cluster quality achieved with the proposed repair approach RLIP when applied to the cluster results of the six previous clustering schemes. Fig. 6 summarizes the achieved F-Measure results (averages over all considered values for similarity threshold θ) for the three datasets with the original clustering schemes only (blue bars on the left) and with additionally applying RLIP (red bars on the right). On the right we also show the average F-measure results for CLIP. We

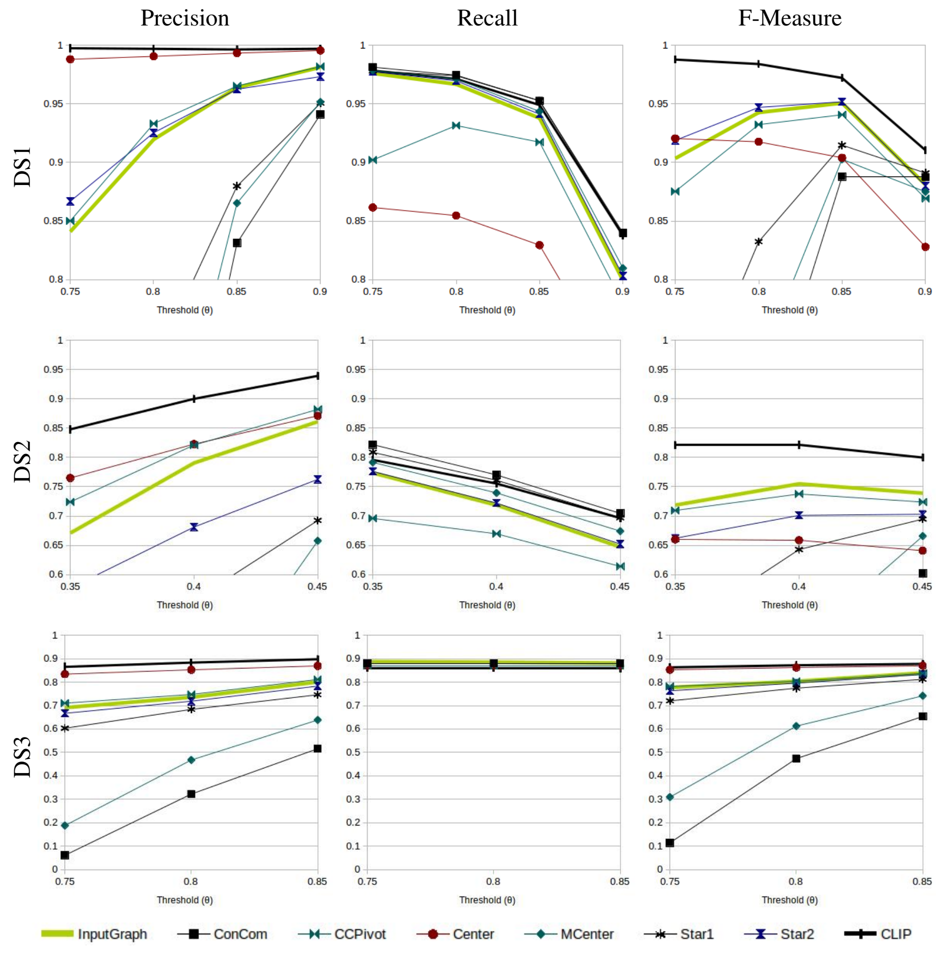


Fig. 5: Cluster quality of CLIP vs other clustering approaches

observe that RLIP can improve F-measure for all algorithms indicating an excellent effectiveness of the proposed cluster repair. The biggest improvements are achieved for the two poorest performing clustering schemes, Connected Component and MergeCenter, that both achieve a high recall but low precision. Here the CLIP component of RLIP achieves a substantial improvement in precision; the already high recall of the input enables that the repaired results for ConnectedComponent and MergeCenter are among the best overall. In fact, the repaired results for ConnectedComponent are essentially identical to the CLIP results. Overlap resolution is only applied to the results of Star-1 and Star-2 and helps to also achieve very good quality for their repaired clusters. The clusters determined with CCPivot and Center can be improved to a lesser degree since these algorithms remove already many links thus hurting recall. The only exception is DS3 for which all clustering schemes achieve a similarly high recall so that the repaired results after applying RLIP are also close together for all algorithms.

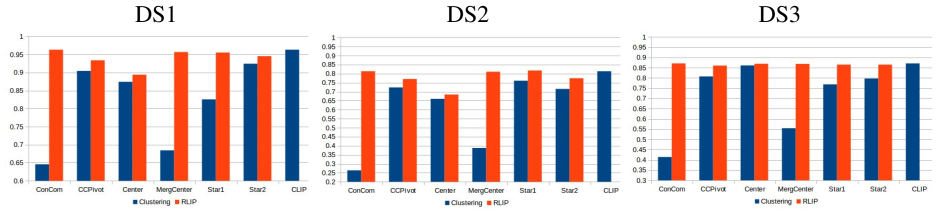


Fig. 6: Cluster quality without and with repair using RLIP

5.3 Runtimes and Speedup

The runtimes of the clustering algorithms are determined on a Hadoop cluster with 16 worker nodes, each consisting of an E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM and two 4 TB SATA disks. The nodes are connected via 1 Gigabit Ethernet. The used software versions are Flink 1.1.2, Hadoop 2.6.0 and openSUSE 13.2. We run Apache Flink with 6 threads and 40 GB memory per worker.

Table 2 shows the runtimes of CLIP for the biggest dataset DS3 with similarity threshold $\theta = 0.8$; Fig 7 depicts the corresponding speedup curve varying the number of workers from 1 to 16. We observe that the parallel execution of CLIP achieves an optimal speedup for up to 8 workers, while the speedup for 16 workers is 11.2 indicating that the dataset DS3 might not be big enough for 16 workers. Comparing the execution times of CLIP with those of the other algorithms (shown in the left part of Table 3), we see that CLIP is relatively slow with the second highest runtimes. Most of the CLIP execution time is spent for the iterative processing of source-inconsistent components in phase 2 of the algorithm. While we process different components in parallel the runtime is determined by the biggest such component which thus becomes a bottleneck.

Table 3 also includes runtimes of repairing clusters with RLIP (middle part) as well as the sum of the execution times for clustering and repair (right part). The shown RLIP execution times include both the times for overlap resolution (for Star-1 and Star-2) and for the CLIP component. We observe that the RLIP runtimes are dominated by the CLIP component while overlap resolution is much faster. The runtimes for the CLIP component of RLIP differ for the different clustering schemes and are the highest for

ConnectedComponent and MergeCenter where precision is improved most by the iterative processing of large components. Interestingly, the combined execution time for the previous clustering schemes and their repair is in several cases lower than only applying CLIP while achieving similar cluster quality, especially for the two Star clustering schemes.

Table 2: CLIP runtimes for DS3 (seconds)

#workers	1	2	4	8	16
runtimes	5477	2696	1303	711	486

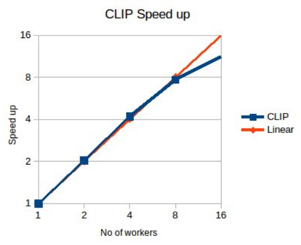


Fig. 7: Speedup

Table 3: DS3 runtimes for clustering schemes and RLIP (seconds)

Method	ER clustering			RLIP						Sum		
#workers	4	8	16	4		8		16		4	8	16
				OV	CLP	OV	CLP	OV	CLP			
ConCom	51	57	55	-	1303	-	711	-	486	1354	768	541
CCPivot	1530	1008	688	-	651	-	361	-	223	2181	1369	911
Center	390	208	117	-	440	-	228	-	144	830	436	261
MCenter	640	349	194	-	882	-	472	-	306	1522	821	500
Star-1	288	149	85	178	489	94	249	55	209	955	492	349
Star-2	214	124	67	202	424	97	236	61	283	840	457	411

6 Conclusion and Outlook

We have proposed a new method called CLIP to cluster matching entities from multiple sources as well as a repair method called RLIP to improve entity clusters determined by other clustering schemes. The approaches avoid or resolve overlapping and source-inconsistent clusters and utilize several features of similarity links in a new way, in particular the link strength. Our evaluation for three datasets shows that the new approaches achieve excellent cluster quality and outperform previous clustering schemes to a large degree. The RLIP repair approach could improve the quality for all considered clustering schemes and achieve comparable quality than applying CLIP alone, in some cases with even lower execution times. The parallel implementations for CLIP and RLIP achieved good speedup values thereby supporting scalability to larger datasets.

In future work, we will investigate further performance optimizations in the iterative portion of the CLIP algorithm. We further plan to make the FAMER tool with the proposed clustering schemes publicly available and apply it in several applications, in particular to build large, high quality knowledge graphs.

7 Acknowledgements

This work is partially funded by the German Federal Ministry of Education and Research under project ScaDS Dresden/Leipzig (BMBF 01IS14014B).

References

1. J. A. Aslam, E. Pelekhov, and D. Rus. The star clustering algorithm for static and dynamic information organization. In *Graph Algorithms And Applications 5*, pages 95–129. 2006.
2. A. Calì, T. Lukasiewicz, L. Predoiu, and H. Stuckenschmidt. A framework for representing ontology mappings under probabilities and inconsistency. *Proc. URSW*, 2007.
3. F. Chierichetti, N. Dalvi, and R. Kumar. Correlation clustering in mapreduce. In *Proc. KDD*, pages 641–650. ACM, 2014.
4. P. Christen. *Data matching*. Springer, 2012.
5. X. Dong et al. Knowledge vault: A web-scale approach to probabilistic knowledge fusion. In *Proc. KDD*, pages 601–610, 2014.
6. A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios. Duplicate record detection: A survey. *IEEE Trans. Knowledge and Data engineering*, 19(1), 2007.
7. O. Hassanzadeh, F. Chiang, H. C. Lee, and R. J. Miller. Framework for evaluating clustering algorithms in duplicate detection. *PVLDB*, 2(1):1282–1293, 2009.
8. O. Hassanzadeh and R. J. Miller. Creating probabilistic databases from duplicated data. *VLDB Journal*, 18(5):1141–1166, 2009.
9. M. Junghanns, A. Petermann, M. Neumann, and E. Rahm. Management and analysis of big graph data: Current systems and open challenges. In *Handbook of Big Data Technologies*, pages 457–505. 2017.
10. L. Kolb, A. Thor, and E. Rahm. Dedoop: Efficient deduplication with hadoop. *PVLDB*, 5(12), 2012.
11. M. Nentwig, A. Groß, and E. Rahm. Holistic entity clustering for linked data. In *Proc. ICDMW*, pages 194–201. IEEE, 2016.
12. M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, and E. Rahm. A survey of current link discovery frameworks. *Semantic Web*, 8(3):419–436, 2017.
13. A.-C. Ngonga Ngomo, M. A. Sherif, and K. Lyko. Unsupervised link discovery through knowledge base repair. *Proc. ESWC*, pages 380–394, 2014.
14. H. Paulheim. Knowledge graph refinement: A survey of approaches and evaluation methods. *Semantic web*, 8(3):489–508, 2017.
15. C. Pesquita, D. Faria, E. Santos, and F. M. Couto. To repair or not to repair: Reconciling correctness and coherence in ontology reference alignments. *Proceedings of the 8th International Conference on Ontology Matching*, pages 13–24, 2013.
16. E. Rahm. The case for holistic data integration. In *Proc. ADBIS*. Springer, 2016.
17. M. A. Rostami, A. Saeedi, E. Peukert, and E. Rahm. Interactive visualization of large similarity graphs and entity resolution clusters. In *Proc. EDBT*, 2018.
18. A. Saeedi, E. Peukert, and E. Rahm. Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In *Proc. ADBIS*. Springer, 2017.
19. Q. Wang, J. Gao, and P. Christen. A clustering-based framework for incrementally repairing entity resolution. *Proc.PAKDD*, pages 283–295, 2016.