Complex Systems Informatics and Modeling Quarterly (CSIMQ)
eISSN: 2255-9922
Published online by RTU Press, https://csimq-journals.rtu.lv
Article 95, Issue 16, September/October 2018, Pages 61–83
https://doi.org/10.7250/csimq.2018-16.04

Scalable Matching and Clustering of Entities with FAMER

Alieh Saeedi[1,2,*] Markus Nentwig[1], Eric Peukert[1,2], and Erhard Rahm[1,2]

[1]Database Group, Department of Computer Science, University of Leipzig, Leipzig, Germany
[2]Competence Center for Scalable Data Services and Solutions, Dresden/Leipzig, Germany

{saeedi, nentwig, peukert, rahm}@informatik.uni-leipzig.de

Abstract. Entity resolution identifies semantically equivalent entities, e.g. describing the same product or customer. It is especially challenging for Big Data applications where large volumes of data from many sources have to be matched and integrated. We therefore introduce a scalable entity resolution framework called FAMER (FAst Multi-source Entity Resolution system) that is based on Apache Flink for distributed execution and that can holistically match entities from multiple sources. For the latter purpose, FAMER includes multiple clustering schemes that group matching entities from different sources within clusters. In addition to previously known clustering schemes FAMER includes new approaches tailored to multi-source entity resolution. We perform a detailed comparative evaluation of eight clustering schemes for different real-life and synthetically generated datasets. The evaluation considers both the match quality as well as the scalability for different numbers of machines and data sizes.

Keywords: Clustering, Matching, Distributed processing, Multi-source.

1 Introduction

Entity resolution (ER) – also called deduplication, record linkage or object matching – is the task of identifying records that refer to the same real-world entity, such as specific costumers, products or publications. This problem is of key importance for improving data quality and for integrating data from multiple sources. Numerous approaches for entity resolution have been developed and investigated [1], [2]. They derive match decisions typically based on the combined similarity of several attribute values and possibly on the contextual similarity of entities (for instance, two publications may match if they have both highly similar titles and co-authors). To achieve high efficiency for large datasets, one has to avoid comparing each entity to all other entities. This is achieved by so-called blocking strategies [1] where only records within the same block (partition) need to be compared with each other, e.g. only publications from the same year are considered.

[*] Corresponding author

Reference: A. Saeedi, M. Nentwig, E. Peukert, and E. Rahm, "Scalable Matching and Clustering of Entities with FAMER," Complex Systems Informatics and Modeling Quarterly, CSIMQ, no. 16, pp. 61–83, 2018. Available: https://doi.org/10.7250/csimq.2018-16.04

Additional information. Authors ORCID iD: A. Saeedi – https://orcid.org/0000-0002-1066-1959, M. Nentwig – https://orcid.org/0000-0001-5268-3083, E. Peukert – https://orcid.org/0000-0002-0338-3364, and E. Rahm – https://orcid.org/0000-0002-2665-1114. PII S225599221800095X. Received: 15 March 2018. Accepted: 15 October 2018. Available online: 31 October 2018.

Entity resolution can also be performed in parallel on multiple processors and computing nodes to achieve additional performance improvements [3].

Most previous ER approaches compare pairs of entities and determine binary match mappings consisting of all correspondences or links between two matching entities. This is a natural approach when one has to integrate only a few data sources but it does not scale well since the number of binary mappings grows quadratically with the number of sources. For instance, integrating data from 200 sources would require the determination (and maintenance) of 19,900 mappings which is not practically feasible with today's ER tools. A better approach for integrating data from multiple data sources is grouping all matching entities within clusters as it allows a more compact match representation than with binary links [4]. It also simplifies the fusion of the matching entities for data integration by combining and consolidating the attribute values of the different cluster members. Furthermore, it allows an incremental integration of additional entities and data sources by comparing them with the set of previously determined clusters.

In our research, we aim at scalable ER approaches for Big Data that are able to deal with large data volumes and multiple data sources. We therefore have developed a new framework called FAMER (FAst Multi-source Entity Resolution system) for multi-source entity resolution that supports clustering matching entities and exploits both blocking and distributed (parallel) processing. It is implemented on top of the distributed dataflow framework Apache Flink to achieve a high scalability to large amounts of data and many machines. FAMER includes multiple clustering schemes to group matching entities; and the main goal of this article is to comparatively evaluate the match quality and runtime performance of these schemes. The considered clustering schemes require, as input, a so-called similarity graph containing all links between matching entities and try to find additional links by considering indirect matches and to eliminate weaker links in favor of more plausible ones. The clustering schemes include previously known clustering schemes (connected components, center clustering, star clustering, merge clustering, correlation clustering) as well as two newly developed approaches for multi-source entity resolution dubbed SplitMerge (introduced in [5], [6]) and CLIP [7]. In total, we perform a detailed comparative evaluation of the match quality and scalability of eight clustering schemes for different real-life and synthetically generated datasets.

This article is an extended version of the conference publication [8]. Compared to [8], we here provide a more detailed discussion of related work and add the description and comparative evaluation of the SplitMerge and CLIP clustering schemes. The CLIP algorithm investigated here is an optimized version of the initial approach of [7] with much better runtimes.

In the next section, we discuss related work on entity resolution and clustering. In Section 3, we provide an overview about our FAMER framework. Section 4 describes the considered clustering algorithms and their distributed implementation. In Section 5, we evaluate the match quality and scalability of the approaches for different datasets. Section 6 summarizes our findings and discusses future work.

2 Related Work

There is a huge amount of literature about ER and there are several books and surveys to provide an overview about the main methods and tools, e.g. [1], [2], and [9]. The decision whether two entities match is typically based on the combined similarity of several attribute values and possibly on the contextual similarity of entities. In current systems, the combination of the similarity values for deriving a match decision is either based on supervised classification models (learned from training examples) or on manually determined match rules. To achieve high efficiency for large datasets, one has to avoid comparing each entity to all other entities. This is made possible by utilizing so-called blocking strategies [1], [10] and additional filter techniques tailored to specific similarity or distance functions (e.g. the triangle inequality for metric-space distance functions) [11]. Further performance improvements are achieved by performing ER in parallel on multiple processors and computing nodes, e.g. on Hadoop platforms. Proposed approaches are primarily based on the use of MapReduce, e.g. [3], [12], and [13]. Only some initial approaches consider the use of the Apache Spark framework for distributed ER [14], [15]. FAMER utilizes Apache Flink which is similar to Apache Spark and both frameworks improve on MapReduce due to a better utilization of in-memory processing and better support for iterative algorithms as needed for clustering [16].

Most of previous ER algorithms try to find matches either in a single source or between two sources only. For a single source, matching entities are typically grouped within disjoint clusters such that any two entities in a cluster should match with each other and no entity should match with entities of other clusters. For two sources, the match result is mostly a binary mapping consisting of pairs of matching entities (also called match correspondences or links). Binary match mappings may be postprocessed to determine clusters of matching entities, e.g. by calculating the transitive closure of the correspondences (connected components) in the simplest case. In FAMER, we extend this approach to more than two sources by first determining a similarity graph with binary match links between entities and then determining clusters of matching entities within the similarity graph. A similar use of similarity graphs has been considered in [17] and [18].

Hassanzadeh and colleagues [19] comparatively evaluated several clustering methods for single-source ER. We implemented parallel versions based on Flink of the best-performing approaches from this study and added them to FAMER, in particular, correlation clustering [20], Center [19], Merge Center [19], and two versions of Star [21] clustering in addition to connected components as a baseline approach. FAMER further includes two clustering algorithms specifically proposed for multi-source entity resolution, SplitMerge [5] and CLIP [7], that will also be evaluated in this article. Both approaches start with determining connected components, but post-process the resulting clusters (components) to obtain better clusters. In SplitMerge, clusters can be split if they contain entities with a low similarity to other cluster members; in a final merge phase some clusters, e.g. singletons from the split phase, can be merged with other similar clusters. CLIP considers different link types in a similarity graph and focuses on the use of so-called strong links for clustering. It is optimized for duplicate-free sources and ensures that each cluster has at most one entity per data source. CLIP can also be used to repair clusters determined by other cluster schemes [7], but this will not be studied in this article.

The comparative evaluation of different clustering schemes, in this article, allows a detailed analysis of their suitability for multi-source ER. For the first time, we here provide the comparison of SplitMerge algorithm with other clustering schemes. In contrast to previous evaluations such as in [19] we consider parallel implementations of the algorithms and also evaluate runtimes and scalability for different data sizes.

3 FAMER Framework for Multi-Source Entity Resolution

Figure 1 illustrates the main components and processing steps of the FAMER framework for distributed multi-source entity resolution. The components are similar to the ones in previous entity resolution tools, but thus support more than two sources and are implemented in Apache Flink to achieve a parallel execution for high scalability. The input of FAMER are thus multiple data sources with the entities to be matched and clustered. The output is a collection of clusters where all entities within a cluster match with each other and different clusters refer to different real-world objects. All entities of a cluster are assumed to match with each other, so that a cluster of m entities represents $m \cdot (m-1)/2$ match pairs.

In this article, we assume that the entities of the different sources are comparable, i.e. they belong to the same entity type (e.g. persons, products cities, etc.) and have comparable attributes. We further assume that all sources are duplicate-free so that we only have to find matching entities between sources. This is based on the experience that data sources should first be preprocessed and cleaned before data integration, in particular duplicates within data sources should first be removed or fused before matching with other sources [22]. The final match clusters should thus be *source-consistent* [7], i.e. they should contain at most one entity from each input data source. As a result, the maximal size of source-consistent clusters is limited by the number of sources.

FAMER consists of two main parts (Figure 1): generation of a similarity graph based on pairwise matches, and clustering. The first component has several steps (blocking, pairwise comparison, match classification), which can be customized according to a configuration input. We provide more details on the different steps below. We also illustrate the workflow of our framework for the person records in Table 1 that originate from four sources A, B, C and D and contain erroneous attribute values as typical for real-world data. Entities with the same index are assumed to belong to the same cluster, e.g. entities a_3 from source A and b_3 from source B. Table 1 groups already the matching records referring to the same person.

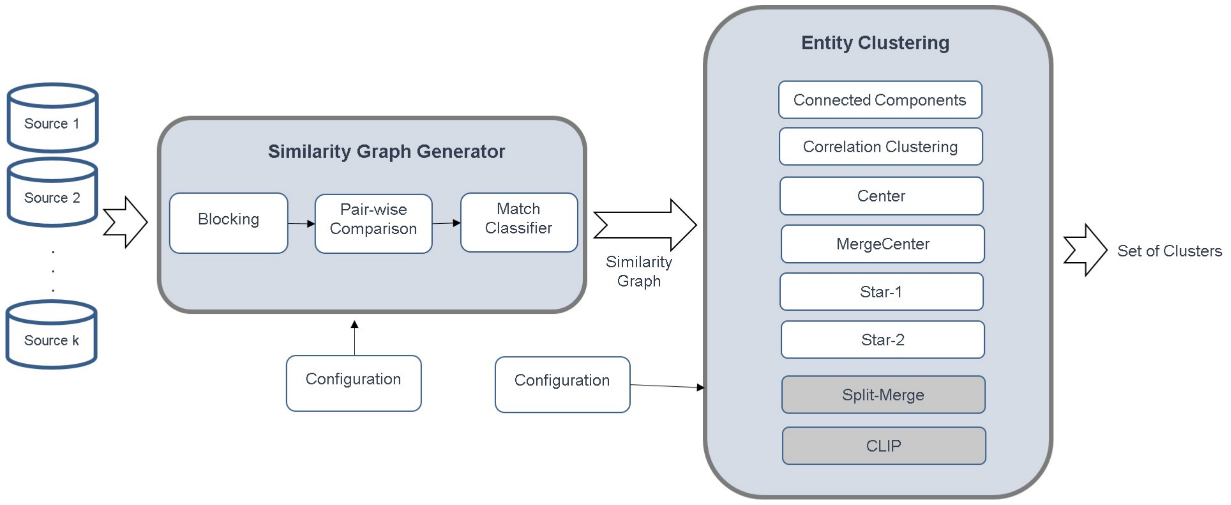


Figure 1. Overview of the FAMER approach for multi-source entity resolution

In the first phase, we start with a *blocking step* to reduce the number of comparisons compared to a naïve approach where each entity of a data source has to be compared against all entities of any other source. FAMER supports different blocking techniques such as Standard Blocking (SB) and Sorted Neighborhood as well as single- and multi-pass blocking [1]. For SB, which we will use in our evaluation, entities are partitioned into blocks by a predefined blocking key (to be provided in the configuration input) on attribute values such that only entities with the same blocking key need to be compared with each other. For the 14 person records in Table 1, we assume that the two initial letters of the surnames form the blocking key. Table 2 shows the resulting blocking key values and blocks sharing the same key value. For this example, even though the entities are not evenly distributed in blocks, blocking reduces the number of comparisons from 91 to only 66+1=67. Such heavily skewed block sizes can result in significant runtime problems in a distributed implementation since the processing of large blocks can overload certain processing nodes while others with smaller blocks are underutilized. In order to achieve load balancing, FAMER supports the so-called *Block Split* method proposed in [12] where large blocks can be processed in several processing nodes. On the other hand, blocking may lead to missing some matches if similar entities are assigned to different blocks (e.g. entities with id c_2 and d_2 are not paired with entities a_2 and b_2). Such missing matches may persist even during clustering and can thus limit the achievable match quality. Multi-pass blocking can reduce this problem (at the expense of more comparisons) by partitioning the entities according to multiple blocking keys.

After blocking, all entities of a block from any of the input data sources are pairwise compared with each other. For each entity pair, we compute the similarity of their attribute values for the attributes and similarity functions specified in the configuration input. Currently, FAMER supports such attribute-based similarity computations for different string similarity metrics (e.g. q-gram,

Table 1. Sample person entities from evaluation dataset DS3.

Id	Name	Surname	Suburb	Post code	SourceId
a_0	Bertha	Watkins	Wilmington	28282	SrcA
b_0	Bertha	Watkins	Wilmingtn	2822	SrcB
c_0	Brtha	Watkins	Wilmington	28222	SrcC
d_0	Bertha	Watkens	Winington	28223	SrcD
a_1	Bernie	Watkins	Wilmsor	28572	SrcA
b_1	Bernie	Watkns	Winstom	2787z	SrcB
c_1	Bernii	Wakens	Windsor	28571	SrcC
a_2	ge0rge	Walker	Winston salem	271o6	SrcA
b_2	Gerge	Waker	Winston salem	27106	SrcB
c_2	George	Alker	Winstons	27106	SrcC
d_2	Geoahge	Alker	Winston	271oo	SrcD
a_3	Gerald	Waker	Winston Salem	27707	SrcA
b_3	Gera1d	Walker	Winston Salem	27707	SrcB
d_4	Larry	Walker	salem	28090	SrcD

Table 2. Keys

Id	Key
a_0	wa
a_1	wa
a_2	wa
a_3	wa
b_0	wa
b_1	wa
b_2	wa
b_3	wa
c_0	wa
c_1	wa
d_0	wa
d_4	wa
c_2	al
d_2	al

Jaro Winkler, edit distance) and domain-specific similarity functions, e.g. using distance between geographical entities. These similarity values are used in the following match classification step to decide about whether or not a pair of entities is assumed to match. The classification approach is also specified in the configuration input, e.g. by match rules specifying the required minimal similarity for the considered attributes. A future version of FAMER will also support supervised match classification where training sets of matching and non-matching entity pairs are used to learn a classification model, e.g. decision trees or support vector machine (SVM) models [23].

The output of match classification is the set of matching entity pairs (links) together with a combined similarity value per link. This output is stored as a *similarity graph* where entities are represented as vertices and match links as edges. Formally, a similarity graph $\mathcal{SG}$= ($\mathcal{V}$,$\mathcal{E}$) is a graph in which vertices of $\mathcal{V}$ represent entities and edges of $\mathcal{E}$ are links between matching entities. There is no direct link between entities of the same source due to the assumption of duplicate-free sources. Edges have a property for the similarity value (real number in the interval [0,1]) indicating the degree of similarity.

The *clustering* step of FAMER aims at grouping together all matching vertices of the similarity graph based on the link structure of the graph and possibly the similarity values. Clustering algorithms typically try to group entities such that the similarity between entities within a cluster is maximized while the similarity between entities of different clusters is minimized. Compared to the similarity graph, the clustering algorithm can ideally add all missing matches (links) and remove all wrong links. As indicated in Figure 1, FAMER currently includes eight clustering algorithms that we describe and evaluate in the following sections. Two of them, SplitMerge and CLIP, require additional configuration parameters.

We have also developed a tool to visually analyze the similarity graphs and clusters determined by FAMER [24]. The tools support the interactive exploration of large graphs and cluster sets, e.g. to analyze potential problems like unusually large, source-inconsistent or overlapping clusters.

Figure 2 illustrates the results of the described workflow for the sample entities of Table 1 and standard blocking as shown in Table 2. The entities are compared pairwise within the blocks and a rule-based match classification is applied resulting in the similarity graph shown in the middle of Figure 2. Compared to the matches assumed in Table 1, the graph misses some links between matching entities, e.g. between a_1 and c_1. Employing ER clustering algorithms, the final clustering determines five clusters which are meant to represent different persons. In fact, the resulting clusters correspond to the ones shown in Table 1 so that even the entities a_2, b_2, c_2, d_2 from different blocks are correctly grouped together (which is possible for SplitMerge clustering).

FAMER is implemented using Apache Flink and a new extension for graph analytics called Gradoop [25], [26]. Hence, all match and clustering approaches can be executed in parallel on Shared Nothing clusters of variable size. Gradoop supports an extended property graph model so that we store the attribute values of entities as key value properties. Analogously, the similarity values of matching entity pairs are represented as edge properties. For the implementation of the parallel clustering schemes we also use the Gelly library of Flink supporting a so-called vertix-centric programming of graph algorithms (see next section).

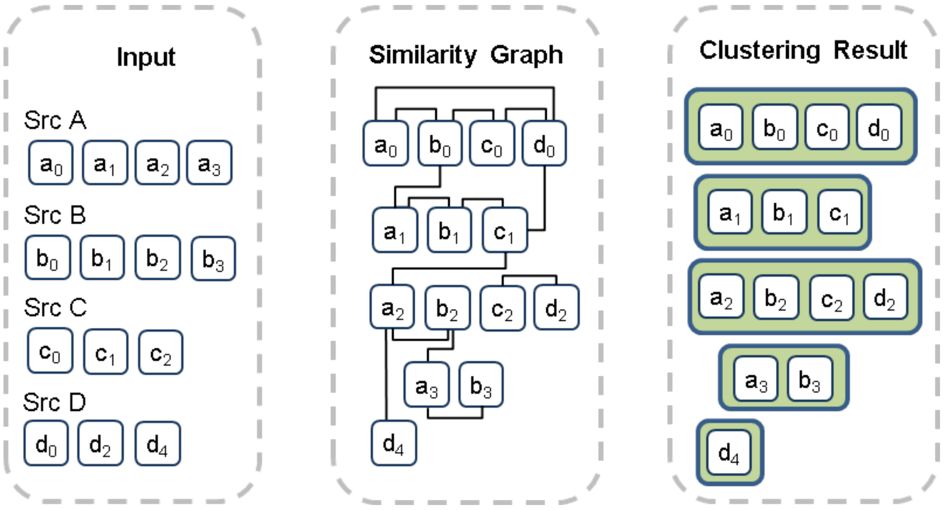


Figure 2. Applying FAMER to the data of Table 1.

4 Clustering Approaches

In this section, we present the eight clustering approaches for entity resolution and their parallel implementation. As described in the previous section, all algorithms use as input a similarity graph with entities from multiple data sources and similarity edges indicating the computed degree of similarity. The clusters determined by the algorithms group a set of entities from different sources that are assumed to represent the same real-world entity. In our implementation, especially for the CLIP algorithm, we also include the similarity links between cluster members from the originating similarity graph. Hence a cluster $\mathcal{C}_i$ is represented by a *cluster graph* $\mathcal{C}_i = (\mathcal{V}_i, \mathcal{E}_i)$ with the clustered entities in $\mathcal{V}_i$ and intra-cluster similarity links in $\mathcal{E}_i$. The SplitMerge algorithm also determines a so-called *cluster representative* for each cluster that is used to determine the similarity between clusters to decide about whether clusters should be merged.

The parallel implementations are based on a vertex-centric programming model, also known as ‘think like a vertex’, to iteratively execute a user-defined program in parallel over all vertices of a graph. In particular, we use the two-step Scatter-Gather model of Gelly that breaks up vertex programming into two functions. In the Scatter step, a value is distributed to all vertex neighbors, and in the Gather step the inputs from the neighbors are collected to update the state of a vertex. The computation proceeds in synchronized iteration steps, called supersteps. Each scatter and each gather execution is performed in a different superstep. Supersteps are executed synchronously, so that messages sent during one superstep are guaranteed to be delivered in the beginning of the next superstep [27]. 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 or range partitioning on the vertex ids. We will explain the vertex-centric implementation in detail for one of the clustering schemes (Center); the other implementations follow similar approaches.

4.1 Connected Components

The similarity graph contains one or more connected components, i.e. subgraphs in which any two vertices are connected to each other and where there is no connection to other components. In the similarity graph of Figure 2, there are two connected components: a small one with the two entities c_2 and d_2 and a bigger one with all other entities. Having the input similarity graph, the connected components are easy to determine in a vertix-centric way by letting every vertex iteratively add all its direct neighbors to its cluster. The approach is therefore easy to implement with Scatter-Gather (as shown in [27]). In the evaluation, we use this approach as a baseline for the comparison with other clustering schemes. It is expected to find additional matches (and thus improving recall) by grouping indirectly matching entities within clusters (components). On the other hand, it may lead to poor precision since indirect matches may not be similar enough to really represent the same real-word object.

4.2 Center Clustering

In contrast to connected components, the Center clustering algorithm [28] utilizes the similarity values (weights) of the edges in the similarity graph. In the sequential algorithm, edges are first sorted based on these weights in descending order and put in a queue. Edges are then removed from the queue and processed one by one. For each edge $e_{(v_i,v_j)}$, if both v_i and v_j are unassigned to any cluster, one of them will be center and the other will belong to the cluster of that center. If one of them is a center and the other is unassigned, the unassigned vertex will belong to the cluster of the center vertex. If both vertices are centers or both of them are non-centers, or one of them is non-center and the other is unassigned, that edge is ignored.

Algorithm 1: Parallel Center

```
   Input  : $\mathcal{SG}$= ($\mathcal{V}$,$\mathcal{E}$)
1  assignVertexPriorities($\mathcal{V}$)
                       /* priority according to a random permutation of vertices */
2  $Center \leftarrow \emptyset$
3  for $v_i \in \mathcal{V}$ in Parallel do
4      repeat
5          $v_{nn} \leftarrow \operatorname{argmax}_j(e_{(v_i,v_j)})$
6          if ($v_{nn} \in Center$) then
7              $v_i$.setClusterId($nn$)
8              $\mathcal{V} \leftarrow \mathcal{V} - \{v_i\}$
9          else if ($v_{nn} \notin \mathcal{V}$) then
10             $\mathcal{E} \leftarrow \mathcal{E} - \{e_{(v_i,v_{nn})}\}$
11         else
12             $v_k \leftarrow \operatorname{argmax}_j(e_{(v_{nn},v_j)})$
13             if (($i = k \wedge i > nn) \vee (v_{nn} = Null)$) then
14                 $Center \leftarrow Center \cup \{v_i\}$
15                 $v_i$.setClusterId($i$)
16                 $\mathcal{V} \leftarrow \mathcal{V} - \{v_i\}$
17     until ($v_i \in \mathcal{V}$)
```

We propose and implemented a parallel version of the Center algorithm (see Algorithm 1). In each round of the algorithm for all unassigned vertices, the outgoing edge with the highest weight must be found. The vertices on both sides of this edge are then processed. If one of them is center, the other will belong to the cluster of that vertex (lines 6–8). If one of them is assigned to another cluster (line 9), i.e, both vertices belong to different clusters, the edge between these two vertices is removed (line 10). If both vertices are unassigned and the edge between them is for both the outgoing edge with the highest weight (line 13, $i = k$), then one of them is assumed as center (line 14) and the other will belong to the same cluster in the next round. For selecting the center in this case we make use of initially assigned (line 1) vertex priorities as done in the sequential algorithm. Hence, the vertex with higher priority is considered as a center (line 16, $i > nn$). If a vertex is not connected to any other vertexes (line 13, $v_{nn} = Null$), it is a singleton. The algorithm iterates until all vertices are assigned to a cluster (line 17).

Algorithm 2: Parallel Center with Scatter-Gather

```
   Input  : $SG = (V, E)$
1  Algorithm Center
2      assignVertexPriorities($V$)
       /* set priority according to a
          random permutation of vertices     */
3      for ($v_i \in V$) do
4          $v_i.K \leftarrow 1$
5      end
6      repeat
7          Phase1: Scatter1 (Vertex)
8                  Gather1 (Vertex, MessageIterator)
            Phase2: Scatter2 (Vertex)
9                  Gather2 (Vertex, MessageIterator)
10     until ($V \neq \{\}$)
11 Procedure Scatter1 (Vertex v)
12     for ($e \in getOutEdges()$) do
13         $msg.Src \leftarrow v.getId()$
14         $msg.Weight \leftarrow e.getWeight()$
15         $sendMessageTo(edge.target(), msg)$
16     end
17 Procedure Gather1 (Vertex v, MessageIterator
    messages)
18     $Array \leftarrow messages.Sort()$
       /* Messages are sorted based on
          their weights descendingly         */
19     $v.NN \leftarrow Array[v.K].getSrc()$
20 Procedure Scatter2 (Vertex v)
21     $msg.Src \leftarrow v.getId()$
22     $msg.NN \leftarrow v.getNN()$
23     $msg.Priority \leftarrow v.getPriority()$
24     for ($e \in getOutEdges()$) do
25         $msg.Weight \leftarrow e.getWeight()$
26         $sendMessageTo(edge.target(), msg)$
27     end
28 Procedure Gather2 (Vertex v, MessageIterator
    messages)
29     $Array \leftarrow messages.Sort()$
       /* sorted based on weights
          descendingly                        */
30     for ($i : v.K \rightarrow Array.Size()$) do
31         $m \leftarrow Array[i]$
32         if ($m.getSrc().isCenter()$) then
33             $v.ClustereId \leftarrow m.getId()$
34             $v.assigned \leftarrow true$
35             $break$
36         end
37         else if ($m.getSrc().isAssigned()$) then
38             $v.K++$
39         end
40         else if ($v.NN = Null \vee (v.NN = m.getSrc() \wedge$
            $v.Priority > m.getPriority())$ ) then
41             $v.ClustereId \leftarrow m.getSrc()$
42             $v.center \leftarrow true$
43             $v.assigned \leftarrow true$
44             $break$
45         end
46     end
```

We implemented parallel Center using the Scatter-Gather model (see Algorithm 2). The algorithm applies two phases that are iteratively executed for all vertices. Phase 1 (Scatter1, Gather1) finds for each vertex v_i its neighboring vertex with the currently highest edge weight, and phase 2 (Scatter2, Gather2) processes the status of the found vertex and assigns v_i to an existing cluster or considers it as a center. Again, we initially assign a priority per vertex (line 3). In phase 1, for each vertex v_i the neighbor with the K-highest edge weight (nearest neighbor NN) is found (lines 13–21). K is a helper variable. It helps to prevent that already assigned vertices are chosen again as neighbors. It is attached to each vertex and initialized with 1 (lines 5–7). It will be incremented in phase 2 when a vertex neighbor has been assigned to a cluster (lines 39–41). In phase 2, all neighbors of a vertex v_i are sorted and processed in descending order of the edge weights (for the edges to v_i) (lines 32–38). Then vertex v_i is set as a center similar to Algorithm 1 (lines 42–47).

4.3 Merge Center

The Merge Center clustering algorithm [28] is a modified version of Center. In contrast to Center, it merges two clusters if a vertex in one cluster is similar to the center of another cluster. Our parallel implementation for Merge Center is very similar to parallel Center but applies an extra iteration for merging clusters. This iteration is initiated right after all vertices are assigned to a cluster. The merge processing is repeated until there are no further cluster changes.

4.4 Star Clustering

The Star clustering algorithm [21] initially computes the degree for each vertex of the similarity graph. Then in each iteration, the unassigned vertex with the highest degree becomes center and all its direct neighbors are assigned to its cluster. The algorithm terminates when all vertices are assigned to a cluster. In contrast to all other clustering approaches, Star clustering can result in overlapping clusters. As a consequence, it introduces the need of a post-processing to select the best cluster for entities that have been assigned to several clusters.

Our parallel version of the Star algorithm is described in Algorithm 3. Initially, the degree of all vertices is computed and, if the degree of a vertex is greater than the degree of all its neighbors, that vertex becomes a center (lines 4–7). If the degree of two adjacent vertices is equal, the one with higher priority is assumed as a center. Similar to the previous parallel algorithms, vertex priority is initially determined by generating a random permutation of vertices (line 1). Then each center and all its neighbors are considered as a cluster. (lines 8–12). The Scatter-Gather version of Algorithm 3 uses three phases. In the first phase the degree of each vertex is computed. In the second phase, centers are selected, and in the final phase, clusters are grown around the centers.

Algorithm 3: Parallel Star

```
   Input  : $\mathcal{SG}$= ($\mathcal{V}$,$\mathcal{E}$)
1  $\mathcal{V}\leftarrow \{v_1, ..., v_n\}$
   /* A random permutation of vertices                    */
2  $Center \leftarrow \{\}$
3  repeat
4      for ($v_i \in \mathcal{V}$) in Parallel do
5          $v_{max} \leftarrow \underset{v_j \in \{v_j | e(v_i,v_j) \in E\} \cup \{v_i\}}{\text{argmax}} (\texttt{computeDegree}(v_j)))$
6          if ($v_i = v_{max}$) then
7              $Center \leftarrow Center \cup \{v\}$
8      for ($v_i \in \mathcal{V}$) in Parallel do
9          for ($e(v_i, v_j) \in E$) do
10             if ($v_j \in Center$) then
11                 $v_i$.addClusterId($v_j$.getId())
12                 $\mathcal{V}\leftarrow \mathcal{V}-\{v_i\}$
13 until ($\mathcal{V}\neq \{\}$)
```

We use two methods for computing the degree of vertices resulting into algorithms Star-1 and Star-2. For Star-1, we count the number of outgoing edges of a vertex, while Star-2 is based on the average similarity degrees of the outgoing edges of a vertex.

4.5 CCPivot Correlation Clustering

The original correlation clustering approach [29] uses a graph with positive and negative edge weights to indicate whether two vertices are similar (positive edge weight) or dissimilar (negative edge weight). The goal is to find a clustering that either maximizes agreements (sum of positive edge weights within a cluster plus the absolute value of the sum of negative edge weights between clusters) or minimizes disagreements (absolute value of the sum of negative edge weights within a cluster plus the sum of positive edge weights across clusters). Gionis et al. propose an approximate and iterative solution for this optimization problem [30] that randomly selects an unassigned vertex as a cluster center in each round. Then all unassigned neighbors of the selected center are added to the cluster and marked as assigned. The algorithm terminates when there is no unassigned vertexes left.

This simple algorithm suffers from too many rounds making it unsuitable for very large graphs. Some studies therefore proposed parallel solutions [20], [31] that select multiple centers in each round. They also address the newly introduced concurrency problem to avoid that a vertex is assigned to more than one center at a time. We implemented the parallel pivot approach of [20], called CCPivot, since it fits well the Scatter-Gather paradigm. In each round of this algorithm, several vertices are considered as active nodes, i.e. as candidates for becoming a cluster center (or pivot). In the next step, active nodes that are adjacent to each other are removed from the set of active nodes; the remaining vertices become centers. Then adjacent vertices of each center are assigned to that center and form a cluster. If one vertex is adjacent of more than one center at the same time, it will belong to the one with higher priority. As in the other algorithms, the vertex priorities are determined in a preprocessing phase.

Our Scatter-Gather implementation of this algorithm uses three Scatter-Gather phases: one for computing the current maximum degree of the graph, one for selecting active nodes and applying the concurrency-aware rule to select final centers, and one for growing clusters around centers.

4.6 SplitMerge Clustering

The SplitMerge approach proposed in [5] is more general than the other clustering schemes as it can deal with entities of different semantic types as well as dirty input sources and links, e. g.with duplicates in sources. Furthermore, SplitMerge can compute additional links between entities based on a similarity function provided within a configuration parameter. Further parameters are similarity thresholds for the split and merge phases and a blocking function for the merge phase.

Algorithm 4 shows the pseudo-code of the SplitMerge approach consisting of three main phases: (1) determining initial clusters by applying connected components and making the components source-consistent, (2) splitting clusters to ensure a high intra-cluster similarity and (3) merging similar clusters. In contrast to [5], we have omitted the preprocessing phase since we only consider entities of a single type and duplicate-free sources in this study. The application of SplitMerge to the similarity graph from Figure 2 is illustrated in Figure 3. More details on the Flink implementation of SplitMerge are described in [6].

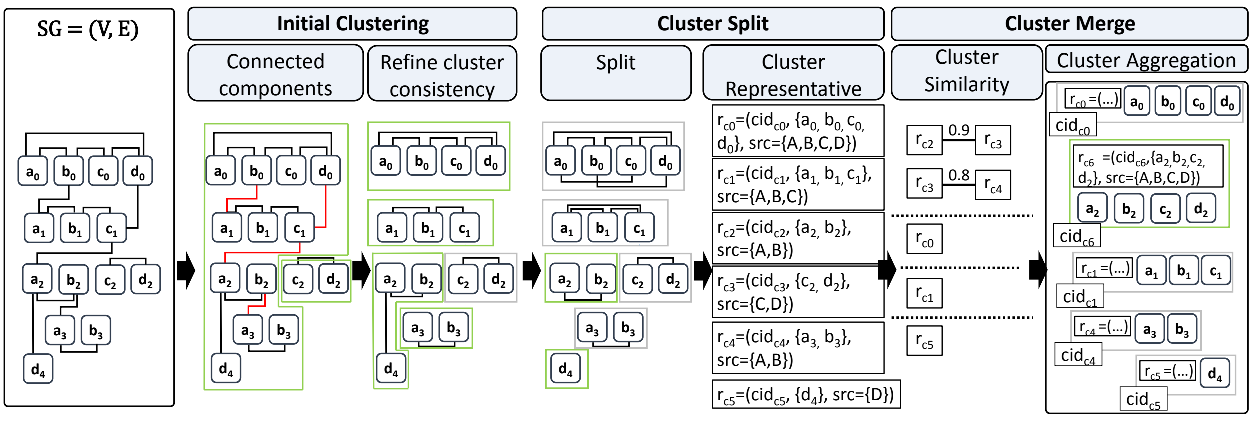

SG = (V, E)
Initial Clustering
Connected components
Refine cluster consistency
Cluster Split
Split
Cluster Representative
Cluster Merge
Cluster Similarity
Cluster Aggregation
r_{c0}=(cid_{c0}, {a_0, b_0, c_0, d_0}, src={A,B,C,D})
r_{c1}=(cid_{c1}, {a_1, b_1, c_1}, src={A,B,C})
r_{c2}=(cid_{c2}, {a_2, b_2}, src={A,B})
r_{c3}=(cid_{c3}, {c_2, d_2}, src={C,D})
r_{c4}=(cid_{c4}, {a_3, b_3}, src={A,B})
r_{c5}=(cid_{c5}, {d_4}, src={D})
0.9
0.8
r_{c6} =(cid_{c6},{a_2,b_2,c_2, d_2}, src={A,B,C,D})

Figure 3. Running example processed with SplitMerge clustering.

SplitMerge starts with computing connected components (line 2 of Algorithm 4) on the input similarity graph to create initial components $\mathcal{C}_{\text{init}}$. The resulting components may often violate the required source consistency since entities from the same source may be indirectly linked and thus become members of the same connected component. In our example in Figure 3, there are only two connected components where the smaller one (with entities c_2 and d_2) is source-consistent but the larger one contains up to four entities per source. To achieve source-consistent clusters, we decompose the inconsistent components by removing links that result in a violation of source consistency. The links between (a_0, b_0) and (b_0, a_1) result in a source inconsistency for source A and we solve this by removing one of the two links (the one with lower similarity). Another example with three links resulting in a source inconsistency is (b_1, c_1, a_2, b_2); again, we eliminate at least one link, e. g., (c_1, a_2), to solve the problem.

To identify the links to be removed, we record for every entity e the set of already associated data sources in an element `assocSrc`(e) which initially contains the source of e (line 4). We iterate over all links of a component in descending order of their similarity. For each considered link (e_s, e_t), we check whether it results in a source inconsistency which is the case if there is a non-empty overlap between `assocSrc`(e_s) and `assocSrc`(e_t). If there is such a conflict, the link will be eliminated (line 8). Otherwise, we update both sets of associated sources to the union of `assocSrc`(e_s) and `assocSrc`(e_t) (line 10). In the example of Figure 3, the conflicting links that are removed are shown in red. For instance, if we first process link (a_0, b_0) we will have sources A and B in `assocSrc`(a_0) and `assocSrc`(b_0). The link (b_0, a_1) will then lead to a conflict for b_0 which is already associated with source A so that this link is eliminated. After the processing of all links, we determine the connected components with the remaining links to compute the source-consistent subcomponents (line 11). In our running example, we obtain the four smaller clusters shown (with green borders) in the third graph from the left in Figure 3.

For the **split phase**, we process the clusters from the first phase in parallel. For each cluster, we first determine link similarities for each pair of entities based on the similarity function f_{sim} provided in the input. This is needed to identify entities with an insufficient similarity to other cluster members. To determine possible splits (line 15) we determine for each entity the average similarity of its links to other cluster members and separate an entity if the average similarity is below the split threshold t_s. After the elimination of such entities, we iteratively repeat this split processing based on recomputed entity similarities until all entity similarities are at least as high as threshold t_s. In our example, this processing leads to the elimination of d_4 from cluster a_2, b_2, d_4 (fourth graph from the left in Figure 3). For each resulting cluster, we next determine a `cluster representative` (line 16) from the properties of the cluster members, e. g.based on the values of preferred sources or a majority consensus of values. As indicated in Figure 3, each cluster representative has a unique id and keeps track of the covered cluster entities and their sources as provenance information. The representatives are used for a simplified computation of cluster similarities as needed for the final merge phase.

The goal of the **merge phase** is to identify highly similar pairs of clusters that likely represent the same real-world entity and should thus be combined. This can also help to assign entities separated during the split phase to a more similar cluster. The first step is to determine a so-called cluster mapping $\mathcal{CM}$ (line 18 of Algorithm 4) consisting of all cluster pairs with a similarity above the merge threshold t_m (merge candidates). The similarity between clusters is computed by applying function f_{sim} on the cluster representatives. Since the computation of these similarities is an expensive process for many clusters, we reduce the number of comparisons by applying a blocking function bf specified as an input parameter (in the current implementation we apply standard blocking on selected properties of the cluster representatives). Furthermore, we only compare clusters with entities from different sources since otherwise merging these clusters would violate source consistency. In our example in Figure 3, we have three clusters in the first block and

Algorithm 4: SplitMerge Clustering

```
Input: $SG = (\mathcal{V}, \mathcal{E})$, simFunction $f_{sim}$, blocking function $bf$, thresholds $t_s, t_m$
Output: Cluster set $CS$
1  $\mathcal{CS} \leftarrow \emptyset$
2  $\mathcal{C}_{\text{init}} \leftarrow$ computeConnectedComponents($\mathcal{V}, \mathcal{E}$)      /* initial clustering */
3  for $\mathcal{C}_i(\mathcal{V}_i, \mathcal{E}_i) \in \mathcal{C}_{init}$ in Parallel do
4      $\mathcal{V}_{\text{i}} \leftarrow$ initAssocSrc($\mathcal{V}_i$)
5      $\mathcal{E}_{\text{sorted}} \leftarrow$ sortLinkSims($\mathcal{E}_i$)
6      foreach $(e_s, e_t) \in \mathcal{E}_{sorted}$ do
7          if (assocSrc($e_s$) $\cap$ assocSrc($e_t$) $\neq \emptyset$) then
8              $\mathcal{E}_{\text{i}} \leftarrow$ removeLink($(e_s, e_t)$)
9          else
10             $\mathcal{V}_{\text{i}} \leftarrow$ updateAssocSrc(assocSrc($e_s$) $\cup$ assocSrc($e_t$))
11     $\mathcal{C}'_{\text{i}} \leftarrow$ computeConnectedComponents($\mathcal{V}_i, \mathcal{E}_i$)
12     $\mathcal{CS} \leftarrow \mathcal{CS} \cup \mathcal{C}'_{\text{i}}$
13 for $\mathcal{C}_i \in \mathcal{CS}$ in Parallel do
14     $\mathcal{C}_{\text{i}} \leftarrow$ computeLinkSim($\mathcal{C}_i, f_{sim}$)
15     $\mathcal{C}_{\text{split}} \leftarrow$ clusterSplit($\mathcal{C}_i, t_s$)      /* cluster split */
16     $\mathcal{C}_{\text{split}} \leftarrow$ createRepresentatives($\mathcal{C}_{split}$)
17     $\mathcal{CS} \leftarrow \mathcal{CS} \cup \mathcal{C}_{\text{split}}$
18 $\mathcal{CM} \leftarrow$ computeClusterSim($\mathcal{CS}, f_{sim}, t_m, bf$)      /* create cluster mapping $\mathcal{CM}$ */
19 while $\mathcal{CM} \neq \emptyset$ do
20     $(c_1, c_2) \leftarrow$ getBestMatch ($\mathcal{CM}$)
21     $c_m \leftarrow merge(c_1, c_2)$      /* cluster merge */
22     $\mathcal{CS} \leftarrow \mathcal{CS} \backslash \{c_1, c_2\} \cup \{c_m\}$
23     $\mathcal{CM} \leftarrow$ adaptMapping ($\mathcal{CM}, \mathcal{CS}, c_m, c_1, c_2, f_{sim}, t_m$)
24 return $\mathcal{CS}$
```

only one in the remaining three blocks. For the first block, we obtain two merge candidates with a sufficiently high cluster similarity.

Cluster merging is an iterative process (lines 19 to 23) that continues as long as there are merge candidates in the determined cluster mapping $\mathcal{CM}$. In each iteration, we select the pair of clusters (c_1, c_2) with the highest similarity from $\mathcal{CM}$ (line 20) and merge it into a new cluster c_m (line 21). This merging also includes the computation of a new representative for c_m. The "old" clusters c_1 and c_2 are removed from the cluster set and the new cluster c_m is added. We further need to adapt $\mathcal{CM}$ by removing all cluster pairs involving either c_1 or c_2 (line 22). Furthermore, we have to extend $\mathcal{CM}$ by similar cluster pairs (c_i, c_m) for the new cluster c_m with a cluster similarity of at least t_m and entities from different sources (line 23). For our running example, we first process the merge candidate with similarity 0.9 and obtain the merged cluster $\{a_2, b_2, c_2, d_2\}$. The second merge candidate will be removed and it is checked whether the new cluster results in new merge candidates. Since the new cluster contains already entities from every source, merging any other cluster would result in a source inconsistency so that no new merge candidates result in the example. The final outcome of SplitMerge contains five clusters which correspond to the perfect result in Figure 2.

4.7 CLIP

The CLIP algorithm (Clustering based on LInk Priority) [7] is able to produce source-consistent clusters. It utilizes different link characteristics such as the link strength and link degree that we introduce first before outlining the approach.

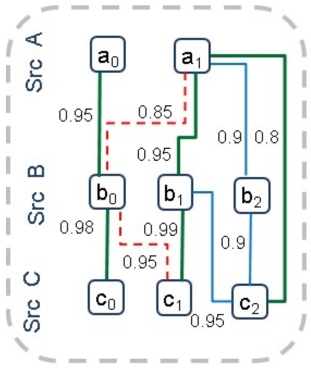


Figure 4. Link strength

In a similarity graph, 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 instance, for entity a_1 in Figure 4 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 them 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:

- 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 Figure 4, 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.
- Link ℓ is called a **normal link**, if it is the maximum link for only one of the two sides. In Figure 4, 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*.
- Link ℓ is a **weak link**, if it is not the maximum link for any of the two sides. In Figure 4, the link between a_1 and b_0 is such a weak link and is shown with a red dashed line.

Furthermore, we define **link degree** of a link as the minimum degree of the two linked vertices. In Figure 4, 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 $min(4, 3) = 3$. Finally, we call a source-consistent cluster that contains entities from all sources a **complete cluster**. In Figure 2, the cluster containing the entities with index 0 is a complete cluster since it is source-consistent (at most one entity per source) and contains at least one entity for each of the four sources. The definition implies that complete clusters contain exactly one entity from each input data source.

The CLIP algorithm favors strong links for finding clusters while weak links will be ignored. This aims at finding 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 clusters are generated.

The pseudocode of CLIP is shown in Algorithm 5. Its input is a similarity graph $\mathcal{SG}$ and a configuration parameter specifying how tho determine link priorties; the output is the cluster set $\mathcal{CS}$. Figure 5 illustrates the algorithm for the entities and similarity graph from our running example from Figure 2. In phase 1, we start with determining the strength of all links (line 2 of Algorithm 5). Then we apply `computeConnectedComponents` on the graph with vertices $\mathcal{V}$ and only strong links $\mathcal{E}_{\text{Strong}}$to identify complete clusters and add these to the output (lines 3–4). In the example of Figure 5, the second graph in the upper half differentiates between strong, normal, and weak links by showing them as green, blue and dashed red lines, respectively. Focusing on strong links, we

Algorithm 5: CLIP

```
Input  : $\mathcal{SG}$= ($\mathcal{V}$,$\mathcal{E}$), $config$
Output: Set of clusters $\mathcal{CS}$
1  $\mathcal{CS}$← ∅
   /* PHASE 1                                                                 */
2  determineLinkStrength($\mathcal{E}$)
                 /* Links are classified so that $E = E_{Strong} \cup E_{Normal} \cup E_{Weak}$ */
3  $\mathcal{CS}'$← computeConnectedComponents($\mathcal{V}$,$\mathcal{E}_{Strong}$)
4  $\mathcal{CS}$← getCompleteClusters($\mathcal{CS}'$)
   /* PHASE 2                                                                 */
5  $\mathcal{V}'$← $\mathcal{V}$- $\mathcal{V}_{\text{Complete}}$, $\mathcal{E}'$← ($\mathcal{E}_{\text{Strong}}$- $\mathcal{E}_{\text{Complete}}$) ∪ $\mathcal{E}_{\text{Normal}}$
        /* Vertices and links of the complete clusters are removed from the current graph G' */
6  $\mathcal{CS}'$← computeConnectedComponents($\mathcal{V}'$,$\mathcal{E}'$)
7  for ($\mathcal{C}_i$∈ $\mathcal{CS}'$) in Parallel do
8      if (isSourceConsistent($\mathcal{C}_i$)) then
9          $\mathcal{CS}$← $\mathcal{CS}$∪ $\mathcal{C}_{\text{i}}$
10     else
11         $\mathcal{E}_{\text{sorted}}$← sortLinksByPriority($\mathcal{E}'$, $config$)
12         foreach $(e_s, e_t) \in \mathcal{E}_{sorted}$ do
13             if (compatible($\mathcal{C}_s$,$\mathcal{C}_t$)) then
                       /* $\mathcal{C}_{\text{s}}$ and $\mathcal{C}_{\text{t}}$ are the clusters of entities $e_s$ and $e_t$, respectively */
14                 mergeClusters($\mathcal{C}_s$,$\mathcal{C}_t$)
15                 updateClusterSet($\mathcal{CS}_i$)
16 $\mathcal{CS}$← $\mathcal{CS}$∪ $\bigcup_{i=1}^{k}$ $\mathcal{CS}_{\text{i}}$
```

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 (lines 5 of Algorithm 5). Again we use `computeConnectedComponents` to consider the resulting connected components as possible clusters (line 6). Afterwards these components $\mathcal{C}_{\text{i}}$ are processed in parallel (line 7). If the cluster $\mathcal{C}_{\text{i}}$ is already a source-consistent cluster, it is directly added to the CLIP output (lines 8–9). Otherwise the component/cluster is source-inconsistent and will be processed as outlined below. In the example of Figure 5, 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 three connected components two of which (with index 2 and index 3) are already source-consistent clusters that are thus added to the output. The remaining source-inconsistent component/cluster needs further processing.

In the processing of source-inconsistent clusters/components we sequentially process the intra-component links (lines 12–15) in the order of their maximal link priority (determined by `sortLinksByPriority` in line 11) which is based on the link similarity value, link strength and link degree. The parameter $config$ in line 11 determines the weight of these three factors to compute the link priority. Assuming that the individual entities are singleton clusters in the beginning, we iteratively process the links to determine whether the clusters of the linked entities can be merged without introducing source inconsistency. In line 13, we thus check for each link (e_s, e_t) whether their clusters $\mathcal{C}_{\text{s}}$ and $\mathcal{C}_{\text{t}}$ are compatible, i.e. they do not include more than one entity per source. Only if this the case, we merge the two clusters and update the cluster set accordingly (lines 14–15). The

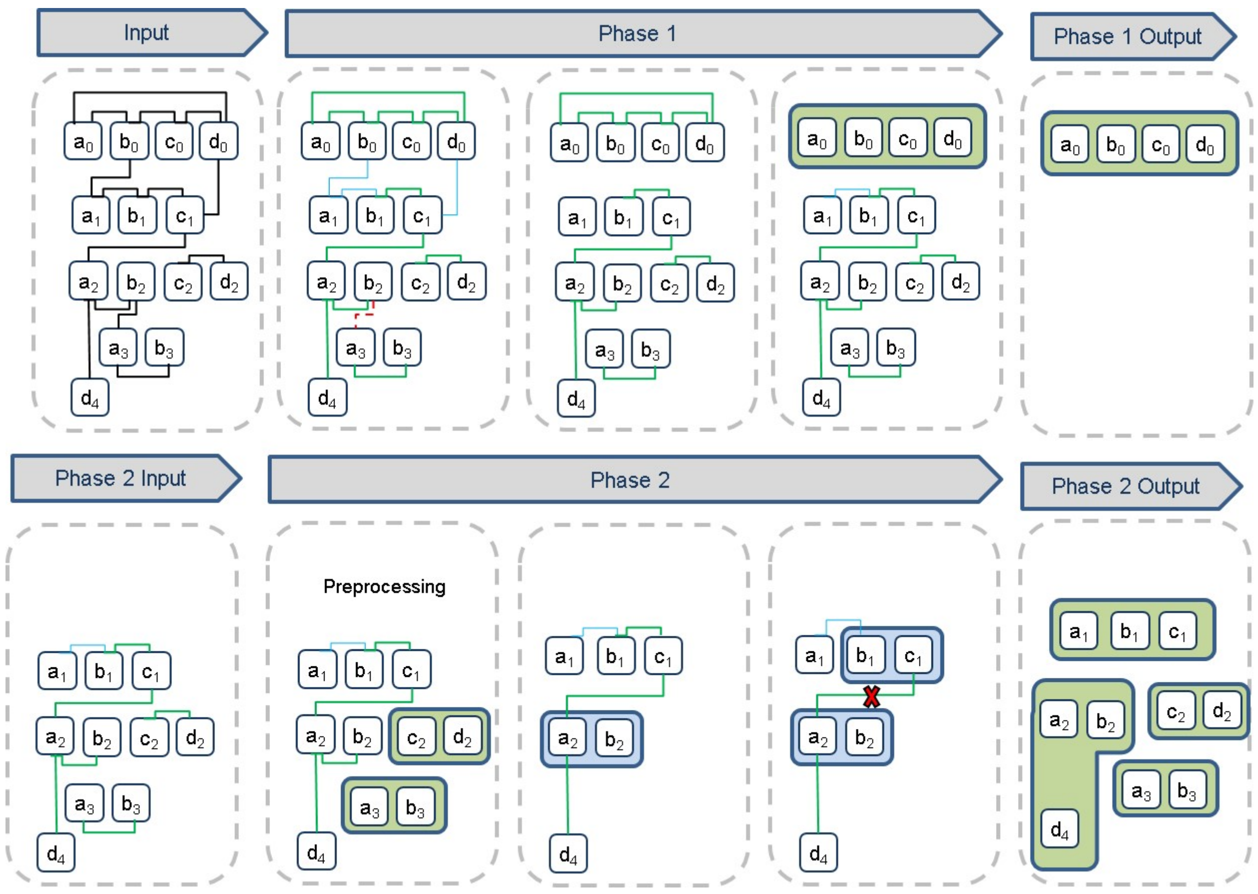


Figure 5. Running example processed with CLIP clustering

union of all cluster sets $\mathcal{CS}_i$ determined in this way for the different components combined with the previously determined clusters in phase 1 form the final output of CLIP (line 16).

In the example of Figure 5, 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. Then the link between b_1 and c_1 is selected and these entities are merged into one cluster as well. Then the link from c_1 to a_2 is taken. The clusters on the ends of this link are not compatible because both have one entity from *source B*. Processing all links in sorted order in the example leads to adding the entity a_1 to the cluster containing entities of index 1. Similarly, the entity d_4 is added to the cluster containing entities a_2 and b_2. The output of phase 2, together with the output of phase 1, results in five clusters. Compared to the perfect result shown in Figure 2, only three clusters (with indices 0, 1, 3) are correct while the entities with index 2 are not grouped together because they were not linked in the similarity graph due to the lossy blocking approach applied.

The clustering in the second phase is an iterative process mainly based on link priority. The initial CLIP implementation of [7] updated link priorities in each iteration thereby causing high runtimes. The optimized CLIP version, described in this section, computes the link priorities only once and thus uses static priorities in the second phase. We found out that the new approach achieves about the same result quality but leads to much lower runtimes which will be presented in Section 5.

5 Evaluation

The goal of evaluation is to comparatively evaluate the effectiveness and efficiency of the considered clustering approaches and their distributed implementations for different datasets and configurations. We first describe the used datasets from three domains and the considered configurations. We then analyze the relative match and clustering effectiveness of the clustering schemes. Finally we evaluate the runtime performance and scalability of the approaches.

5.1 Datasets and Configuration Setup

For our evaluation we use datasets from three domains for different numbers of duplicate-free sources. Table 3 shows the main characteristics of the datasets, in particular, the number of clusters and match pairs of the perfect ER result. The smallest dataset DS1 contains geographical real-world entities from four different data sources (DBpedia, Geonames, Freebase, NYTimes) and has already been used in the OAEI competition[3]. For evaluation we focused on a subset of settlement entities as we had to manually determine the perfect clusters and thus the perfect match pairs.

Table 3. The specifications of datasets.

domain	attributes	#entities	#sources	#perfect match pairs	#clusters
geography (DS1)	label, longitude, latitude	3,054	4	4,391	820
music (DS2)	title, length, artist, album, year, language	20,000	5	16,250	10,000
persons (DS3)	name, surname, suburb,	5,000,000	5	3,331,384	3,500,840
	postcode	10,000,000	10	14,995,973	6,625,848

For the two larger evaluation datasets DS2 and DS3 we applied advanced data generation and corruption tools [32] to be able to evaluate the ER quality and scalability for larger datasets and a controlled degree of corruption. DS2 is based on real records about songs from the MusicBrainz database but uses the DAPO data generator to create duplicates with modified attribute values [32]. The generated dataset consists of five sources and contains duplicates for 50% of the original records in two to five sources. All duplicates are generated with a high degree of corruption to stress-test the ER and clustering approaches. DS3 is based on real person records from the North-Carolina voter registry and synthetically generated duplicates using the tool GeCo [33]. We consider two configurations with either 5 or 10 sources each having 1 million entities; i.e. we process up to 10 million person records. Each source is duplicate-free, but 50% of the entities are replicated in all sources without any corruption. Moreover, 25% of entities are corrupted and replicated in all sources, and the remaining 25% are corrupted but present in only some sources. For the generation of corrupted records we applied a moderate corruption rate of 20%, i.e. most attribute values remained unchanged. The datasets are available on the website[4]

To generate the similarity graphs for the different datasets as the input of the clustering schemes, we experimented with a large spectrum of blocking and match configurations. Due to space restrictions, we will mostly report results only for the default configurations specified in Table 4 that resulted already in good match quality even without clustering. All configurations apply standard

Table 4. Default blocking and match configuration for different datasets.

dataset	blocking key	similarity functions	match rule
DS1	prefixLength1(label)	sim1: Jarowinkler (name) sim2: geographical distance	sim1 $\geq \theta$ & sim2 $\leq$ 1358 km
DS2	prefixLength1(album)	sim1: 3Gram (title)	sim1 $\geq \theta$
DS3	prefixLength3(surname)	sim1: Jarowinkler (name) sim2: Jarowinkler (surname) sim3: Jarowinkler (suburb) sim4: Jarowinkler (postcode)	sim1$\geq$ 0.9 & sim2 $\geq$ 0.9 & sim3 $\geq \theta$ & sim4 $\geq \theta$

blocking with different blocking keys. The match rules specify the conditions when a pair of entities is considered a match. As shown in Table 4, we use different similarity functions (string similarity

[3] OAEI 2011 IM: http://oaei.ontologymatching.org/2011/instance/
[4] https://dbs.uni-leipzig.de

functions or geographical distance) to compute attribute similarities and require the similarities to reach or exceed a minimal fixed or variable similarity threshold θ.

5.2 Match Quality of Clustering Approaches

To evaluate the ER quality of our clustering results we use the standard metrics precision, recall and their harmonic mean, F-Measure. These metrics are determined by comparing the computed match pairs (derived from the computed clusters assuming that all entities in a cluster match) with the perfect match results.

In Figure 6, we compare the obtained precision, recall and F-Measure results for the eight clustering schemes, different similarity thresholds θ and our three datasets using the default configurations from Table 4 to determine the initial similarity graphs. We also include the results

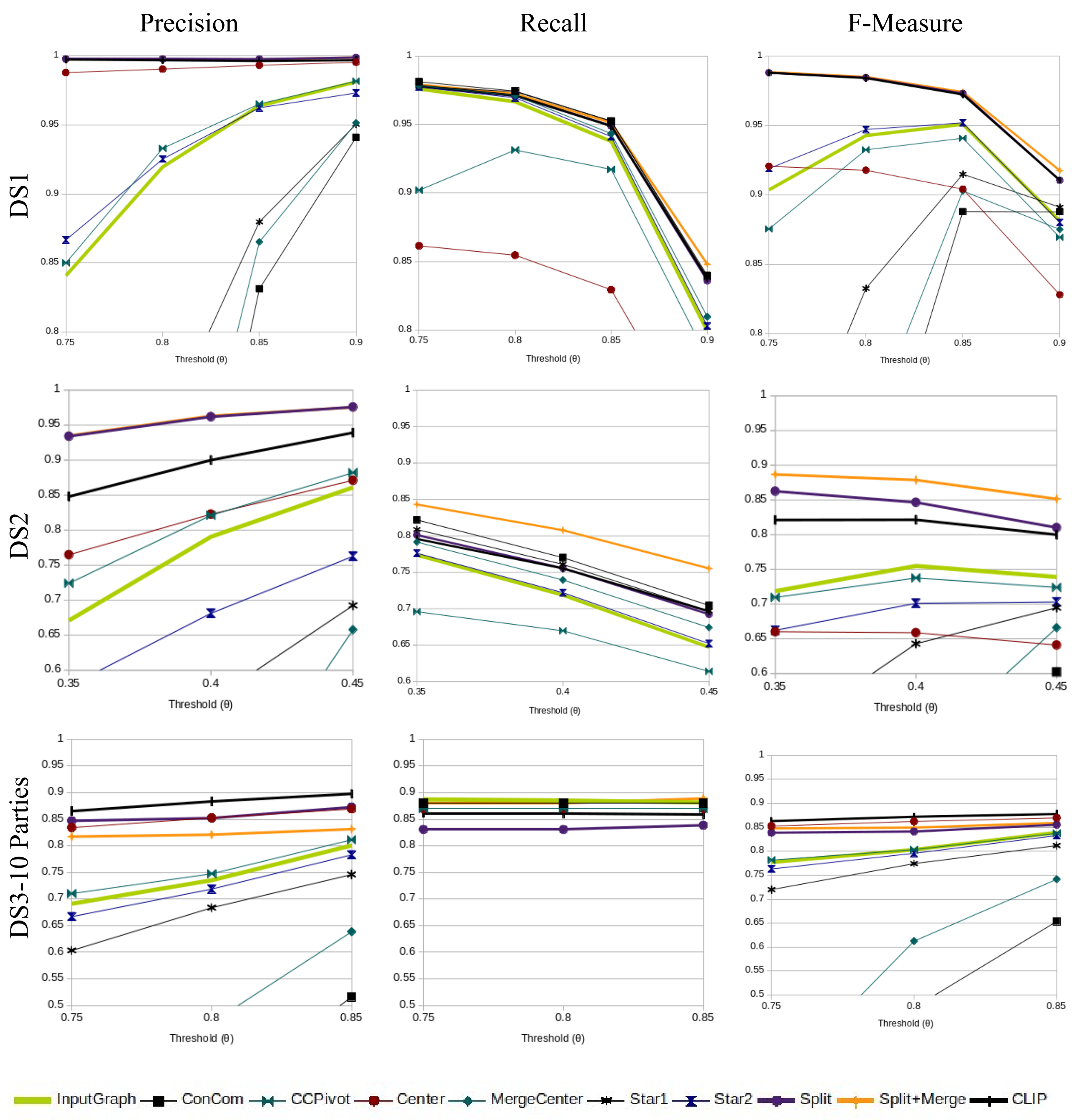


Figure 6. Match quality of clustering-based ER approaches.

obtained already with the similarity graphs used as input to the clustering schemes, although these graphs only contain links, but no clusters. Furthermore, we show the results for a SplitMerge variation called *Split* that leaves out the merge phase for faster processing. We observe that for DS1 and DS3 most clustering schemes achieve a relatively high F-Measure of more than 0.9 (DS1) and 0.8 (DS3) for the considered θ range between 0.75 and 0.9. By contrast, for the noisy data records of DS2 we had to lower the similarity thresholds to values between 0.35 and 0.45 and still could mostly not exceed the quality of the input similarity graph (with a maximal F-Measure of about 0.75) underlining that DS2 represents a more difficult match problem than DS1 or DS3. For SplitMerge, we also experimented with different values for the split and merge thresholds and we found that the split threshold should be chosen lower than the similarity threshold θ so that clusters are only split when there are links with a low similarity. By contrast the merge threshold should be higher than θ so that only very similar clusters should be merged. The shown results refer to a fixed setting per dataset, e.g. a split threshold of 0.4 and a merge threshold of 0.8 for DS1.

Comparing the clustering schemes, we observe that there are substantial differences in their relative match quality. Connected Components reaches the lowest F-Measure for all datasets and almost all threshold values because it suffers from very poor precision values. Merge Center shows a similar behavior in terms of poor precision and F-Measure, indicating that the merging of clusters can often lead to wrong cluster decisions. From the other previously known ER clustering schemes (CCPivot, Center, Star-1, and Star-2), Star-1 has the lowest F-Measure especially for lower values of the similarity threshold values. The other approaches, Center, Star-2 and CCPivot, are superior although they can exceed the F-Measure of the input graph in only few cases (Star-2 for DS1, Center for DS3). The better quality of Center comes from its initial focus on edges with high weights thereby ignoring edges with lower similarity. Star-2 is better than Star-1 since its degree-based selection of cluster centers is based on a high degree of similarity to neighbors rather than only the number of neighbors. CCPivot improves precision over the input similarity graph but suffers from lower recall so that F-Measure is not improved over the similarity graph.

By contrast, the two newly introduced algorithms, CLIP and SplitMerge (as well as Split), achieve excellent match quality and outperform all previous algorithms (and the input similarity graph) in terms of precision and F-Measure for all three datasets. CLIP generally reaches the best precision due to its ignorance of weak links making it effective even for low similarity thresholds as necessary for low data quality. The recall of CLIP, SplitMerge and Split is also among the best values achieved, especially for SplitMerge which is based on connected components and where the final merge phase helps to find additional links. 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. Comparing Split and SplitMerge, SplitMerge always achieves a slightly better F-Measure because its merge phase leads to a better recall than for Split that more than outweighs a somewhat reduced precision. For DS2, Split resp. SplitMerge are significantly better than CLIP and the other approaches due to a high precision resp. recall while CLIP outperforms SplitMerge for DS3 due to a better precision.

These observations are confirmed by Figure 7 showing the average F-Measure results of the clustering schemes over all threshold configurations. The vertical lines show the F-Measure spread between the minimal and maximal value for the different threshold values used to determine the input similarity graphs. We again observe the low and highly variable match quality of connected components and MergeCenter. By contrast, the remaining algorithms including the top-performing SplitMerge and CLIP algorithms are more robust and achieve much better F-Measure values. Interestingly, the Split approach alone achieves almost the same high F-Measure than SplitMerge. SplitMerge always achieves the same or better recall and F-Measure than Split, but the additional gains in F-Measure are small (at most 2% for DS2). CLIP is similarly effective as Split and SplitMerge, but it is easier configurable since it does not require the specification of additional similarity thresholds for splitting and merging.

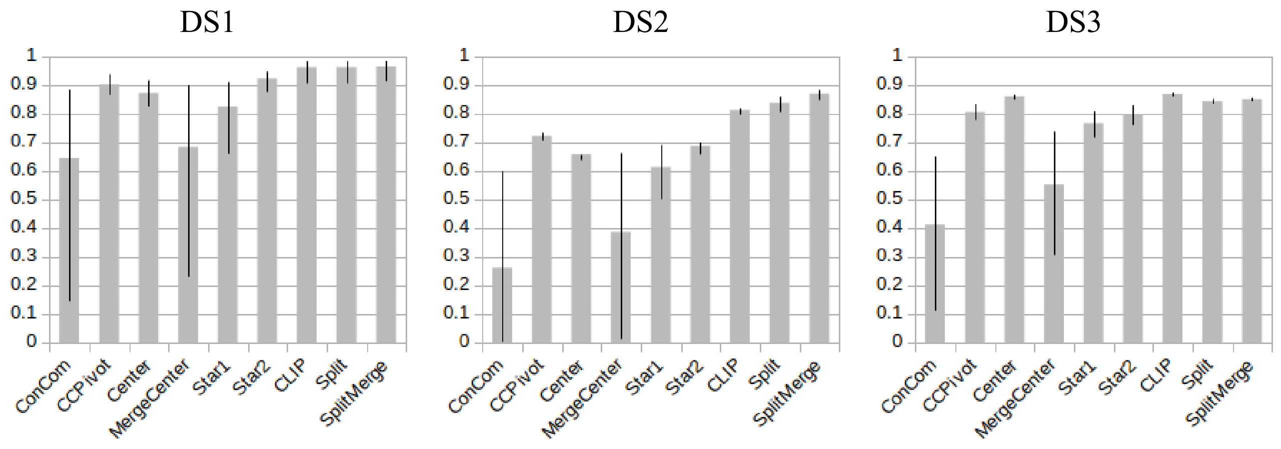


Figure 7. Average F-Measure results with range between minimal and maximal values

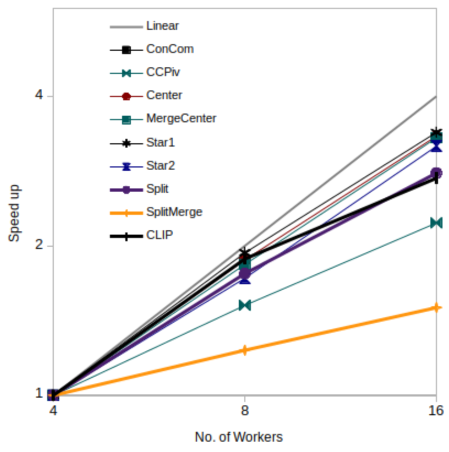
5.3 Runtimes and Speedup

We determine the runtimes of the clustering algorithms on a shared nothing cluster with 16 worker nodes. Each worker consists of an E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM, two 4 TB SATA disks and runs openSUSE 13.2. The nodes are connected via 1 Gigabit Ethernet. Our evaluation is based on Hadoop 2.6.0 and Flink 1.1.2. We run Apache Flink standalone with 6 threads and 40 GB memory per worker. In our experiments, we vary the number of workers by setting the parallelism parameter to the respective number of threads (e.g. 4 workers correspond to 24 threads). The runtime of all algorithms is measured for the largest dataset DS3 with 5 and 10 parties applying the configuration from Table 4 with $\theta = 0.80$. The DS3 input datasize is thus doubled for 10 parties compared to 5 parties. We only evaluate the runtimes for the clustering algorithms since the time to determine the similarity graphs is the same for all clustering approaches. Some clustering approaches could not be executed for 1 or 2 workers only due to high memory requirements. We thus evaluate the runtimes for configurations between 4 and 16 workers.

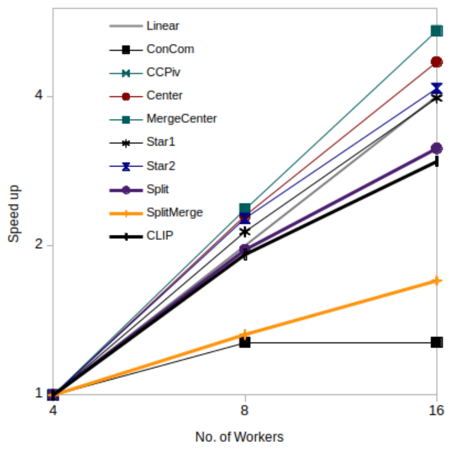
Table 5 shows the measured runtimes for the two DS3 datasets. The increased dataset size for 10 parties leads to higher runtimes for all algorithms although to different degrees. As expected, the fastest runtimes are achieved by the simple Connected Components approach. By contrast, CCPivot and SplitMerge have the worst runtimes due to large memory requirements and a high message overhead for iterative processing. CCPivot even suffered from out-of-memory errors and could only be executed for 16 workers for the bigger dataset (10 parties). Table 5 also shows the runtimes for Split, i.e. SplitMerge without the final merge phase, which achieved already a top match quality (Figure 7). These runtimes are much faster than for SplitMerge and among the fastest of all algorithms. This shows that the final merge phase is the main performance bottleneck of

Table 5. Runtimes for clustering schemes (seconds)

dataset	DS3 - 5 parties			DS3 - 10 parties		
#workers	4	8	16	4	8	16
ConCom	51	57	55	101	79	79
CCPivot	1530	1008	688	-	-	1303
Center	390	208	117	1986	864	423
MergeCenter	640	349	194	3767	1592	695
Star-1	288	149	85	783	367	197
Star-2	214	124	67	720	317	173
Split	255	145	86	873	445	278
SplitMerge	1754	1423	1168	4792	3618	2819
CLIP	190	101	69	674	351	228



a. Speedup (DS3-5 parties)



b. Speedup (DS3-10 parties)

Figure 8. Runtimes and speedup

SplitMerge since it requires the similarity computation for a large number of cluster pairs and an expensive iterative merge processing. CLIP with the new implementation is even faster than Split and thus among the fastest algorithms. The old, iterative version of CLIP needed about 5000 s with 16 workers for DS3 with 10 parties [7] so that the new implementation improves runtimes by about a factor 20 for this dataset.

Except for connected components, all algorithms can reduce their runtimes by applying more workers, especially for the larger dataset with 10 parties. Figure 8 shows the resulting speedup values. For DS3 with 5 parties, most algorithms except the iterative CCPivot and SplitMerge approaches achieve an almost linear speedup. By contrast, the high-quality approaches Split and CLIP scale well for this dataset.

For the bigger dataset with 10 parties, speedup values are mostly even better and partly super-linear. The latter, however, is an artifact for the slower algorithms like Merge Center that perform poorly for 4 workers because of memory bottlenecks (its runtime for 4 workers is almost 6 times higher for 10 parties than for 5 parties). The substantially increased aggregate memory capacity for 8 and 16 workers thus enabled super-linear runtime improvements but without reaching the absolute runtimes of fast algorithms like Star-2. Again, SplitMerge scales poorly due to the overly expensive merge phase while Split and CLIP achieve both low absolute runtimes and good speedup.

The high runtimes for SplitMerge (and CCPivot) are heavily influenced by the underlying Flink and Gelly systems and its approaches for iterative processing leading to high memory and communication overhead. We are investigating possible performance optimizations to make the approaches more scalable.

6 Conclusions and Outlook

We presented a new scalable entity resolution (ER) framework called FAMER supporting the parallel linking and clustering of entities from multiple sources. The parallel execution of ER workflows is based on the Big Data framework Apache Flink. For entity resolution, FAMER first builds a similarity graph linking similar entities from all sources and then applies clustering to group together matching entities. For parallel clustering we currently support eight approaches that have been comprehensively evaluated for datasets from three domains. The evaluation showed that the clustering approaches CLIP, SplitMerge and Split (SplitMerge without merge phase) achieve a high match quality that is clearly superior to other previously known ER clustering schemes. In particular, they ensure source-consistent clusters with at most one entity per source as required for duplicate-free sources. Unfortunately, the current implementation for SplitMerge is expensive and not yet scalable to large datasets. By contrast, both Split and CLIP achieve high match quality and good execution times and scalability making them good default schemes for multi-source entity clustering.

We are currently investigating performance optimizations of the SplitMerge algorithm to make it more scalable. We have also started to investigate incremental clustering approaches, where entity clusters are incrementally extended for new entities and new datasets [34]. 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.

Acknowledgement

This work was partly funded by the German Federal Ministry of Education and Research within the project Competence Center for Scalable Data Services and Solutions (ScaDS) Dresden/Leipzig (BMBF 01IS14014B) and by the German Research Foundation under grant number RA 497/19-2.

References

[1] P. Christen, *Data matching: concepts and techniques for record linkage, entity resolution, and duplicate detection*. Springer, 2012.

[2] H. Köpcke and E. Rahm, “Frameworks for entity matching: A comparison,” *Data & Knowledge Engineering*, vol. 69, no. 2, pp. 197–210, 2010. [Online]. Available: https://doi.org/10.1016/j.datak.2009.10.003

[3] L. Kolb, A. Thor, and E. Rahm, “Dedoop: Efficient deduplication with Hadoop,” *PVLDB*, vol. 5, no. 12, pp. 1878–1881, 2012. [Online]. Available: https://doi.org/10.14778/2367502.2367527

[4] E. Rahm, “The case for holistic data integration,” in *Proc. Advances in Databases and Information Systems (ADBIS)*. Springer LNCS, pp. 11–27, 2016. [Online]. Available: https://doi.org/10.1007/978-3-319-44039-2_2

[5] M. Nentwig, A. Groß, and E. Rahm, “Holistic entity clustering for linked data,” in *IEEE ICDMW*, 2016. [Online]. Available: https://doi.org/10.1109/icdmw.2016.0035

[6] M. Nentwig, A. Groß, M. Möller, and E. Rahm, “Distributed holistic clustering on linked data,” in *Proc. On the Move to Meaningful Internet Systems. OTM, Part II, Springer LNCS 10574*, pp. 371–382, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-69459-7_25

[7] A. Saeedi, E. Peukert, and E. Rahm, “Using link features for entity clustering in knowledge graphs,” in *Proc. European Semantic Web Conference (ESWC)*, pp. 576–592, 2018. [Online]. Available: https://doi.org/10.1007/978-3-319-93417-4_37

[8] A. Saeedi, E. Peukert, and E. Rahm, “Comparative evaluation of distributed clustering schemes for multi-source entity resolution,” in *Advances in Databases and Information Systems*. Springer, pp. 278–293, 2017. [Online]. Available: https://doi.org/10.1007/978-3-319-66917-5_19

[9] M. Nentwig, M. Hartung, A.-C. Ngonga Ngomo, and E. Rahm, “A survey of current link discovery frameworks,” *Semantic Web*, vol. 8, no. 3, pp. 419–436, 2017. [Online]. Available: https://doi.org/10.3233/sw-150210

[10] G. Papadakis, J. Svirsky, A. Gal, and T. Palpanas, “Comparative analysis of approximate blocking techniques for entity resolution,” *Proc. VLDB Endowment (PVLDB)*, vol. 9, no. 9, pp. 684–695, 2016. [Online]. Available: https://doi.org/10.14778/2947618.2947624

[11] A.-C. Ngonga Ngomo and S. Auer, “LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data,” in *Proc. International Joint Conferences on Artificial Intelligence (IJCAI)*, pp. 2312–2317, 2011.

[12] L. Kolb, A. Thor, and E. Rahm, “Load balancing for MapReduce-based entity resolution,” in *Proc. IEEE 28th Int. Conf. on Data Engneeing (ICDE)*, pp. 618–629, 2012. [Online]. Available: https://doi.org/10.1109/icde.2012.22

[13] L. Kolb, A. Thor, and E. Rahm, “Multi-pass sorted neighborhood blocking with MapReduce,” *Computer Science-Research and Development*, vol. 27, no. 1, pp. 45–63, 2012. [Online]. Available: https://doi.org/10.1007/s00450-011-0177-x

[14] D. Mestre, C. Pires, D. Nascimento, A. deQueiroz, V. Santos, and T. Araujo, “An efficient Spark-based adaptive windowing for entity matching,” *Journal of Systems and Software*, vol. 128, pp. 1–10, 2017.

[15] L. Gagliardelli, S. Zhu, G. Simonini, and S. Bergamaschi, “Bigdedup: a Big Data integration toolkit for duplicate detection in industrial scenarios,” in *Proc. Int. Conf. on Transdisciplinary Engineering (TE2018)*, vol. 7, pp. 1015–1023, 2018. [Online]. Available: https://doi.org/10.3233/978-1-61499-898-3-1015

[16] D. Singh and C. K. Reddy, “A survey on platforms for big data analytics,” *Journal of Big Data*, vol. 2, no. 1, p. 8, 2015. [Online]. Available: https://doi.org/10.1186/s40537-014-0008-6

[17] A. Gruenheid, X. L. Dong, and D. Srivastava, “Incremental record linkage,” *Proc. on PVLDB*, vol. 7, no. 9, pp. 697–708, 2014. [Online]. Available: https://doi.org/10.14778/2732939.2732943

[18] M. Pershina, M. Yakout, and K. Chakrabarti, “Holistic entity matching across knowledge graphs,” in *IEEE Int. Conf. on Big Data*, pp. 1585–1590, 2015. [Online]. Available: https://doi.org/10.1109/bigdata.2015.7363924

[19] O. Hassanzadeh, F. Chiang, H. Lee, and R. Miller, “Framework for evaluating clustering algorithms in duplicate detection,” *PVLDB*, vol. 2, no. 1, pp. 1282–1293, 2009. [Online]. Available: https://doi.org/10.14778/1687627.1687771

[20] F. Chierichetti, N. Dalvi, and R. Kumar, “Correlation clustering in MapReduce,” in *Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining - KDD ’14*, pp. 641–650, 2014. [Online]. Available: https://doi.org/10.1145/2623330.2623743

[21] J. Aslam, E. Pelekhov, and D. Rus, “The star clustering algorithm for static and dynamic information organization.” *J. Graph Algorithms Appl.*, vol. 8, pp. 95–129, 2004. [Online]. Available: https://doi.org/10.7155/jgaa.00084

[22] E. Rahm and H. H. Do, “Data cleaning: Problems and current approaches,” *IEEE Data Eng. Bull.*, vol. 23, no. 4, pp. 3–13, 2000.

[23] H. Köpcke, A. Thor, and E. Rahm, “Learning-based approaches for matching web data entities,” *IEEE Internet Computing*, vol. 14, no. 4, pp. 23–31, 2010. [Online]. Available: https://doi.org/10.1109/mic.2010.58

[24] M. A. Rostami, A. Saeedi, E. Peukert, and E. Rahm, “Interactive visualization of large similarity graphs and entity resolution clusters,” in *Proc. 21st International Conference on Extending Database Technology (EDBT)*, 2018.

[25] M. Junghanns, A. Petermann, N. Teichmann, K. Gómez, and E. Rahm, “Analyzing extended property graphs with Apache Flink,” in *Proc. ACM SIGMOD Workshop on Network Data Analytics*, 2016. [Online]. Available: https://doi.org/10.1145/2980523.2980527

[26] M. Junghanns, M. Kießling, N. Teichmann, K. Gómez, A. Petermann, and E. Rahm, “Declarative and distributed graph analytics with GRADOOP,” *Proc. VLDB Endowment (PVLDB)*, vol. 11, no. 12, pp. 2006–2009, 2018. [Online]. Available: https://doi.org/10.14778/3229863.3236246

[27] 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*. Springer, 2017, pp. 457–505. [Online]. Available: https://doi.org/10.1007/978-3-319-49340-4_14

[28] O. Hassanzadeh and R. Miller, "Creating probabilistic databases from duplicated data," *The VLDB Journal*, vol. 18, no. 5, pp. 1141–1166, 2009. [Online]. Available: https://doi.org/10.1007/s00778-009-0161-2

[29] N. Bansal, A. Blum, and S. Chawla, "Correlation clustering," *Machine Learning*, vol. 56, no. 1-3, pp. 89–113, 2004. [Online]. Available: https://doi.org/10.1023/b:mach.0000033116.57574.95

[30] A. Gionis, H. Mannila, and P. Tsaparas, "Clustering aggregation," *ACM Trans. on Knowledge Discovery from Data (TKDD)*, vol. 1, no. 1, p. 4, 2007. [Online]. Available: https://doi.org/10.1145/1217299.1217303

[31] X. Pan, D. Papailiopoulos, S. Oymak, B. Recht, K. Ramchandran, and M. Jordan, "Parallel correlation clustering on big graphs," in *Advances in Neural Information Processing Systems*, pp. 82–90, 2015.

[32] K. Hildebrandt, F. Panse, N. Wilcke, and N. Ritter, "Large-scale data pollution with Apache Spark," *IEEE Transactions on Big Data*, p. 1, 2017. [Online]. Available: https://doi.org/10.1109/tbdata.2016.2637378

[33] P. Christen and D. Vatsalan, "Flexible and extensible generation and corruption of personal data," in *Proceedings of the 22nd ACM international conference on Conference on information knowledge management - CIKM '13*, 2013, pp. 1165–1168. [Online]. Available: https://doi.org/10.1145/2505515.2507815

[34] M. Nentwig and E. Rahm, "Incremental clustering on linked data," in *2018 IEEE 18th International Conference on Data Mining Workshops (ICDMW)*, 2018.