Universität Leipzig

Fakultät für Mathematik und Informatik
Institut für Informatik
Abteilung Datenbanken

A Distributed Hierarchical Clustering Algorithm for Multi-source Entity Resolution

Bachelorarbeit

vorgelegt von:
Lucie David

Matrikelnummer:
3752995

Studiengang:
B.Sc. Informatik

Betreuer:
Alieh Saeedi
Prof. Dr. Erhard Rahm

Leipzig, März 2021

Abstract

Entity Resolution describes the identification and aggregation of semantically equivalent objects within one or between multiple data sources. With the ever growing amount of data this task is essential to facilitate data integration and to assure high data quality. The Famer framework addresses this issue and focuses on integrating data from not only a single but numerous data sources. Alongside traditional Entity Resolution methods like blocking strategies, it applies an additional clustering step that can further improve the match quality and simplifies the match representation. Hierarchical agglomerative clustering is a commonly used clustering technique where each data point represents a cluster in the beginning and new clusters are formed by merging the most similar clusters. The traditional algorithm suffers from high computational costs and no obvious parallel structure and can therefore not often be applied on large data sets. Using the preprocessed data from the Famer framework and the concept of reciprocal nearest neighbors, this study presents a distributed implementation of hierarchical clustering based on Apache Flink that reaches good match quality results and scales well on large data sets. The algorithm is compared to seven other clustering schemes from the Famer framework using a large variety of data sets comprising clean, mixed and dirty data sources.

Contents

1 Introduction

1.1 Motivation

The International Data Corporation (IDC) estimates the globally created data to grow to 175 Zeta Byte (175 trillion GB) by 2025, expecting further expansion. Today's economy, research and also personal life relies on this massive amount of data with increasing tendency [35, p. 6]. To take the best possible advantage of the available data, e.g. by effectively applying predictive analysis or machine learning techniques or by integrating into a company's supply chain, the maintenance, integration and high quality of the data has to be ensured, for they constitute the basis to succeed in all those activities.

But even within the same organization data might be generated by different tools, stored using different data base management systems or exposed to any other condition leading to several representations of the same real-world object. The author of this study could for instance not only be referred to as Lucie David but also as L. David or David, Lucie and moreover there may be other authors publishing under the same name. So before further utility can be extracted from the massive amounts of accessible data, one of the core activities to achieve high data quality and facilitate data integration is the task called Entity Resolution (ER). Also known as data-deduplication, record-linkage, object matching or link discovery, ER aims to identify semantically equivalent objects within one or across data sources. While this presents a large variety of applications in e-commerce, health care, geographical and public data as well as many other domains, the challenges associated with ER are equally big as its uses [41, p. 1–6]. The commonly addressed Big Data V's, such as *Volume*, *Variety*, *Velocity* and *Veracity* also apply to the field of ER. Besides the sheer massive amount of data accompanied by an increasing complexity of networks and semantic relationships (*Volume*), other issues include loose structuredness or extreme structural diversity due to schema variations (*Variety*), data streams of entities that need to be integrated (*Velocity*) or missing values, abbreviations and data corruption (*Veracity*) [5, p. 4].

Although Entity Resolution has been widely addressed and investigated in the past [5, 20, 34, 41], the steadily growing amount of data and data sources requires further focusing on the Big Data challenges and developing large-scale solutions that allow to effectively parallelize ER-task on multiple processors.

So-called blocking techniques already address the data's *Volume* and are able to significantly reduce the number of necessary object comparisons by pre-grouping records based on chosen attribute values. [32]. However, not only the amount of data within data sources increases, but also the number of data sources themselves. The application of an additional clustering phase within traditional ER-pipelines addresses the problem of multi-source ER, producing two main advantages: Firstly, most previous ER approaches determine binary match mappings between the data sources but using a cluster representation instead has several benefits regarding data maintenance and integration. The number of binary match mappings to maintain is growing quadratically with the number of sources, leading to poor scalability since each binary match mapping is a heavy weight object that stores all pair-wise matches be-

tween two data sources. A cluster where all the records within are assumed to match is a more compact representation that also simplifies the fusion of matching entities by combining the attribute values of different cluster members and facilitates the incremental integration of entities from additional data sources by comparing them with the previously discovered clusters [34, p. 19]. Secondly, a good clustering algorithm is able to improve the match quality by detecting wrong links and by inferring undetected links through completing the transitive closure within a cluster [37, p. 281].
The ER-framework *Famer* focuses on multi-source Entity Resolution and already provides several clustering techniques, each of them showing different behavior regarding suitability and scalability [37, 38]. Since each clustering scheme leads to different results depending on the characteristics of the input data or the Famer input specifications, the implementation of additional clustering methods within the framework extends its flexibility and applicability regarding various problem domains, addressing Big Data *Variety* in a broader sense.
Hierarchical clustering has previously been applied in several domains such as document classifcation [44], recent fields in bioinformatics [1] and environmental research [11] and even in context of the current COVID-19 pandemic [42], but relatively little work has been done regarding its parallelization. In the context of the Famer framework this task is simplified due to undertaken preprocessing steps, including similarity calculations and match classification prior to clustering. The implementation and evaluation of a distributed hierarchical clustering algorithm as part of the Famer framework will therefore be the subject of this study.

1.2 Research Objectives

The main contributions of this thesis are the following:

(1) An analysis of related approaches to parallelize hierarchical clustering algorithms and the examination of their use for the Famer framework.

(2) The distributed implementation of three hierarchical agglomerative clustering variants based on Apache Flink and their integration into the Famer framework.

(3) A comparison of the newly implemented hierarchical clustering algorithm to the already existing clustering methods from Famer using different real-life and synthetically corrupted data sets, considering match quality and scalability for a different number of machines and data sources.

1.3 Organization

This thesis is organized as follows: Chapter 2 covers the necessary theoretical background, briefly introducing basic Entity Resolution concepts and the large-scale ER framework Famer including its workflow as well as the software components it is based on. It further covers the serial hierarchical clustering method, outlines the decisions that have to be made before applying the algorithm and formally defines the three implemented hierarchical clustering

variants. Chapter 3 follows up with an analysis of related work regarding the application of clustering methods in the context of ER and theoretical and practical research results on distributed hierarchical clustering from other domains. Chapter 4 describes the theoretical concept and the implementation of the distributed algorithm, including a vertex-centric implementation as well as a naive version based on partitioning. The former is evaluated in chapter 5 where the empirical results regarding match quality and speedup are presented, analyzing the impact of a newly introduced input parameter and the match quality output dependent on the data set characteristics. Eventually, chapter 6 summarizes the results and suggests possible subjects of future work based on this thesis.

2 Theoretical Background

This chapter will briefly describe the basic concepts of Entity Resolution and its activities introducing the Famer framework as an exemplary large-scale ER-tool. It outlines the Famer architecture and the role of this study within the Famer workflow. It will further introduce the concept of hierarchical clustering and covers the three hierarchical clustering variants that have been implemented within the scope of this study. Their characteristics and behavior will be explained before illustrating their application within the serial hierarchical clustering algorithm. Additionally, the vertex-centric programming model will be presented as it constitutes the basis for the distributed implementation, making it a necessary tool to achieve scalability for large data sets.

2.1 Basics of Entity Resolution

The term *entity* describes any real-world object, such as a person, a place, a product or a publication. Each entity can be characterized by its *attributes*, e.g. a name, a serial number, geographical coordinates or a publishing date. The collection of attribute values for a specific entity is called a *reference*. Using these terms, *Entity Resolution* can be defined as the process of determining whether two references refer to the same entity or to different real-world objects. This pair-based definition can further be extended by systematically and successively applying it to larger sets of references and therefore identifying and aggregating all references to the same entity by forming groups or clusters of entities [40, p. 1].
A set of references, each holding a unique identifier, is called an *entity collection*. A naive approach to identify matching objects would be the comparison of all references and attribute values to each other, which is obviously not feasible regarding the challenges of Big Data. Therefore, Entity Resolution usually comprises several successively executed tasks that transfigure an entity collection into a set of *resolved entities* [5, p. 5].

Depending on the characteristics of the given input entity collection, ER-problems can be categorized as follows [5, p. 5]:

(1) *Clean-Clean ER* focuses on determining matching entities between two overlapping but individually clean, i.e. duplicate-free, entity collections.

(2) *Dirty ER* aims to identify duplicates within a single entity collection.

(3) *Multi-source ER* is used when the input comprises more than two entity collections, which is the case for this study.

In almost all previous work on multi-source ER, the individual data sources are assumed to be clean, so that only matching entities between but not within the data sources have to be found, this will be referred to as *Multi-source Clean ER (MSC ER)*. Recent research has been conducted to further support the application of multi-source ER on dirty or a mixture of clean and dirty data sources, called *Multi-source Clean-Dirty ER (MSCD ER)* [23].

This study will cover both cases by analyzing the clustering quality on three MSC data sets as well as an additional MSCD data set which comprises clean and dirty data sources in different portions.

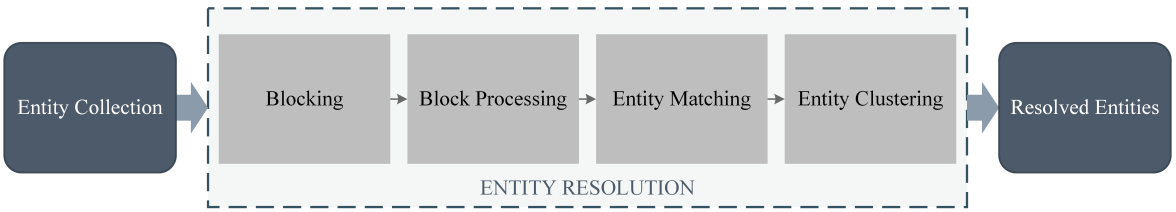


Figure 2.1: Generic End-to-End Workflow for ER, adopted from Christophides et. al. [5, p. 5]

A generic end-to-end ER workflow including the commonly applied ER-tasks independent of the underlying type of ER-problem was developed by Christophides et. al. [5] and is shown in figure 2.1. *Blocking* is usually the initial activity as it helps breaking down large volumes of data into blocks so that entities from different blocks are unlikely to match. To further avoid redundant comparisons *Block Processing* methods can be applied before actually calculating similarities and deciding about *Entity Matching*. *Clustering* as the final tasks is then able to infer indirect matching relations among the detected match pairs.
The individual activities will be described in more detail on the basis of the Famer framework whose workflow includes each of the pictured tasks.

2.2 The Famer Framework for Multi-source Entity Resolution

Famer (FAst Multi-source Entity Resolution) is an ER approach that supports the clustering of matching entities and utilizes blocking techniques and distributed processing to make it a feasible tool for large-scale ER [37]. By allowing customized input specifications and a variety of clustering techniques it can be applied to entity collections from various domains including *MSC* and *MSCD ER* problems.
The Famer workflow consists of two main parts and is illustrated in figure 2.2, the new clustering scheme is highlighted red. The first phase comprises three sub-tasks producing a so-called similarity graph as an intermediate match result, which is the input for the second phase that produces the final clustering output so that all entities within the same cluster match.

Blocking The first step of the Famer workflow consists of preprocessing the records from multiple data sources within a *blocking phase*. All records are partitioned into groups according to a blocking key specified in the configuration input. This aims to reduce the number of pair-wise comparisons by only comparing entities from the same block. For instance, only publications published in the same year or products with the same serial number are compared to each other. Famer supports several blocking strategies such as *Standard Blocking*, *Sorted Neighborhood Blocking* and *Single-* and *Multipass Blocking* [37, p. 280–281].

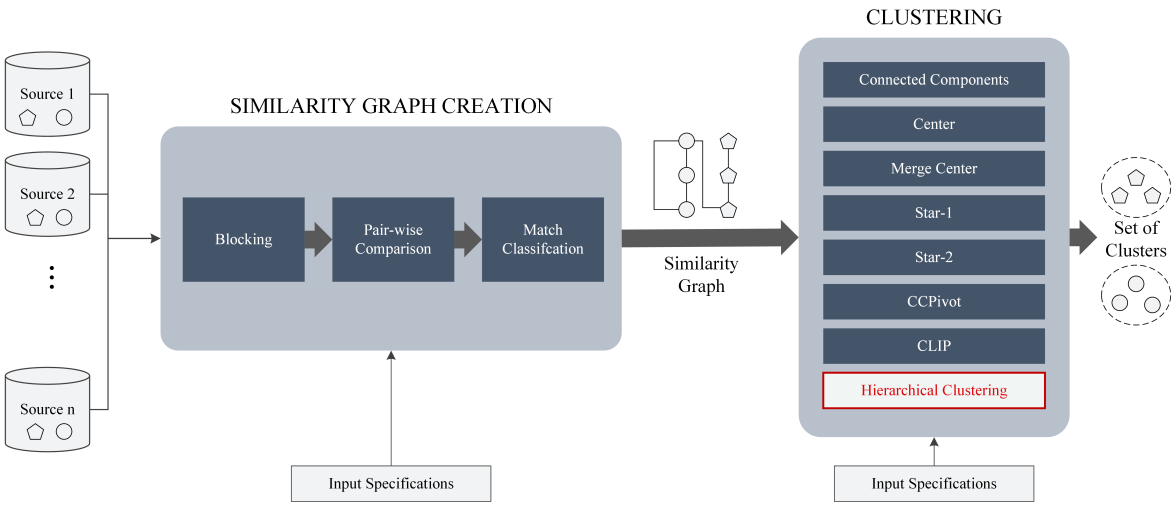


Figure 2.2: The Famer Workflow including the new Clustering Scheme, adopted from Saeedi et. al. [37, p. 281]

Pair-wise Comparison Afterwards, all entities from the same block are compared to each other. Similarity values are calculated using the functions and attributes specified in the configuration input. A similarity function *sim* maps a pair of entity descriptions (e_i, e_j) to a value between 0 and 1, indicating their degree of likeness. Examples are string similarity functions like *Jarowinkler* or *3Gram* on string attributes, e.g. a person's name, a book's title or an address or numerical distance functions on numerical attributes like a person's age, a publishing date or geographical coordinates. A combination of several similarity functions is possible too.

Match Classification Given the resulting similarity values, a *match classification step* decides about the final output of the first phase. This is done by applying a simple match function M, which maps each pair of entity descriptions (e_i, e_j) to $\{true, false\}$, where $M(e_i, e_j) = true$ means, that e_i and e_j are considered a match and vice versa. Given the threshold value θ and the similarity function *sim* from the configuration input, the match function can be described as

$$M(e_i, e_j) = \begin{cases} true, & \text{if } sim(e_i, e_j) \geq \theta, \\ false, & \text{otherwise} \end{cases} \tag{2.2.1}$$

Similarity Graph Output The outcome of the first part is a so-called *similarity graph* where each vertex represents an entity and each edge indicates that its source and target vertex are considered a match. Their degree of similarity is represented by the weight of the edge as a value between 0 and 1 [37, p. 280–282].
Famer is implemented using a new extension for graph analytics called Gradoop [15], which supports the *Extended Property Graph Model (EPGM)* and is useful to store the similarity graph with its vertex and edge properties.
An example similarity graph is shown in figure 2.3, entities with the same index are assumed perfect match pairs.

In the case of MSC ER, links within the similarity graph can be categorized according to the concept of *link strength* proposed by Saeedi et. al. [38, p. 4–5]. It utilizes the fact that a vertex v_a from source A may have more than one link to another source B but it can be assumed that there is at most one correct match for v_a with respect to source B (as both A and B can not include any matches).

Definition 2.1. Let $G = (V, E)$ be a similarity graph, A and B clean data sources with $A, B \subset V$, $v_a \in A$ and $v_b \in B$ entities descriptions from different data sources and $e_{(v_a,v_b)} \in E$. The edge with the highest similarity for v_a relative to B can be determined and is called the *maximum edge*:

$$e_{max}(v_a, B) = \underset{b}{\text{argmax}}\{e_{(v_a,v_b)}\} \tag{2.2.2}$$

If an edge $e_{(v_a,v_b)}$ is the maximum edge for both vertices at either side of it, the edge is classfied as *strong*:

$$e_{(v_a,v_b)} = e_{max}(v_a, B) \wedge e_{(v_a,v_b)} = e_{max}(v_b, A) \tag{2.2.3}$$

If it is the maximum edge for exactly one of the adjoining vertices, the edge is classified as *normal*:

$$e_{(v_a,v_b)} = e_{max}(v_a, B) \veebar e_{(v_a,v_b)} = e_{max}(v_b, A) \tag{2.2.4}$$

and if none of both cases apply, it is classified as *weak*:

$$e_{(v_a,v_b)} \neq e_{max}(v_a, B) \wedge e_{(v_a,v_b)} \neq e_{max}(v_b, A) \tag{2.2.5}$$

The chapters on the parallel implementation and the evaluation will refer to this concept later on.

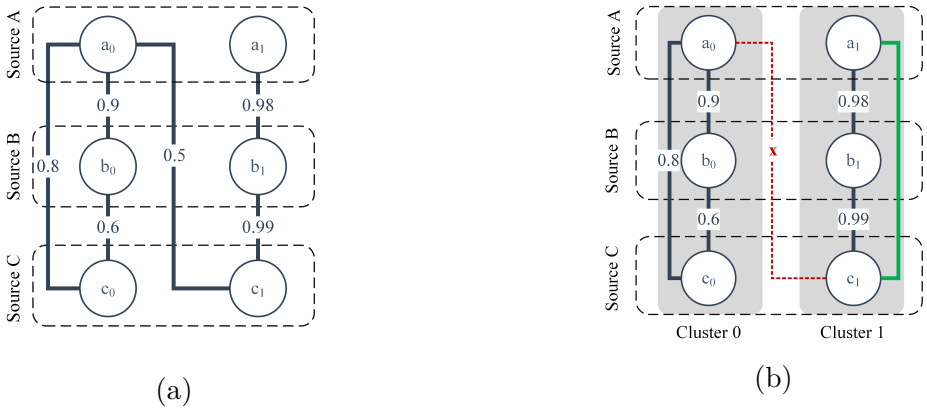


Figure 2.3: Example Similarity Graph before (a) and after (b) Clustering

Clustering Using the similarity graph generated during the previous stage, clustering aims at further refining the match classification by maximizing the intra-cluster similarity and

minimizing the inter-cluster similarity. The main benefits of this approach have already been described in the previous chapter (see section 1.1). Subfigure (b) in figure 2.3 illustrates how clustering can complete transitive closures and eliminate wrong matches from the example similarity graph by highlighting clusters in grey. The entities a_0 and c_1 have been classified as a match with regard to the similarity graph but are then decoupled as they are grouped into different clusters. Moreover, the green link connecting entities a_1 and c_1 is added as they are part of the same cluster, but have not been marked as a match in the similarity graph. Sometimes matches can not be identified during match classification e.g. due to the applied similarity functions or blocking technique. In this case clustering is particularly useful. Assuming that the individual data sources do not contain any duplicates, inferring new matches is only possible and necessary for data sets comprising more than two data sources, as clusters determined based on n clean sources are limited to a size of n. In the case of $n = 2$, inferring additional links based on the existing ones would lead to a cluster size $> n$ since one link already connects two entities.

A cluster that contains at most one entity per source is called *source-consistent*, which is a preferable characteristic if working with clean data sources, as no two entities from the same source are assumed to match. Some clustering methods may produce *overlapping* clusters where one entity belongs to more than one cluster [38, p. 5]. Figure 2.4 exemplifies the idea of source-consistency using the example similarity graph. Cluster C_0 in subfigure (a) is source-inconsistent as it contains two entities from source C. All clusters from subfigure (a) are source-consistent, including no two entities from the same source. If a cluster is source-consistent and furthermore includes an entity from each source, it is called a *complete* cluster, like cluster C_2 from subfigure (b).

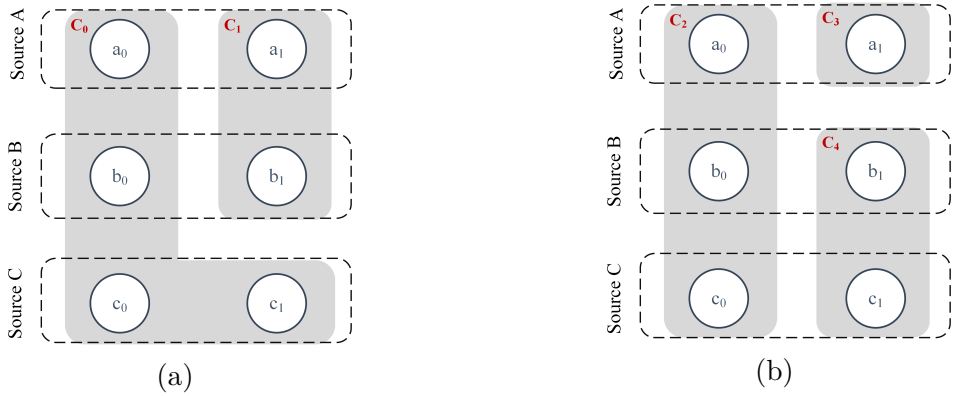


Figure 2.4: Example Similarity Graph with source-inconsistent (a) and source-consistent (b) Clusters

2.3 Vertex-centric Programming

In order to provide high scalability and distributed graph processing, Famer is implemented based on Apache Flink, a distributed stream and batch processing framework [4]. Flinks Graph API Gelly supports the vertex-centric programming model, also known as 'think like a vertex' (TLAV) model. Firstly published by Google in 2010 [25], TLAV frameworks aim at

overcoming the drawbacks of sequential graph processing algorithms, which require random access to all graph data. A vertex-centric computing framework iteratively executes a user-defined function over all vertices of a graph. Instead of the global perspective of the data employed by other distributed graph processing frameworks, programmers are encouraged to take a local view of computation and 'think like a vertex ' [27, p. 2].
Flinks Graph API Gelly supports the TLAV model with a *scatter-gather* implementation where the user-defined vertex function comprises two stages: During the *scatter phase* each active vertex produces messages that will be send to other vertices. The vertex values are then updated within the *gather phase*, depending on the incoming messages. If a vertex does not receive any messages during one iteration it will stop forwarding messages to its neighbors in the next iteration. Messages are recieved during the same iteration as they are sent. The iterations, called *supersteps*, proceed synchronously until no more messages are sent or the specified maximum of supersteps is exeeded. [16, p. 41].
The scatter-gather model has been applied successfully for all clustering algorithms implemented in Famer. It will therefore form the basis of parallelizing the hierarchical clustering algorithm which will be introduced the next section.

2.4 Hierarchical Agglomerative Clustering

Hierarchical cluster analysis comprises a group of clustering algorithms where every cluster is a partition of two clusters of which each is again hierarchically clustered itself. The resulting hierarchy is usually represented using a so-called dendrogram, a tree-like structure where each leaf is a single data point and the nodes at each level symbolize the clusters at the respective stage. Hence, the root node represents the output where all records are clustered into one single group. This hierarchy can apparently be build in two ways: Either each record is considered a cluster itself in the beginning and new clusters are formed by merging two clusters (known as bottom-up or agglomerative approach) or all records are treated as one cluster in the beginning and new clusters are formed by splitting former clusters in two (known as the top-down or divisive approach) [10, p. 14]. As there are 2^n possibilities of splitting a set of n objects, the latter approach is usually not feasible for practical applications [17, p. 49]. Therefore, this study will only cover the agglomerative method, which is succinctly described by Leskovec et. al. [24, p. 246] as follows:

Algorithm 2.1: Basic Hierarchical Agglomerative Clustering

1 Initialize each data point as a cluster
2 Pick the best two clusters to merge
3 Combine those two clusters into one cluster
4 If it is not time to stop go back to 2.

Once two clusters were merged they can not be separated during later steps. Hierarchical agglomerative clustering (HAC) does not produce overlapping clusters but assigns only one cluster ID per vertex.

The basic approach from algorithm 2.1 gives rise to the following questions:

(1) How is the similarity between the data points calculated?

(2) How are the next two clusters to merge chosen?

(3) When does the algorithm stop merging clusters?

The first question can be answered easily: Unlike in other fields of application, the HAC algorithm implemented within this study is not applied on a group data points but the graph data from Famers linking module output. Hence, the similarity values are precalculated and given in the form of the edge properties of the similarity graph.
The other two questions require more complex answers and will be subject of the succeeding sections.

2.4.1 Linkage-Strategies

The decision which clusters to select for merging is mainly determined by the chosen *linkage-strategy*. Assuming the best cluster candidates for merging are those having the highest similarity value, a technique for redetermining the inter-cluster similarities after a cluster merge has to be specified. There is a wide range of linkage-strategies addressing this computation, each of them showing a different impact on the final clustering result. This study will cover three commonly used approaches, called single-, complete- and average-linkage. They will be referred to as S-LINK, C-LINK and A-LINK, respectively. Other linkage-strategies like *median-* and *centroid-linkage* or *Ward's method* are not discussed within this study as they require a more complex calculation of inter-cluster similarites, but their definition and characteristics have been studied by many authors, such as Fung [10], Kaufmann et. al [17] or Murtagh et. al [30].

Definition 2.2. Let A and B be two clusters and $sim\colon A \times B \mapsto [0,1]$ a similarity function, then the three chosen linkage-strategies can be described as follows:

$$Sim_{S-LINK}(A,B) = max_{a\in A, b\in B}\{sim(a,b)\} \tag{2.4.1}$$

$$Sim_{C-LINK}(A,B) = min_{a\in A, b\in B}\{sim(a,b)\} \tag{2.4.2}$$

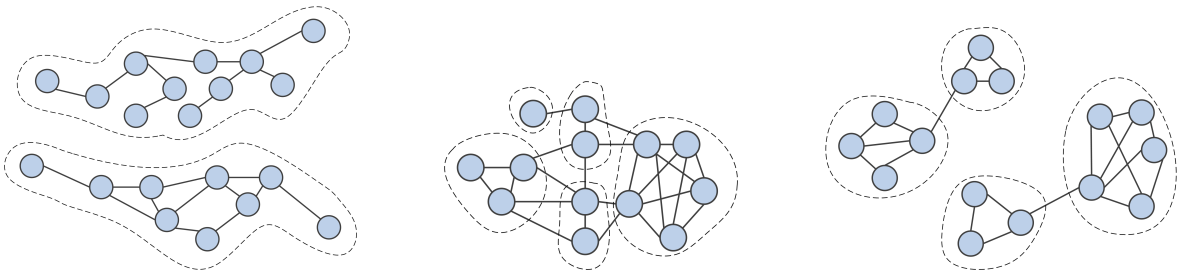
$$Sim_{A-LINK}(A,B) = \frac{1}{|A||B|}\sum_{a\in A, b\in B}\{sim(a,b)\} \tag{2.4.3}$$

[30, p. 89]

S-LINK determines the cluster similarity based on the two closest points from each cluster and is also referred to as the nearest-neighbor strategy. This method is susceptible to outliers which leads to a so-called chaining effect. Whenever two cluster come too close together even at one point only, the clusters get merged immediately (and can not be separated in later steps). This may result in larger clusters, where some cluster members are very far from each other, like in figure 2.5 (a) [17, p. 47–48].

C-LINK is also known as the furthest-neighbor strategy and the cluster similarity is determined by the two points most dissimilar from both clusters. This tends to produce very compact clusters, where each cluster member is very close to the others, like in figure 2.5 (b) and is useful to identify clusters that are not well separated [17, p. 47–48].
Note that the similarity graph may contain many entities that are not connected to each other, those will be treated with a similarity of zero. Under the assumption that there are no links between records from the same data source and clusters with a similarity of zero will never get merged, clusters formed using this linkage strategy are always *source-consistent*. This can be proven by contradiction: Assume that there is a cluster C build using the C-LINK strategy that contains two entities v, w from the same data source. Then there must have been a cluster merge between two clusters C_v and C_w with $v \in C_v, w \in C_w$ and $C = C_v \cup C_w$ where $Sim_{C-LINK}(C_v, C_w) > 0$. By definition of C-LINK the cluster similarity is the minimum similarity between all entities from both cluster. Because of the assumption of clean data sources it holds that $sim(v, w) = 0$. Therefore it is $Sim_{C-LINK}(C_v, C_w) = 0$, which leads to a contradiction.

A-LINK acts as a compromise of the other two methods by defining the cluster similarity as the average similarity over all edges connecting the two clusters. This results in clusters, where all members have roughly the same similarity to each other, like in figure 2.5 (c) [17, p. 47–48]. Again, missing edges will be treated with a similarity value of zero, leading the cluster-linkage to drop significantly if there are unconnected vertices between two clusters.



(a) elongated

(b) compact but not well separated

(c) ball-shaped

Figure 2.5: Cluster Types, adopted from Kaufmann et. al. [17, p. 48]

2.4.2 Stopping Condition

The application of HAC results in not only one but a set of clusterings, one at each level of the cluster hierarchy. Working with small data sets, the complete dendrogram and clusters at each node can easily be visualized, enabling the user to decide about the optimal clusters. Using larger data sets complicates this process and generally determining the optimal clustering from the hierarchy is not a trivial decision. Using the number of clusters as the stopping condition is usually the easiest approach, but in the context of ER this number is unknown and hence other methods have to be considered. There are several metrics, criteria and even approaches to 'learn' the optimal threshold in HAC [7, 28] but they all come at high

computational costs. To limit the scope of this theses, a new parameter, the merge threshold t, will be introduced. The algorithm terminates if the maximum inter-cluster similarity falls below t.

2.4.3 The Serial Hierarchical Agglomerative Clustering Algorithm

Now that the questions from section 2.4 have been answered, the serial HAC algorithm can be described in algorithm 2.2.

Algorithm 2.2: Serial Hierarchical Agglomerative Clustering

Data: $G = (V, E)$, threshold $t \in [0, 1]$, linkage $l \in \{S-LINK, C-LINK, A-LINK\}$

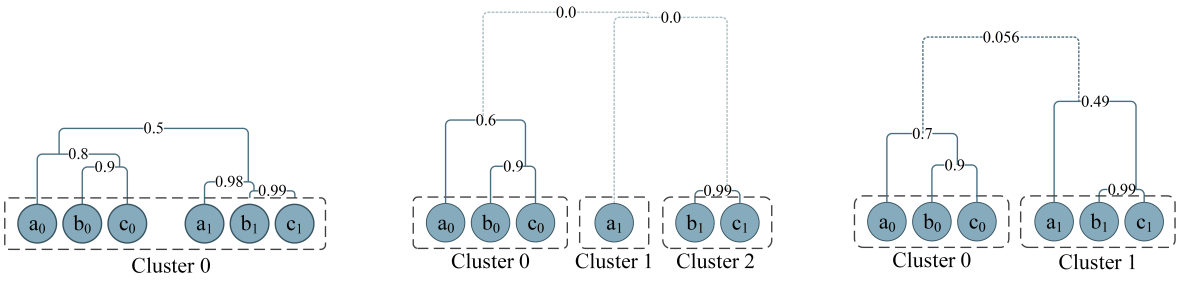
1 **Algorithm** `SerialHAC()`
2 $Q \leftarrow initPriorityQueue(E)$
3 $M \leftarrow initHashMap()$
4 **for** $v_i \in V$ **do**
5 $v_i.clusterID \leftarrow i$
6 $M.put(i, \{v_i\})$
7 **repeat**
8 $e_{(i,j)} \leftarrow Q.pop()$
9 `mergeClusters` (i, j)
`/* assume i > j, i is the new cluster ID */`
10 `updateSimilarities` (i, l)
11 **until** $e_{(i,j)}.weight < t$ *or* $Q = \{\}$
12 **Procedure** `mergeClusters`(*clusterID* c_{new}, *clusterID* c_{old})
13 **for** $v_k \in M.get(c_{old})$ **do**
14 $v_k.clusterID \leftarrow c_{new}$
15 $M.get(c_{new}).add(v_k)$
16 $M.remove(c_{old})$
17 **Procedure** `updateSimilarities`(*clusterID* c_{new}, *linkage* l)
18 **for** $i \in M.keySet \setminus \{c_{new}\}$ **do**
19 $E_{link} \leftarrow \bigcup e_{(i,c_{new})}$
20 $Q \leftarrow Q - E_{link}$
21 $allLinks \leftarrow M.get(i).size * M.get(c).size$
22 $e_{new} \leftarrow initEdge(c_{new}, i, 0)$
23 **if** $l = A\text{-}LINK$ **then**
24 $e_{new}.weight \leftarrow \frac{1}{allLinks} \sum_{e \in E_{link}} e.weight$
25 **else if** $l = S\text{-}LINK$ **then**
26 $e_{new}.weight \leftarrow \text{argmax}_k\{(e_k \in E_{link})\}$
`/* only set the edge weight if there are no unconnected vertices */`
27 **else if** $l = C - LINK \wedge |E_{link}| = allLinks$ **then**
28 $e_{new}.weight \leftarrow \text{argmin}_k\{(e_k \in E_{link})\}$
29 $Q \leftarrow Q \cup e_{new}$

In contrast to most traditional implementations that use a matrix representation for storing the cluster and vertex distances, this algorithm uses the preprocessed edges of the input similarity graph and processes them using a priority queue based on the edge weights in

descending order (line 2). To allow accessing all members of one cluster without iterating over all vertices and comparing their assigned cluster IDs, a map data structure is used that maps each cluster ID to the set of vertices that belong to the cluster (line 3).
First, each vertex is initialized with a unique cluster ID and an entry is added to the map of cluster memberships (lines 4-6), hence each vertex is considered as a cluster of its own. Now the edge with the maximum similarity value has to be found and processed, which can be done by accessing the top element of the priority queue (line 8). The clusters from both ends of the edge will be merged into one cluster (line 9) by unifying all vertices' cluster IDs from both clusters (lines 13–14) and adjusting the cluster map by removing the old cluster ID (line 16) and adding all vertices from the old cluster to the set of vertices that is mapped to the new cluster ID (line 15). Next, all edges connecting the newly formed cluster and any another cluster have to be updated according to the specified linkage strategy (line 10). This involves iterating over all other cluster IDs from the cluster map and determining all edges that connect the new newly formed cluster and the one from the current iteration (line 19). These edges will then be removed from the queue (line 20) so they can later be replaced by a single edge representing the aggregated cluster similarity. In order to retrieve this similarity, the number of all possible connections between the two clusters is calculated (line 21), which is necessary as there might be unconnected vertices, whose weights have to be included into the calculation of the new inter-cluster similarity by assuming a similarity of 0. The new edge connecting the two currently processed clusters is initiated with a weight of 0 (line 22), its new weight is then calculated according to the specified linkage strategy (lines 23 –28) and it is added to the priority queue.
This procedure is repeated until the maximum edge similarity falls below the specified threshold or until all edges have been processed (line 11). The output is a similarity graph where all vertices within a cluster have been assigned the same cluster ID and vertices from different clusters hold different cluster IDs.

For the S-LINK variant the algorithm can be even shortened by skipping the similarity update procedure in line 10, significantly improving performance as the update of the edge weights can only be calculated in quadratic runtime. This is possible because the topmost element from the priority queue will always be the edge with the highest similarity with regard to both adjoining clusters. It might happen that those have already been merged but this can easily be tested by comparing the vertices' cluster IDs prior to merging. This modified S-LINK HAC version highly resembles the idea of Kruskals algorithm [21] for computing the *Minimum Spanning Tree (MST)* of a graph and there is indeed a strong relation between the MST and the S-LINK dendrogram of a graph, which will be further discussed in terms of related work in chapter 3.

To illustrate the difference between the three linkage strategies, figure 2.6 shows the cluster hierarchies and clustering output when HAC is applied on the example similarity graph from figure 2.3 using different linkages but an equal merge threshold of 0.4. Dashed lines represent cluster merges with a similarity lower than t. In this example only A-LINK is able to determine the perfect clusters, while S-LINK only detects one large cluster and C-LINK splits the second cluster in two smaller components. Note that algorithm 2.2 calls for strict



(a) S-LINK (b) C-LINK (c) A-LINK

Figure 2.6: Cluster Hierarchies for the Example Similarity Graph using $t = 0.4$ and different Linkage-Strategies

inequality of t and the currently processed edge weight (line 11), which means that even for $t = 0$ the clusters 1 and 2 in figure 2.6b would not get merged, meaning that in this particular case, C-LINK is not able detect the correct clustering not even by changing the merge threshold configuration. S-LINK however would benefit from a higher merge threshold, the correct clustering can be retrieved when t is chosen between 0.5 and 0.8. For A-LINK on the contrary, a relatively low choice of t has a positive impact on the clustering result. Raising t above 0.49 would split the cluster containing the entities with index 1 in two and a threshold above 0.7 would cause the second cluster to split up as well. Whereas the same split would already be performed for $t > 0.6$ using C-LINK, S-LINK is able to preserve the ideal clustering until $t \leq 0.8$.

This simple toy example already underlines the different behavior regarding the three linkage-methods and points out the considerable impact of the newly introduced parameter t which will be further evaluated using real-life data sets in chapter 5.

3 Related Work

Entity Resolution has been the subject of many books, articles and surveys, including literature on the main methods and tools [20, 41] but also studies focusing on large-scale ER [5]. This chapter analyzes previously undertaken studies regarding the application of clustering methods in the context of large-scale ER and refers to a variety of distributed implementations of HAC from other domains. It intends to put this study in contrast to related ones and aims to identify possibly adaptable approaches of parallel HAC for this thesis.

3.1 Clustering for Entity Resolution

Applying distributed clustering methods for multi-source ER has mainly been studied in the context of the Famer framework [36–38]. By utilizing Apache Flink it can achieve better performance and scalability than other parallel ER-approaches that are based on MapReduce [3, 19]. Previously implemented clustering techniques in Famer include Connected Components, Center- and Merge- Center-Clustering, Correlation-Clustering and two versions of Star-Clustering, all evaluated by Saeedi et. al. [37], a new algorithm called CLIP (Cluster Repair based on Link Priority) and an approach for cluster repair based on CLIP, called RLIP (Cluster Repair based on Link Priority), also studied by Saeedi et. al. [38]. They will serve as a reference for the evaluation conducted in chapter 5 and will therefore be briefly described for a more comprehensible and traceable argumentation regarding match quality and runtime comparisons. The following descriptions are based on the implementations by Saeedi et. al.:

Connected Components is the most straight-forward clustering approach as it simply determines the subgraphs of a graph that are not connected to each other, regardless of the edge weights.

Center Clustering focuses on edges with high similarities by iteratively determining the edge with the highest similarity value and processing both adjoining vertices depending on their center status. If both vertices are unassigned, they will form a new cluster where one of them is picked center. If one of them is already chosen cluster center and the other is unassigned, the latter will be added to the cluster of the center vertex. In any other case, the edge will be ignored.

Merge Center Clustering is a modified version of the Center algorithm that merges two clusters if a vertex from one cluster is very similar to the center of another cluster.

Star Clustering is based on computing a degree for each vertex and determining the unassigned vertex with the highest degree, whose neighbors are then added to its cluster. *Star-1* calculates the degree based on the number of outgoing edges, while *Star-2* uses their average similarity degrees. This clustering approach may result in overlapping clusters.

Correlation Clustering is an optimization problem that considers a graph with positive and negative edge weights indicating the vertices similarities. The clustering is retrieved by maximizing the sum of positive edge weights within clusters plus the sum of absolute negative edge weights between the cluster. The implementation by Saeedi et. al. uses an approximate approach called *CCPivot*.

CLIP has been developed based on the concept of link strength described in section 2.2. It initially discovers clusters based on the connected components only considering strong links and starts by determining only complete clusters. It then processes the remaining connected components, additionally including normal links, and iteratively transforms them into source-consistent clusters.

Other clustering algorithms like *Markov Clustering*, *Cut Clustering* and *Ricochet Clustering* have also been evaluated but only in the context of *Dirty ER* [12].
Hierarchical clustering has mainly been applied on smaller data sets due to its runtime complexity and there is no known application of HAC in the context of multi-source ER at the time of this study. The further course of this chapter will therefore focus on parallel HAC approaches from other domains, considering them as possible bases for the distributed implementation of this study.

3.2 Distributed Hierarchical Clustering

Most prior work on parallel HAC algorithms focused on scaling the single-linkage algorithm. This results from the fact that computing the the single-linkage clustering hierarchy can be reduced to the problem of creating the *Minimum Spanning Tree (MST)*, which has been covered in many previous studies. Theoretical results on this subject have been presented by Bentley et. al. [2], where a parallel algorithm for computing the MST is introduced. Dehne et. al. [9] and Dahlhaus [6] cover the relation between the MST construction and single-linkage HAC and Olsen [31] also presents theoretical results on distributed single-linkage HAC. Practical parallel single-linkage implementations include an algorithm called SHRINK [13] which is based on SLINK, the state-of-the-art single-linkage HAC algorithm [39], an implementation based on the MST concept and build on top of Apache Spark [14] and an approximate single-link algorithm using *Locality-Sensitive Hashing (LSH)* [18].
Despite its advantages concerning runtimes, the single-link version of HAC is less interesting for this study as its clustering results coincide exactly with those from the Connected Components approach, when the latter is executed on a similarity graph with a match threshold θ equal to the merge threshold t of the single-linkage algorithm. The quality results presented in section 5.3 will further confirm this fact. Because Connected Components already achieves the overall fastest runtime results of all in Famer implemented clustering algorithms, single-linkage HAC is not expected to make any new contributions. However, for the sake of completeness and comparability it will be implemented alongside the other two HAC algorithms.

A general approach for distributed HAC derives when looking into additional studies that do not focus exclusively on single-linkage. None of the related studies considered using the vertex-centric model but rather applied the naive concept of parallelization through partitioning: The input data is split into sub data sets on which an (optimzed) sequential HAC algorithm is performed whose results are then successively merged. Whereas Jin et. al. [14] and Hendrix et. al. [13] randomly distribute the input data into equally sized subsets, other approaches aim at grouping similar data points together. This includes the pPOP algorithm, that uses axis-aligned overlapping cells [8], the concept of a *Nearest-Neighbor Boundary* based on the idea of *K-D-* and *Quadtrees* [43], the application of a divisive clustering algorithm such as *k-means* [26] or the use of LSH [18, 22].
A substantial difference between those studies and this one occurs when looking at the type of the input data. Unlike the related approaches, which process a set data points (mostly euclidean and two-dimensional) the HAC algorithm for this study is executed on the preprocessed similarity graph. Its edge values are already independent of the type of the underlying data and methods like pPOP, K-D-Trees or LSH would require to access to the data points and their properties. Partitioning methods that exclusively depend on the calculated similarity values are therefore preferable. The similarity graph itself already provides such a partition by its natural decomposition into connected components. This structure should not be ignored but used to benefit and further splitting seems only reasonable if this structure can be preserved or even refined. Another issue to consider is load-balancing: Neither using the connected components nor any of the other mentioned approaches can guarantee equally sized subsets; with the exception of a random distribution. This however would abolish the mentioned and useful graph structure which has already been achieved by preprocessing tasks prior to clustering.

Summarizing the conducted research it becomes clear that none of the previously applied methods is directly adaptable to the problem domain of this thesis. In addition to a naive approach utilizing partitioning based on connected components, the distributed version of HAC is developed using a different variant of HAC. It is based on the concept of *Reciprococal Nearest Neighbors* and enables a vertex-centric perspective of the algorithm. The idea was originally surveyed by Murtagh et. al [29] and is mentioned again later by the same author [30] where its suitability for a parallel implementation is pointed out but without reference to practical realizations.

4 Implementing Distributed Hierarchical Clustering

This section presents the developed distributed implementation of hierarchical agglomerative clustering within Famer. It will first outline the challenges encountered during parallelization of the serial HAC algorithm and proceed by describing the drawbacks of a naive solution using partitioning. Introducing the previously mentioned alternative HAC algorithm using *Reciprocal Nearest Neighbors* it will thoroughly explain how a vertex-centric version of HAC can be developed based on this concept. The implemented scatter-gather algorithm will be described in detail and its application will be illustrated using a simple example graph. All source code is available at the public GitLab repository of the Famer project [1].

4.1 Challenges of Parallelization

In spite of the simple idea that underlies the basic HAC algorithm, a parallel implementation encounters several challenges, including the inherently sequential structure and high data dependence of the HAC algorithm. Each merge strongly depends on all previous merges, because all updated similarity values have to be considered and the global maximum similarity edge has to be found. This requires access to all graph data and makes it difficult to adapt to the vertex-centric model, which is why a naive implementation based on partitioning has been developed prior to a vertex-centric one. Machines working in parallel would need to find their local maximum similarity edge before communicating this value to all the other parallel instances and determining the overall maximum value. Dividing the similarity graph into its connected components allows to skip this communication effort as the similarity between two records from one connected component to the other is always zero, as there are no links between any two connected components. Therefore, their hierarchies are build independently from each other which further avoids the need of merging the built sub-hierarchies. However, applying the serial HAC simultaneously on all connected components can not guarantee load balancing. Although many links have already been removed during preprocessing tasks in Famer, the size of the connected components might be unbalanced, depending on the characteristics of the input data. Especially in the case of dirtier data sets where a lower match threshold configuration is necessary, the similarity graph might have less and larger connected components. The overall runtime is hence determined by the largest connected component, leading to a bottleneck.

4.2 Naive Implementation

Nevertheless, the naive approach of applying the serial HAC algorithm in parallel on each connected component has been implemented and tested prior to a vertex-centric implementation. A simplified Java code snippet is shown in listing 4.1. The implementation is based

[1] https://git.informatik.uni-leipzig.de/dbs/FAMER/-/tree/master/famer-clustering/src/main/java/org/gradoop/famer/clustering/parallelClustering/hierarchicalClustering

on the *coGroup* operator provided by Apache Flinks DataSet API. It combines two data sets by first grouping each data set after a key and then joining the groups by calling a function with the two sets for each key. This is necessary since the serial HAC algorithm requires a set of vertices as well as a set of edges as the input. After applying the Connected Components algorithm on the input graph (lines 10-14) all vertices from the same connected component have been assigned the same cluster ID. Additionally, the cluster ID needs to get associated with all the edges that belong to that connected component. This is implemented in lines 17-24 using the *groupReduceOnNeighbors* operator provided by Flinks Graph API Gelly. For each edge a tuple consisting of the edge and its source vertex is created, enabling the join within the *coGroup* operator based on the cluster ID of the respective connected component.

```
1  if (removeWeakLinks) {
2    /*keep all edges with selection status > 0 (0 = weak)*/
3   DataSet<EPGMEdge> allEdgesWithSelectedStatus = ... ;
4   edges = allEdgesWithSelectedStatus.filter(new
        FilterEdgesOnSelectedStatus(0));
5  }
6
7  /*run connected components*/
8  inputGraph = inputGraph.callForGraph(
9    new ConnectedComponents(...));
10
11 /* create a data set that contains each edge from the gelly graph
       and its source vertex, this way an edge can be associated with
       the cluster id of the connected component it is part of */
12 Graph<GradoopId, EPGMVertex, Double> gellyGraph = ... ;
13 DataSet<Tuple2<Edge<GradoopId, Double>, Vertex<GradoopId,
       EPGMVertex>>> edgesWithSourceVertex =
14  gellyGraph.groupReduceOnNeighbors(new
       GellyGraphToEdgeWithSourceVertex());
15
16 /* run the serial hierarchical clustering algorithm on each
       connected component*/
17 DataSet<EPGMVertex> resultVertices = inputGraph.getVertices()
18  .coGroup(edgesWithSourceVertex)
19  .where(new EPGMVertexClusterIdSelector())
20  .equalTo(new EdgeSourceClusterIdSelector())
21  .with(new SerialClusteringOnSubgraph(linkageType, threshold));
```

Listing 4.1: Naive Implementation of Distributed HAC using Apache Flinks DataSet and Gelly APIs

As assumed, first results have shown that the approach is not feasible for large data sets due to unacceptable runtimes. An exception however, is the S-LINK version as it does not require any updates regarding the edge weights and its serial version can be executed in linear time. Runtime results can be found in appendix B. Since the aim of this study is developing a scalable solution for all three HAC varaints, the first naive approach turned out to be

yet insufficient. A further attempt to resolve this issue has been undertaken by utilizing the concept of *link strength*, outlined in section 2.2: By removing links classified as *weak* prior to clustering, the number of connected components increases while their average size decreases, leading to improved conditions for load-balancing. As *weak* links are matches that are unlikely to be correct, their removal does not adversly affect the clustering result. On the contrary, an evaluation has shown that apart from slightly better runtimes the clustering quality also improved by a noticeable degree. Detailed quality results rearding the removal of weak links can be found in appendix A. Still, the runtimes for the A-LINK and C-LINK algorithm were unaccetable for large data sets, making even this enhanced naive approach impracticable for the application in Famer.

4.3 Vertex-centric Implementation

During an analysis of related work in chapter 3 it already became clear that HAC is no obvious fit for vertex-centric computing as there exist no studies on this subject so far. To overcome the outlined challenges such as the inherently sequential structure and high data dependence of the traditional HAC algorithm, a modified version of HAC has been developed for the vertex-centric implementation. In order to enable a vertex-centric view, it utilizes the concept of *Reciprocal Nearest Neighbors (RNNs)*. The main idea and the accompanying advantages are described in the following section.

4.3.1 Hierarchical Clustering based on Reciprocal Nearest Neighbors

The definition below formally introduces the concept of RNNs:

Definition 4.1. Let $G = (V, E)$ be an undirected similarity graph, $v_i, v_j, v_k \in V$ vertices from that graph, $e_{(v_i,v_k)} \in E$ the edge connecting v_i, v_j, $sim\colon V \times V \mapsto [0, 1]$ a similarity function and $p \in [0, 1]$ some simlarity value. For v_i, the vertex on the other end of the edge with the highest similarity value is called its *Nearest Neighbor* (NN):

$$NN(v_i) = \underset{k}{\operatorname{argmax}}\{e_{(v_i,v_k)}\} \tag{4.3.1}$$

Two vertices v_i and v_j are called *Reciprocal or Mutual Nearest Neighbors* (RNNs) if the following equation holds:

$$NN(v_i) = v_j \wedge NN(v_j) = v_i \tag{4.3.2}$$

Given the two RNNs v_i, v_j and a linkage strategy $link \in \{S-LINK, C-LINK, A-LINK\}$ the following properties hold:

$$sim(v_i, v_j) > p \tag{4.3.3}$$
$$sim(v_i, v_k) < p \tag{4.3.4}$$
$$sim(v_j, v_k) < p \tag{4.3.5}$$
$$Sim_{link}(v_i \cup v_j, v_k) < p \tag{4.3.6}$$

[29, p. 356]

Formulas 4.3.3 to 4.3.5 describe the characteristics of RNNs, stating that if v_i and v_j are RNNs, there exists no other vertex v_k that is more similar to v_j or v_i than v_i or v_j respectively. The inequation 4.3.6 is known as the *reducibility property* and states that after merging v_i and v_j into one cluster, there still is no other vertex that is more similar to the newly merged cluster than v_i and v_j to each other. This enables merging all RNNs simultaneously without affecting the cluster hierarchy. Figure 4.1 illustrates the (reciprocal) nearest-neighbor relations within the example similarity graph: Here a_0 is the NN of c_0 because the edge weight connecting c_0 to a_0 is greater than the one linking c_0 and b_0. However, a_0 has b_0 as its NN and a_0 is also the NN of b_0. This makes them RNNs, indicated by red arrows.

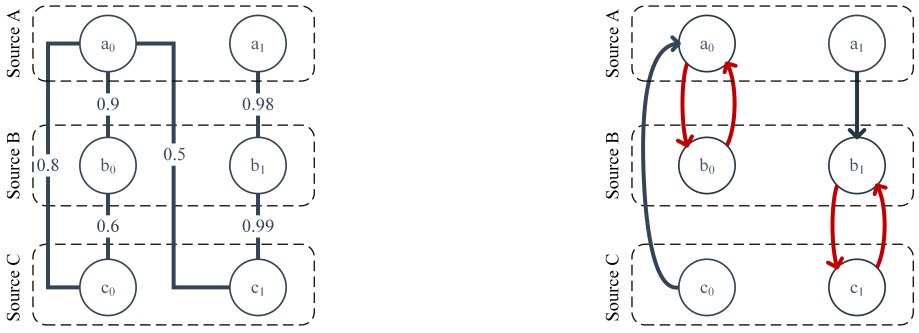


Figure 4.1: Nearest Neighbor Relations within the Example Similarity Graph

Based on the definition from above, the original HAC-RNN algorithm is outlined by Murtagh [29, p. 356] as follows:

Algorithm 4.2: Hierarchical Agglomerative Clustering based on RNNs

1 Determine all RNNs
2 Replace all RNNs by cluster centers, thus reducing the set of points
3 Redetermine the NNs and hence RNNs
4 Goto to line 2 until only one point remains

The structure of this algorithm is obviously more suitable for parallel processing, allowing simultaneous merges and moreover the detection of RNNs from a vertex-centric perspective. Instead of finding the global maximum edge which requires access to the whole graph, RNNs

can easily be determined by iterating over all in-and outgoing edges of each vertex. Furthermore does the use of a similarity graph provide an enormous advantage over the traditional HAC version, where a matrix representation is used to maintain the inter-cluster similarities. Instead of finding and updating the similarities to all other clusters, only connected clusters have to be considered as an RNN.
Still, updating the similarity values after a cluster merge involves regarding all in- and outgoing edges from all cluster members. This aggregation can hardly be parallelized and requires a centralized computation. By exclusively applying vertex-centric programming the global view of clusters and cluster memberships is lost which leads to the need of an alternative cluster representation. For previously implemented clustering methods in Famer maintaining and updating a cluster ID per vertex sufficed, but in the case of hierarchical clustering intra-cluster communication is necessary to enable computing cluster similarities based on all edges from each single cluster member. This issue is solved by applying the following concepts:

(1) Each cluster is represented by a center vertex which maintains all cluster information including cluster members and inter-cluster similarities.

(2) A center vertex keeps a list of all cluster members which in turn only store their center vertex ID.

(3) Initially all vertices are centers (as each vertex is a single cluster).

(4) After merging two clusters a new center vertex is chosen and all links are adjusted in order to only connect cluster centers, enabling them to compute the new inter-cluster similarities based on the chosen linkage method.

The message-passing scatter-gather concept provided by Flinks Gelly API allows the center vertices to communicate with all their cluster members and the other cluster centers and thereby enables them to maintain the described vertex-centric cluster structure.

Algorithm 4.3: Parallel Hierarchical Agglomerative Clustering based on RNNs

Data: $G = (V, E)$, threshold $t \in [0, 1]$, linkage $l \in \{S - LINK, C - LINK, A - LINK\}$

```
1  initVertexProperties()
2  for v_i ∈ V in parallel do
3      repeat
4          v_j ← v_i.computeNN(l)
5          if v_j.computeNN(l) = v_i then
               /* assume i > j, i is the new center                    */
6              removeConnectingEdges(i, j)
7              v_j.isCenter ← false
8              for v_k ∈ v_j.clusterMembers do
9                  v_k.clusterID ← v_i.clusterID
10                 v_i.clusterMembers ← v_i.clusterMembers ∪ {v_k}
11                 v_i.edges ← v_i.edges ∪ v_k.edges
12     until computeLinkage(v_i, v_j, l) ≤ t or v_j = null;
```

Algorithm 4.3 illustrates the parallel version of HAC based on RNNs. Exactly as in the serial HAC algorithm, the input data is given in form of a similarity graph, a merge threshold and a

linkage strategy. Prior to applying a procedure in parallel on each vertex of the input graph, vertex properties are assigned including a priority values, initial cluster IDs and the center status (line 1). Each center vertex then determines its nearest-neighbor by finding the in- or outgoing cluster connection with the highest similarity value (line 4). If there is more than one candidate the vertex is chosen depending on a random permutation of vertex priorities assigned during the prior stage. If an RNN pair is detected (line 5), the edges connecting them will be removed to prevent selecting them as RNNs again (line 6) and one of the vertices will be chosen as the new cluster center (line 7). Now the cluster center vertex has to collect all information from the cluster of the vertex it was merged with (lines 8-11). For each cluster member of the former cluster (including the old center vertex), a new cluster ID will be set (line 9), it will be added to the list of cluster members of the new cluster center (line 10) and all edges will be added to new cluster center (line 11). The algorithm terminates if no more nearest neighbor could be determined (line 12, $v_j = null$) or the similarity between two RNNs is less or equal than the chosen merge threshold.

4.3.2 Scatter-Gather Algorithm

The algorithm from the previous section will now be converted into a scatter-gather implementation. As updating the edge weights after a cluster merge is a fundamental part of the algorithm and Flinks Gelly API does not support changing edge values but only the vertex properties, an additional property is required. It stores all neighboring vertices and corresponding edge weights, which enables edge updates during the gather phases. In order to reduce memory requirements, not every single edge but a representative map entry for each connected cluster is maintained. For the S-LINK variant, only the current maximum similarity per connected cluster is stored, all other edges can be ignored. After a cluster merge the newly added edge weights are compared to the current maximum and the map entries are updated accordingly. A similar approach is applied for C-LINK, but additionally to the current minimum similarity per cluster the exact number of connecting edges (but not every single edge weight) is also stored and updated. This is necessary to detect the case of unconnected vertices between two clusters, which leads to a cluster similarity of zero. For A-LINK the sum of all similarities is kept and updated. This way the average can be easily determined by dividing the saved sum by the product of both cluster sizes. Another consequence deriving from the impossibility of edge updates is that their is no way of storing the resulting hierarchy, i.e. an output graph including the modified edge weights. The algorithms outcome is simply a graph representing the clustering at one level of the hierarchy respective to t. By supporting to use a clustering output generated in a previous execution as the input for another run with a higher threshold value, computation effort could be saved since not every cluster merge has to be recalculated but only those above the threshold of the input graph. This strategy has not been tested within the scope of study but could be implemented with minor code changes in future work.

The scatter-gather version of algorithm 4.3 consists of four phases. Firstly, RNNs are determined, then the clusters of both RNN vertices are merged. After that, the edge property of

Algorithm 4.4: Scatter-Gather Hierarchical Agglomerative Clustering

Data: $G = (V, E)$, threshold $t \in [0, 1]$, linkageType $l \in \{single, complete, average\}$

```
1  Algorithm ScatterGatherHAC()
2      initVertexProperties()
3      repeat
           /* Phase 1: determine RNNs */
4          Scatter1(Vertex v)
5          Gather1(Vertex v, MessageIterator
             messages)
           /* Phase 2: merge clusters */
6          Scatter2(Vertex v)
7          Gather2(Vertex v, MessageIterator
             messages)
           /* Phase 3: update edges */
8          Scatter3(Vertex v)
9          Gather3(Vertex v, MessageIterator
             messages)
           /* Phase 4: recalculate similarities */
10         Scatter4(Vertex v)
11         Gather4(Vertex v, MessageIterator
             messages)
12     until no more messages are sent
13 Procedure Scatter1(Vertex v)
14     if v.NN != null then
15         msg.src ← v.ID
16         msg.prio ← v.prio
17         sentMessageTo(v.NN, msg)
18 Procedure Gather1(Vertex v, MessageIterator
     messages)
19     for msg ∈ messages do
20         if msg.src == v.NN then
21             if msg.prio > v.prio then
22                 v.isCenter ← false
23                 v.clusterID ← msg.src
24                 v.centerID ← msg.src
25                 v.centerChanged ← true
26             else
27                 v.clusterMembers ←
                     v.clusterMembers ∪ msg.src
28             v.edges ← v.edges − v.edges.get(msg.src)
29 Procedure Scatter2(Vertex v)
30     if v.CenterChanged then
31         msg.centerID ← v.centerID
32         for w ∈ v.clusterMembers do
33             sendMessageTo(w, msg)
34         msg.clusterMembers ← v.clusterMembers
35         sendMessageTo(v.centerID, msg)
36         msg.messageType ← 1
37         for x ∈ v.edges do
38             sendMessageTo(x, msg)
39 Procedure Gather2(Vertex v, MessageIterator
     messages)
40     for msg ∈ messages do
41         if msg.messageType == 1 then
42             v.updateEdges(msg.centerID)
43         else
44             if v.isCenter then
45                 v.clusterMembers ← v.clusterMembers ∪
                     msg.clusterMembers
46             else
47                 v.clusterID ← msg.centerID
48                 v.centerID ← msg.centerID
49 Procedure Scatter3(Vertex v)
50     if v.centerChanged then
51         for edge ∈ v.edges do
52             msg.ID ← edge.tar
53             msg.similarity ← edge.value
54             sendMessageTo(v.centerID, msg)
55 Procedure Gather3(Vertex v, MessageIterator
     messages)
56     for msg ∈ messages do
57         v.updateEdges(msg.ID, msg.similarity)
58     if v.centerChanged then
59         v.centerChanged ← false
60 Procedure Scatter4(Vertex v)
61     if v.isCenter then
62         msg.src ← v.ID
63         msg.prio ← v.prio
64         for edge ∈ v.edges do
65             msg.clusterSize ← v.clusterMembers.size
66             sendMessageTo(edge.tar, msg)
67 Procedure Gather4(Vertex v, MessageIterator
     messages)
68     newNN ← null
69     for msg ∈ messages do
70         linkCount =
             v.clusterMembers.size ∗ msg.clusterSize
71         sim ←
             calcSim(v.edges.get(msg.src), linkCount, l)
72         if sim > t && (sim > max || sim == max &&
             msg.prio > newNN.prio) then
73             max ← sim
74             newNN ← msg.src
75     v.NN ← newNN
77
```

all involved vertices is updated and lastly the new center vertices update their similarities and thereby their nearest neighbors. Like in algorithm 4.3, vertex properties, e.g. priority values, inital NNs, edge properties and cluster IDs are initialized prior to the first scatter-gather superstep. The detailed procedure is outlined in algorithm 4.4. It repeatedly passes

the four phases until no more messages are sent, meaning that no new cluster merges can be performed.

Phase 1 During the first scatter step a message including the own vertex ID and priority is send to the nearest neighbor vertex, if there is one (lines 14-17). In the gather step, each vertex compares the sender ID of all incoming messages with its nearest neighbor ID. If a match is detected, the vertices are RNNs (line 20) and the vertex properties are adjusted exactly as previously described in algorithm 4.3 (lines 22-28).

Phase 2 If a vertex changed its center status to false during the previous phase (line 30), it now produces the following messages to complete the cluster merge: one for each cluster member informing it about the new cluster center and cluster ID (line 33), one for the new center including the new cluster members (line 35) and one for each neighbor including the new center ID, so they can update their edges accordingly, making sure all edges are only connecting center vertices before sending them to the new cluster center in the next phase (line 38). A message type is used (line 36) to distinguish the incoming messages during the gather step, where all receiving vertices update their properties accordingly (lines 40-48).

Phase 3 Now that all edges were adjusted, the old center vertices produce messages for their new cluster centers including all their neighbors and corresponding link values (lines 50-54), which then update their edge property according the received messages in the gather step (lines 56-57). The old centers can now remove their center-changed property because they do not need to produce any more messages (lines 58-59).

Phase 4 During the last phase the nearest neighbor vertices are recalculated. All centers produce a message for each connected center including their own vertex ID, priority and the size of their own cluster (lines 61-66), which is necessary for correctly calculating the new similarity during the gather step (lines 70-71). For determining the new nearest neighbor, the vertex iterates over all incoming messages (hence all connected clusters) finding the one with the highest similarity and only keeping it if the similarity is greater than the merge threshold (line 72). If this is not the case, the new NN will be null and the vertex will not produce any messages during the following phase. Thus, after each round of four supersteps the number of active vertices and clusters decreases.

Figure 4.2 illustrates the application of the described algorithm on the example graph from figure 2.3 using A-LINK and a merge threshold of $t = 0.4$ There are two supersteps necessary until the algorithm terminates, shown in the first and second line respectively. Clusters are marked with dashed lines, center vertices are shown blue and non-centers white, messages are represented by thick arrows and red lines illustrate adjusted edges.

During the first phase of superstep 1, the vertex pairs (a_0, b_0) and (b_1, c_1) are detected as RNNs. Until this point all clusters only comprise one vertex which is why no additional cluster members have to be informed about the cluster merge during phase 2. The center

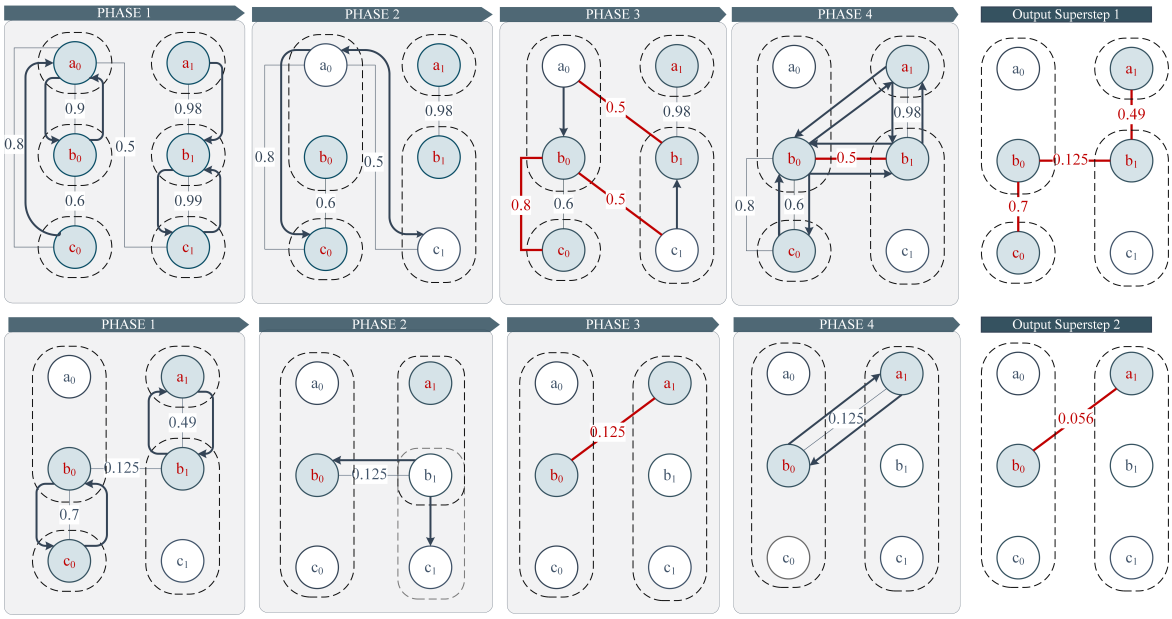


Figure 4.2: A-LINK Scatter-Gather HAC applied on the Example Similarity Graph with $t = 0.4$

status of vertices a_0 and c_1 is removed and they inform their neighbor vertices. Their edge targets are then updated accordingly, visible in phase 3. Now the old centers a_0 and c_1 send their adjusted neighbors to new cluster centers b_0 and b_1. To produce the firsts superstep output, all center vertices communicate their new cluster sizes among each other in phase 4. E.g. vertex b_0 receives a message from b_1 including the information that the cluster of b_1 consists of two vertices. Using its own cluster size, the new edge weight is computed according to the A-LINK strategy by the sum of all edge weights divided by the product of both cluster sizes , i.e. $\frac{0.5}{2*2} = 0.125$.

Employing the adjusted edge weights, RNNs are now redetermined in phase 1 of the second superstep. Whereas the cluster center of the first cluster is not changed, the second cluster has now a_1 as the new center. Since c_1 is also part of this cluster, b_1 needs to send a message to c_1 including the new cluster and center ID. As before, the old center vertex b_1 informs its neighbor b_0 about the center change, causing the connecting edge to adjust. No messages are necessary in phase 3 since all edges are already only connecting cluster centers. Again, the remaining center vertices communicate their cluster sizes among each other in phase 4, leading to the final clustering output. To ensure the correct computation of the new average edge weight, the sum and number of all connecting edges is internally stored, i.e. the new average is $\frac{0.5}{3*3}$. Even though b_0 and a_1 are RNNs, the algorithm terminates because their similarity is lower than the merge threshold of 0.4.

5 Evaluation

The scatter-gather algorithm will now be comparatively evaluated regarding its effectiveness and efficiency using four data sets from various domains applying different input specifications. All match quality results obtained using the scatter-gather implementation correspond to those from the naive implementation. The quality metrics used for this evaluation and the characteristics of the data sets will be described before analyzing the match quality and subsequently runtimes and scalability. All results will be compared to the seven previously implemented clustering schemes from Famer. Lastly, a comparative rating of the three HAC variants will be presented.

5.1 Quality Metrics

To evaluate the match quality of the clustering results, the standard metrics precision, recall and f-measure are used. They are determined by comparison of the computed match pairs (all entities from the same cluster are assumed as matches) with the perfect match pairs, that were either determined manually or were available because the data set was synthetically corrupted. For the MSCD data set an approximate *golden truth* had to be used.

	real match	**no match**	
match detected	true postive (tp)	false positive (fp)	detected positives (dp)
no match detected	false negative (fn)	true negative (tn)	detected negatives (dn)
	real positives (rp)	real negatives (rn)	

Table 5.1: Contingency Table, adopted from Powers [33, p. 2]

Determining the match quality can be traced back to a simple binary classification problem, assessing the condition if a detected match is actually a real match. The computation of the three metrics are based on a so-called contingency table, describing the relation between actual matches and the determined ones as shown in table 5.1.

Definition 5.1. Let tp, fp, dp and rp refer to the definitions from table 5.1. Then the following definitions hold:

$$\text{Precision } P = \frac{tp}{tp + fp} = \frac{tp}{dp} \tag{5.1.1}$$

$$\text{Recall } R = \frac{tp}{tp + fn} = \frac{tp}{rp} \tag{5.1.2}$$

$$\text{F-Measure } F = 2 * \frac{P * R}{P + R} \tag{5.1.3}$$

[33, p. 2–4]

While the precision defines the share of correctly clustered entities of all entities within a detected cluster, the recall defines the share of correctly clustered entities from all entities that should belong to a cluster. A good precision does therefore indicate that only truly matching entities are found, whereas a good recall indicates that all matching entities have been found. The f-measure combines both metrics in their harmonic mean. By increasing the match threshold θ, links within the similarity graph are eliminated and therefore the amount of false positives decreases, because lower similarity edges are more likely to be incorrectly detected matches. Hence, precision increases. At the same time, the amount of false negatives rises as the chance of removing correct matches increases with the match threshold, thus the recall falls.

5.2 Data Sets and Configurations Setup

This evaluation is based the same data sets that where used to study the seven clustering schemes that are already part of the Famer framework [23, 37, 38]. The four data sets each vary in size, their number of sources and degree of corruption. Their main characteristics and the applied input configurations are summarized in tables 5.2 and 5.3.

name	**domain**	**attributes**	**#entities**	**#sources**	**#clusters**	**#perfect match pairs**
DS1	geo-graphical	label, longitude, latitude	3,054	4	820	4,391
DS2	music	artist, title album, year, length	20,000	5	10,000	16,250
DS3-5M DS3-10M	persons	name, surname, suburb, postcode	5,000,000 10,000,000	5 10	3,500,840 6,625,848	3,331,384 14,995,973
DSC	camera models	hetereo-genous key-value pairs	21,023	23	3,910	368,546

Table 5.2: Data Set Characteristics

The three data sets DS1, DS2 and DS3 have been cleaned prior to clustering, i.e. the individual data sources do not contain any duplicates. The smallest data set DS1 is a real-world data set including geographical entities for which the perfect clusters were manually determined. The other two data sets have been synthetically modified using data generators. The second-largest data set DS2 is based on real data from the MusicBrainz database and is heavily corrupted containing duplicates for 50% of the original records in two to five of the five sources. DS3 is the largest data set and consists of five and ten sources respectively, each including one million entities from the North Carolina voter registry. 50% of its records are

duplicated without any corruption and present in all sources, the other 50% are corrupted using a moderate corruption rate of 20%, half of them are present in all sources whereas the remaining half appear in only some sources.

name	**blocking key**	**similarity functions**	**match rule**	**merge threshold**
DS1	PreLen1(label)	sim1: Jarowinkler(name) sim2: distance	sim1 $\geq \theta$ sim2 $\leq$ 1358km	θ
DS2	PreLen1(album)	sim: 3Gram(title)	sim $\geq \theta$	θ
DS3	PreLen3(surname)	sim: avg(3Gram(name), 3Gram(surname), 3Gram(suburb), 3Gram(postcode))	sim $\geq \theta$	θ
DSC	PreLen3(manu-facturers name) PreLen100(model number)	sim1: numerical similarity for numerical values sim2: 3Gram for string values	sim1 ≥ 0.3 avg(sim1, sim2) $\geq \theta$	θ

Table 5.3: Data Set Input Specifications

To further evaluate hierarchical clustering in the case of MSCD ER, a data set including mixed and dirty data sources is used, its additional specifications can be found in tables 5.4 and 5.5. The data set is based on test data from the ACM SIGMOD 2020 Programming Contest[2], containing approximately 30,000 product specifications from 24 dirty data sources. Each source is an e-commerce website and their names and numbers of entities are listed in table 5.4. All product specifications refer to camera models. For the purpose of this study, the original data set was filtered, the data source *www.alibaba.com* has been excluded and eight variants of the data set were generated, varying the share of clean data sources between 0 and 100 percent as outlined in table 5.5. As source number 7 (*www.ebay.com*) presents by far the largest data source, the choice of considering it as dirty or clean greatly influences the clustering result. DSC-26 and DSC-32 are used to evaluate this impact and therefore contain the data source in a clean and a dirty form respectively. Furthermore, two different data sets with a share of 62 percent clean sources have been generated. Both of them include source number 7 in a clean form, but all other clean sources from DSC-62A are left dirty in DSC-62B and vice versa. As the corruption concerning data set DSC is not generated synthetically and the perfect match pairs can not be determined manually, the match pairs of the SIGMOID contest winner [3] is considered the perfect match result. Applying extensive domain specific preprocessing and utilizing a list of nearly all available cameras on the market, the winners result achieved an f-measure of 0.99. On the contrary, Famer is a generic ER-tool that can be used for data sets regardless of any domain information.

Prior to clustering, the similarity graphs were generated by Famer's linking module using the specifications from table 5.3. Firstly, *Standard Blocking* was applied on all data sets. For the clean data sets DS1-DS3, the prefix of the specified property was used. In case of the MSCD

[2]http://www.inf.uniroma3.it/db/sigmod2020contest/index.html
[3]http://www.inf.uniroma3.it/db/sigmod2020contest/posters/SIGMODPoster.pdf

data set, a key combined of the manufacturers name and the model number was applied and only pairs with exactly the same model name, manufacturers part number (mpn) or european part number (ean) were kept. Secondly, similarities were calculated based on different string similarity functions on chosen properties, as well as in the case of DS1, the geographical distance and a numerical similarity for for numerical attributes in DSC.
The match threshold θ is varied from 0.75 to 0.9 for DS1 and from 0.6 to 0.9 for DS3. For the harder match problems DS2 and DSC θ had to be chosen lower to achieve acceptable results and is set between 0.35 an 0.45 in the case of DS2 and between 0.3 and 0.7 for DSC. All mentioned configurations have been determined during previously conducted studies [23, 37] by performing a large range of experiments using different input specifications. The match threshold t is chosen equal to θ for all three HAC algorithms. This decision will be justified in detail in the following section by evaluating a variety of merge and match thresholds configurations and their impact on the clustering result.

name	ID	# entities	# entities deduplicated
www.buy.net	1	358	244
www.cammarkt.com	2	198	94
www.buzzillions.com	3	832	630
www.cambuy.com.au	4	118	56
www.camerafarm.com.au	5	120	59
www.canon-europe.com	6	164	163
www.ebay.com	7	14,009	3,255
www.eglobalcentral.co.uk	8	190	75
www.flipkart.com	9	118	47
www.garricks.com.au	10	130	69
www.gosale.com	11	895	578
www.henrys.com	12	181	137
www.ilgs.net	13	102	64
www.mypriceindia.com	14	347	279
www.pcconnection.com	15	211	126
www.price-hunt.com	16	327	282
www.pricedekho.com	17	366	325
www.priceme.co.nz	18	740	475
www.shopbot.com.au	19	516	334
www.shopmania.in	20	630	556
www.ukdigitalcameras.co.uk	21	129	73
www.walmart.com	22	195	115
www.wexphotographic.com	23	147	87

Table 5.4: Camera Data Sources

name	% clean sources	clean source IDs	# clean entities	# dirty entities
DSC-0	0	-	0	21,023
DSC-26	26	1-6, 8-23	4,868	14,009
DSC-32	32	7	3,255	7,014
DSC-50	50	7, 18, 19, 20, 22, 23	4,882	4,786
DSC-62A	62	1, 4, 6, 7, 9, 11, 13, 15, 17, 17, 19, 20	5,748	3,536
DSC-62B	62	2, 3, 5, 7, 8, 10, 12, 14, 16, 18, 21-23	5,630	3,478
DSC-80	80	1-12, 14-18	6,894	3,478
DSC-100	100	1-23	8,123	0

Table 5.5: MSCD Data Sets

5.3 Match Quality of Clustering Approaches

The following match quality evaluation is split into three subsections. Before comparing the match quality results of the HAC algorithms among each other and to the ones already implemented in Famer, the impact of the merge threshold t will be evaluated by varying it on similarity graphs with different match threshold configurations. Afterwards the match quality will be analyzed using the three clean data sets and finally the behavior of HAC for dirty data sources will observed using the MSCD camera data set.

5.3.1 Choosing the Merge Threshold

This section will examine how the chosen merge threshold t affects the quality of the detected matching. The toy example from section 2.4 has already shown that the choice of t can significantly influence the clustering result. As the perfect match threshold will be unknown and furthermore differs for each data set and the applied input configuration, a good default strategy is required to facilitate the application of HAC.

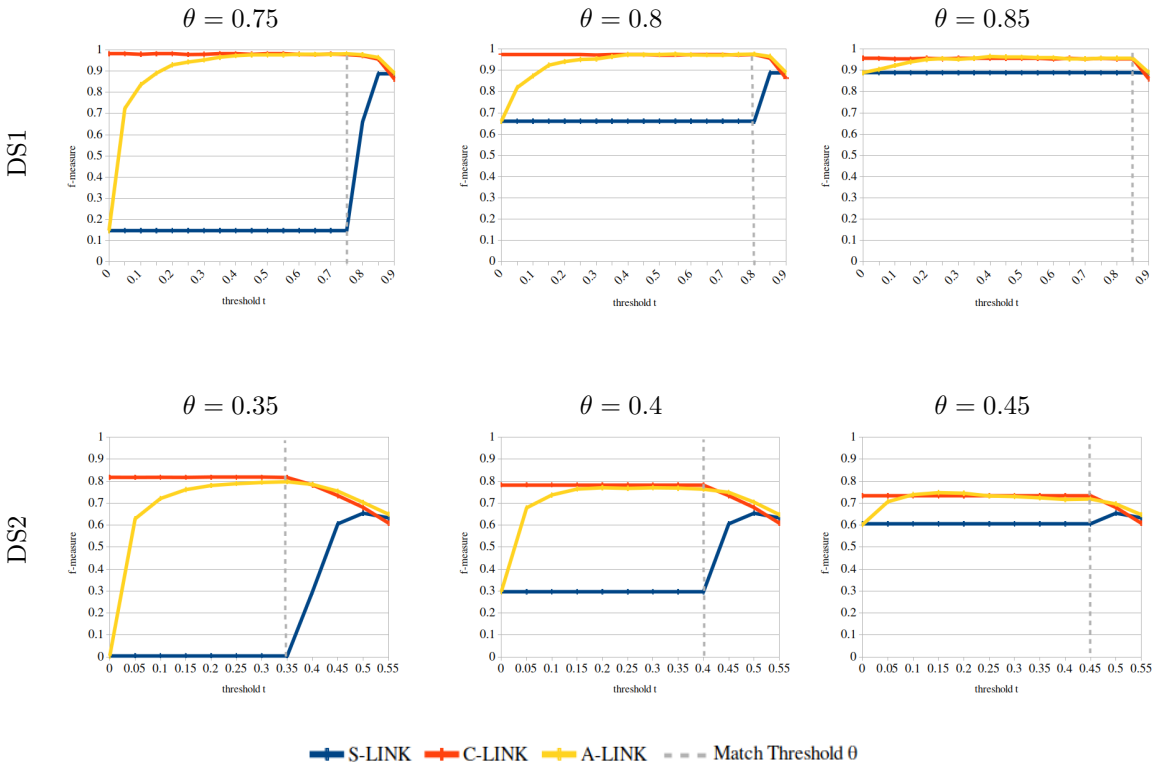


Figure 5.1: F-Measure dependent on Merge and Match Threshold on MSC data sets

An analysis of different merge and match threshold configurations is shown in figures 5.1 and 5.2 where the f-measure is plotted dependent on the merge threshold and a fixed match threshold. For S-LINK and C-LINK only merge thresholds $t \geq \theta$ produce different results whereas for all merge thresholds $t < \theta$ the f-measure stagnates on a single value, visible left

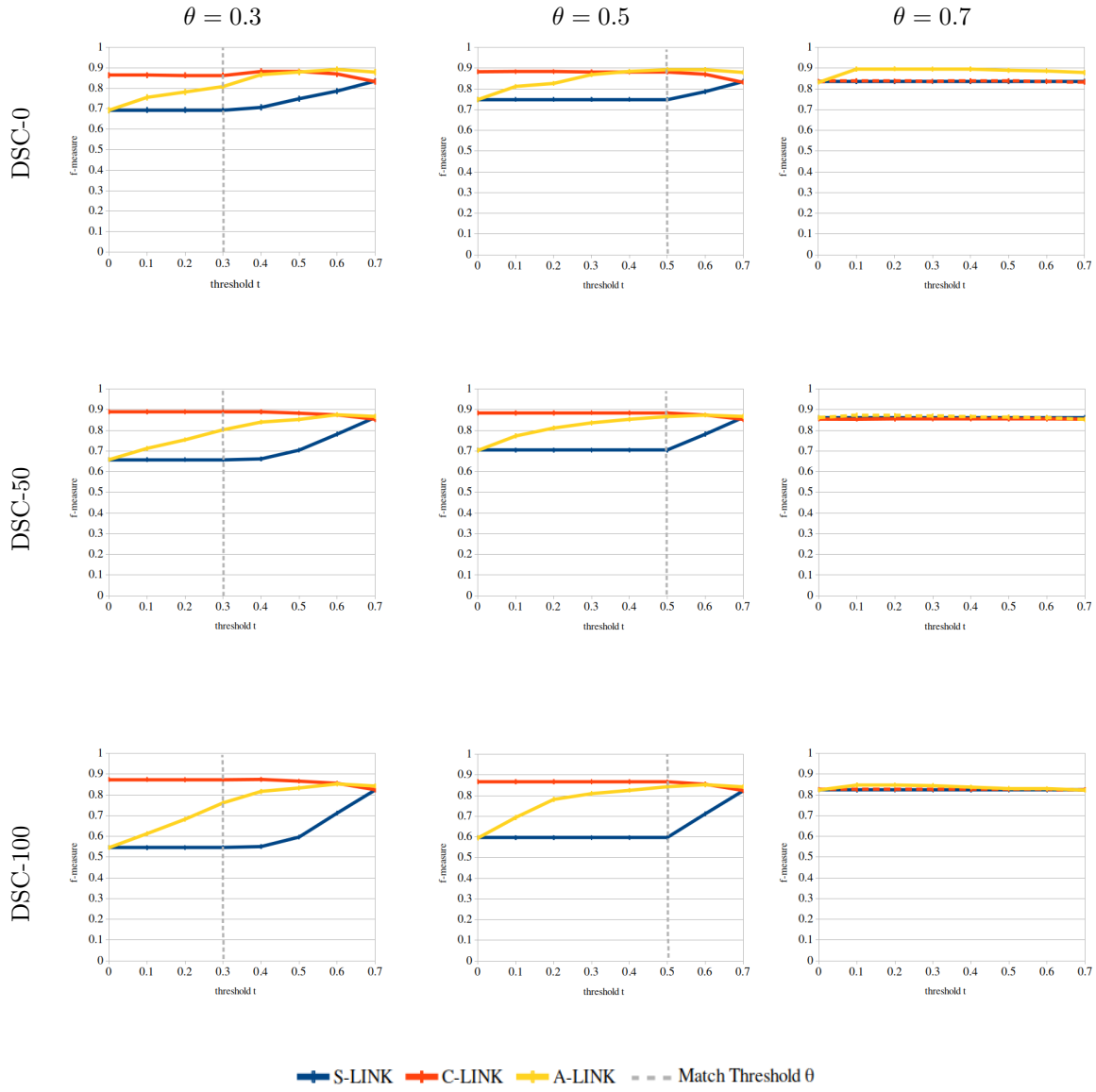


Figure 5.2: F-Measure dependent on Merge and Match Threshold on MSCD data sets

and right of the dashed grey line respectively. This observation can be explained easily, as for S-LINK and C-LINK there exist no links with similarity $< \theta$ and both methods do not alter but only remove certain edges during the recalculation of inter-cluster similarities. Choosing $t = 0.35$ or $t = 0.38$ for a data set generated with $\theta = 0.4$ will make no difference at all since no links with a weight < 0.4 are generated during the algorithms execution. For A-LINK however, new edges with a similarity less than θ are produced and cluster merges based on an inter-cluster similarity lower than θ are carried out. Although this leads to improved recall, the f-measure decreases with lower choices of t due to a significant loss in precision. Especially on the MSC data sets this impact is only visible for extremely low choices of t, the overall f-measure is steady for the majority of values. This leads to the conclusion that the newly calculated edge weights are in a range $\ll \theta$. The involvement of missing links and their consideration as edges with a weight of zero significantly pulls down the calculated average similarity. For $t = 0$ A-LINK and S-LINK results match since a merge threshold of

zero represents the root node of the cluster hierarchy where all vertices are grouped into one cluster. Due to the requirement of strict inequality between t and the calculated similarities and unconnected entities within the similarity graph this case corresponds to the graphs original connected components for A-LINK and S-LINK. It does not apply to C-LINK as the algorithm removes links from the similarity graph in a way that changes the graphs connected components. With increasing match threshold the cluster quality of the connected components increases and thereby the S-LINK and A-LINK minimum f-measure in range $t \leq \theta$.

Since A-LINK can not noticeably perform better for $t < \theta$ than for $t \geq \theta$, it can be concluded that only merge thresholds above the match threshold are worth considering. The further course of this section will therefore focus on analyzing the match quality for configurations with $t \geq \theta$.

While the results of the S-LINK algorithm in the range above θ vary highly and the f-measure increases significantly when increasing the merge threshold, A-LINK- and C-LINK are far more robust against the choice of t. Choosing the maximum similarity between the vertices from each cluster, as S-LINK does, results in poor match results especially for lower threshold configurations because too many vertices and hence wrong matches get incorporated into the formed clusters. Whereas S-LINK carries out merges on extremely high similarity values, A-LINK and C-LINK form clusters based on lower inter-cluster similarities, which leads to significantly better match results for lower choices of t, especially on the MSC data sets. As C-LINK's calculated cluster-linkages tend to be even lower than the ones computed using the A-LINK strategy, its performance suffers slightly for higher merge threshold configurations while A-LINK performs poorer for lower choices of t. Similar behavior can be observed on the other DSC data sets. Moreover is the range of results for the S-LINK algorithm considerably smaller on the MSCD data sets than on the MSC data sets which is further illustrated in figure 5.3. It shows the average f-measure achieved over all merge threshold configurations between the lowest and the highest match threshold including the minimum and maximum reached values for DS1-DS3 and moreover for the MSC data sets including portions of 0, 50, 80 and 100 percent clean sources. To observe the largest range of different clustering results the similarity graph generated using the lowest match threshold was used.

Since S-LINK favors larger clusters, its improved performance on DSC can be attributed to the high number of match pairs and clusters in within the data set. Yet, not only the occurrence of matches within the individual data sources but also the great number of sources in DSC improves the S-LINK results. This is visible for DSC-100 which only comprises clean sources but still leads to better match quality for S-LINK. The difference between A-LINK and C-LINK is also more pronounced on DSC where C-LINK can not only achieve slightly better average f-measure values but also leads to a smaller range of results than A-LINK. The maximum reachable f-measure however is roughly the same for both algorithms.

As already mentioned in section 2.4, the optimal merge threshold can not be obtained easily but considering the presented results, choosing the match threshold as the merge threshold seems a good default configuration. For C-LINK and A-LINK this enables almost optimal results except for the similarity graphs with the lowest threshold, e.g. for DSC-0 and $\theta = 0.3$, $t = 0.4$ would be the best choice for C-LINK and $t = 0.6$ for A-LINK. Note, that this decision

slightly favors the C-LINK variant due to its better results for lower threshold configurations. Especially on the MSCD data sets A-LINK could retrieve better results for higher choices of t on similarity graphs with a lower match threshold. C-LINK on the contrary achieves extremely steady results, hence the chance of retrieving high quality results using a fixed and randomly chosen threshold value is greater for C-LINK than for A-LINK. C-LINK's robustness against the choice of t can therefore justify the decision of setting $t = \theta$ for the evaluation and comparison in the following sections.

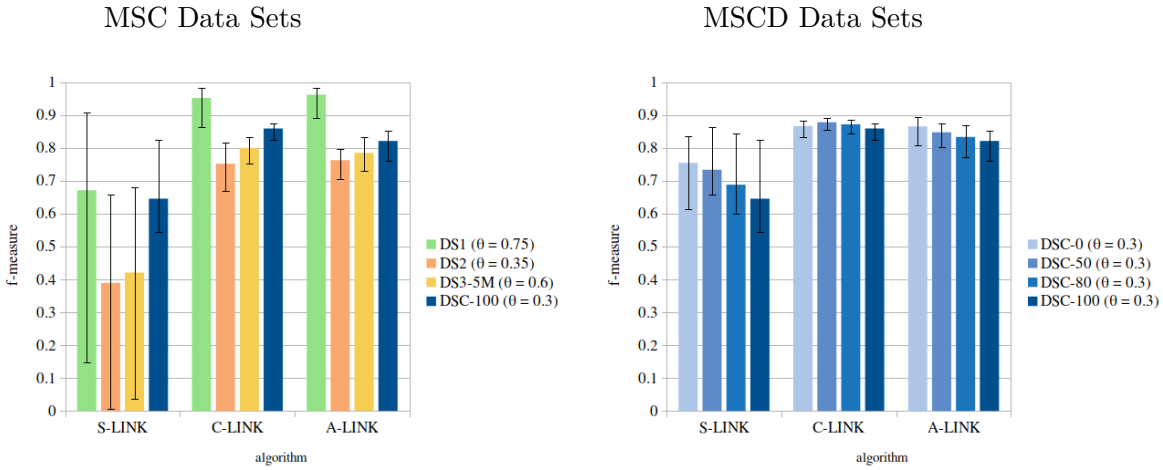


Figure 5.3: Average F-Measure Results over different Merge Thresholds including Min and Max Values

5.3.2 Comparison of Clustering Approaches for MSC Entity Resolution

Figure 5.4 shows the achieved results for the three metrics and different similarity thresholds for all ten clustering methods (S-LINK HAC, C-LINK HAC, A-LINK HAC, CLIP, Connected Components, CCPivot, Center, Merge-Center, Star-1 and Star-2) on the three clean data sources applying the configurations from table 5.3.
As already pointed out in chapter 3, the results of the S-LINK algorithm correspond to those from the Connected Components algorithm, which is analyzed in detail by Saeedi et. al. [37]. In summary, Connected Components reaches the overall lowest f-measure because it suffers from very poor precision, removing none of the matches from the similarity graph but only adding new ones by completing the transitive closure of the subgraph components. Although this enables great results in recall and lightly improves with increasing match threshold, its general performance is considered weak compared to other clustering schemes. This evaluation will therefore focus on analyzing the match quality of C-LINK and A-LINK HAC. Slight differences between S-LINK and Connected Components match quality results (as visible for DS2) are owed to the fact that Connected Components matches all entities with similarity $\geq \theta$ whereas S-LINK matches only those with similarity $> \theta$. Furthermore does Connected Components require the determination of one similarity graph per match threshold configuration to observe different results, while S-LINK is able to generate the same results on a single similarity graph by varying t.

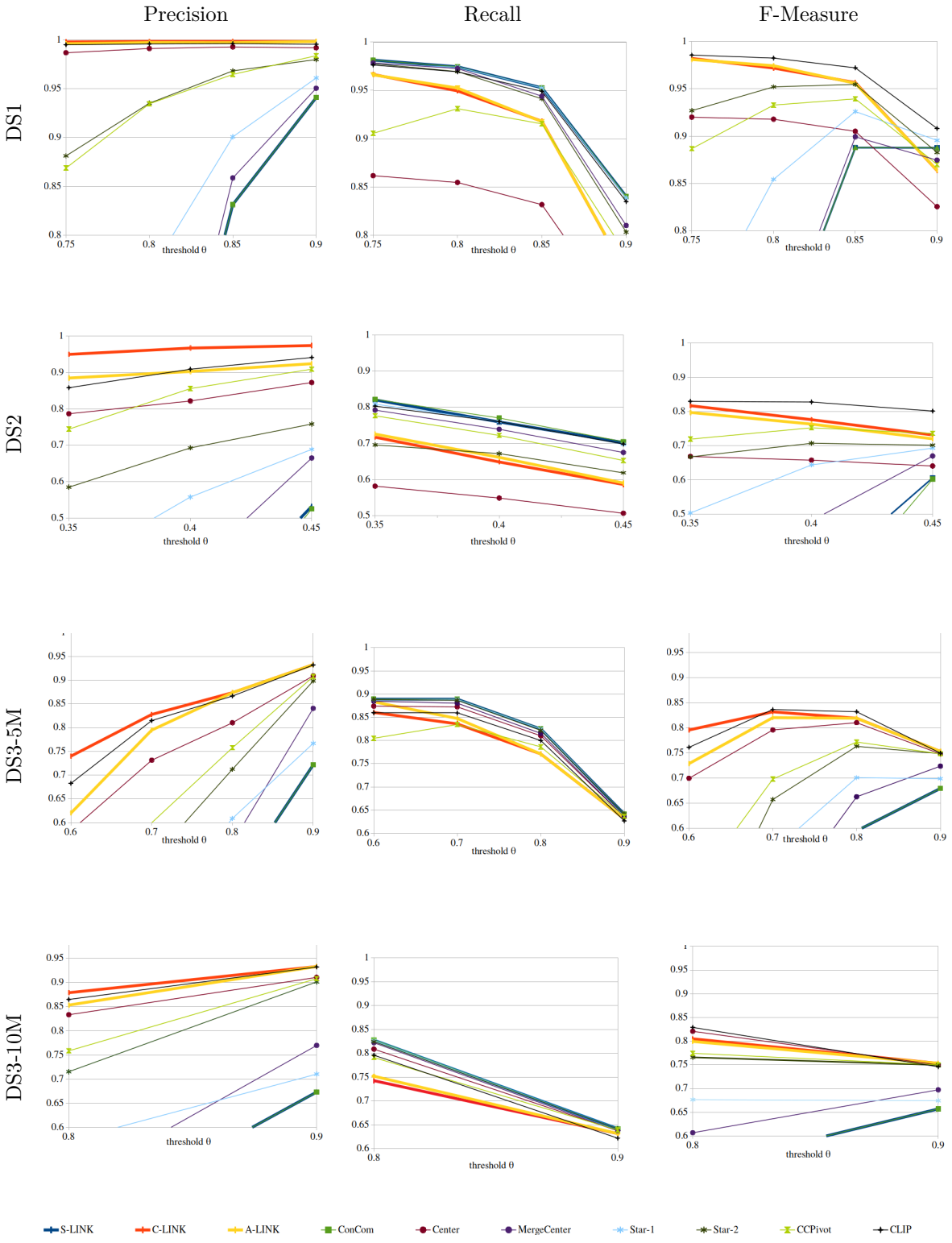


Figure 5.4: MSC Match Quality Results

It can be observed that both A-LINK and C-LINK HAC achieve good match quality results on all three data sets. Except for the highest match thresholds they outperform the six originally implemented clustering methods from Famer. For the lowest threshold values their f-measure almost reaches the excellent results attained by the new CLIP algorithm. C-LINK even gains the best result overall on DS3 with $\theta = 0.6$. Generally, C-LINK performs slightly

better than A-LINK, except for DS1 where both achieve roughly equal results. Generally, lower threshold configurations influence the HAC match quality results positively, as a larger number of links is still present in the input similarity graph, which means that less similarities have to be assumed to be 0 during the recalculation of inter-cluster similarities. This contrasts other clustering methods like Star-1 and Star-2, CCPivot or MergeCenter which require higher threshold configurations to achieve equally good results. They mainly determine larger clusters which leads to the inclusion of numerous wrong matches especially for lower match thresholds. Yet, they are able to detect additional links which can not be determined by other algorithms that mainly form smaller but more precise clusters, such as C-LINK and A-LINK HAC or Center and CLIP.
Furthermore is the achieved range of f-measure values over all threshold configurations for C-LINK and A-LINK HAC rather small compared to most other algorithms, the greatest difference considering the minimum and the maximum reached values does not exceed 0.1. Similar behavior can be observed for CLIP and Center, but especially the application of MergeCenter, Connected Components and Star-1 can lead to highly different clustering outputs regarding the match quality.
Since the recall for both HAC algorithms is among the worse of all approaches (only Center and CCPivot achieve worse results on some input configurations), the good f-measure values are primarily determined by extremely high precision values. Especially on the dirtier data set DS2, C-LINK significantly outperforms all other algorithms regarding precision. As CLIP and Center, that also achieve high precision, HAC focuses on edges with high similarities. In fact, the concept of link strength exploited in CLIP and the concept of RNNs are strongly related: Each link connecting two RNNs is also a strong link, because it is the link with the highest similarity for both vertices by the definition of RNNs. However, not every strong link is an RNN-link, as one entity might have more than one strong link (one with respect to each source), but only one nearest neighbor. As outlined in section 2.4, C-LINK HAC, like CLIP, only detects source-consistent clusters, further explaining the good results. But while the application of CLIP satisfies source-consistency by default, C-LINK HAC is only one special case of HAC and HAC can generally not guarantee source-consistent clusters. Because CLIP is based on determining the connected components preferring only strong and normal links it reaches significantly better recall values than A-LINK or C-LINK HAC. Yet, A-LINK HAC achieves a slightly better recall than C-LINK as inter-cluster similarities are only reduced but not set to zero if there are unconnected entities between two clusters, which leads to a higher probability for cluster merges. C-LINK is not even able to add any matches that have not been already determined during the similarity graph creation. While this seems to be a drawback at first and explains the poor recall, it is especially helpful for DS2 where the match threshold is chosen relatively low, meaning that many wrong links are still present in the similarity graph. On the contrary, it is interesting to notice that both C-LINK and A-LINK perform poorly on DS1 for the highest threshold of 0.9, where even S-LINK reaches better f-measure. This is can be reasoned similarly: DS1 presents the easiest match problem of all three data sets, where the similarity graph already contains many correct and few wrong matches. As both HAC versions tend to remove links rather than adding new ones, this leads to wrong cluster decisions by removing already correctly found matches, resulting in very

poor recall. The worse result on DS3-10M compared to DS3-5M is presumably owed to the fact that with the increasing number of sources the number of links between entities from different sources decreases which results in even smaller clusters and hence worse recall for A-LINK and C-LINK.
Overall, the positive results for HAC compared to all clustering schemes can be mainly attributed to two of the algorithms characteristics: Firstly to the focus on high similarity edges and secondly (and even more importantly) to the ability of including similarity values from every vertex within already identified clusters. The Center and Star-2 algorithms also achieve good match quality results by focusing on high edge weights, but they only consider one edge at at time (in case of Center) or all connecting edges between one single vertex and its neighbors (as for Star-2). By calculating the overall graphs degree, CCPivot is the only other algorithm that also regards multiple vertex connections at a time. This leads to accordingly high quality clusters, yet does the detection of clusters based on successive merges outperform the CCPivot method.

5.3.3 Comparison of Clustering Approaches for MSCD Entity Resolution

Figure 5.5 depicts the match quality results achieved by all ten clustering methods using the data sets DSC-0, DSC-50, DSC-80 and DSC-100. The top row therefore shows the results when all data sources contain duplicates, the bottom row represents an MSC ER problem and for both middle rows the data sets comprise a mixture of clean and dirty sources, including 50% and 80% clean sources respectively. The special cases focusing on the impact of the large data source 7 using DSC-26, DSC-32 and DSC-62A/B will be covered separately.
Mainly, the application of HAC on MSCD data sets shows similar behavior as the application on clean data sources. Summarizing, this involves extremely high precision values for C-LINK HAC on all thresholds and data sets, albeit at the cost of very poor recall results, where only the Center algorithm performs worse. A-LINK HAC can not reach the same excellence in precision as C-LINK but results in better recall. The overall f-measure benefits from lower threshold configurations especially for C-LINK, which outperforms the other algorithms greatly on the lowest threshold of 0.3 but only slightly on the middle threshold of 0.5 and not at all on the highest threshold of 0.7. An exception poses the CLIP algorithm on DSC-100 where it significantly surpasses all other algorithms since it is specialized in clean multi-source ER. Its assumption of clean individual data sources does consequently lead to very poor results on all other DSC data sets as it wrongly removes all links within the single dirty data sources.
With increasing share of dirty data sources, the difference in match quality between HAC and the other algorithms rises. This is less attributable to the increasing performance of HAC on clean data sources but to the decreasing performance of the other algorithms. Whereas especially C-LINK gains overall steady results, both Star algorithms, CCPivot, Merge Center and Connected Components/S-LINK perform noticeably better on DSC-0 than DSC-100. Only the Center algorithm shows similar steady results as C-LINK, but only because its drop in precision with increasing share of dirty data sources is balanced by the simultaneous rise in recall. Vice versa results can be observed for C-LINK: Its precision rises with rising

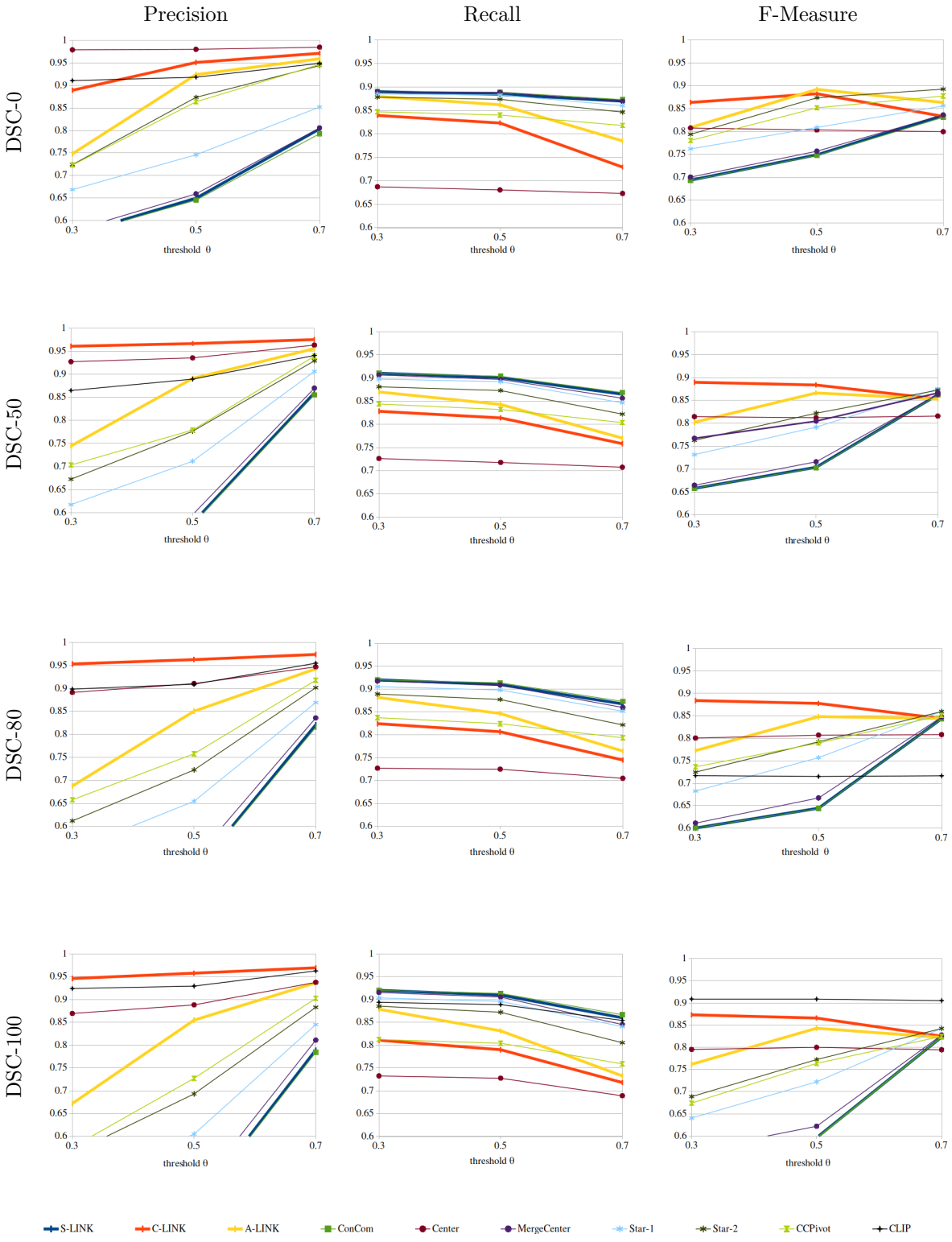


Figure 5.5: MSCD Match Quality Results

portions of clean sources, at the same time the recall drops, but the overall change is not as pronounced as for the Center algorithm. Whereas C-LINK decides about removing links by considering all entities within already detected clusters, the Center algorithm processes only one link, i.e. two vertices at a time. The decision about removing the link is made based on the cluster membership and the similarity of the currently processed edge regardless of

the similarity to other cluster members. This can lead to many wrong cuts when there are additional links present within the individual data sources. The consideration of all edges connecting two clusters results in better recall, although precision suffers lightly.
The main difference regarding the behavior of the HAC algorithms on MSC data sets compared to MSCD data sets is the considerably greater difference in match quality between C-LINK and A-LINK HAC. While their f-measure and recall on DS1-DS3 were extremely close by, the results on DSC are noticeably distinguishable, especially for lower match thresholds. If not too many links are already removed during the similarity graph creation, C-LINK mainly determines the same clusters regardless of the input specifications. This can again be attributed to the high probability of removing inter-cluster links. Exclusively fully connected clusters are detected which change only marginally over different configurations. If a cluster consists of n vertices and is about to get merged with only one single other vertex, then n links between the cluster and the vertex are required to perform the merge (for a cluster merge with another cluster of size m, $n \times m$ links would be needed). Any number of links lower than n would force the clusters to stay separated; no matter if there are 0 or $n-1$ links, the cluster-linkage would be set to a value of 0 in any case. When applying A-LINK however, $n-1$ links would lead to a significantly higher probability of merging because the calculated inter-cluster similarity adapts to a change in the number of links in any range.
Interestingly, A-LINK HAC is able to outperform C-LINK HAC on the purely dirty data set DSC-0 for thresholds of 0.5 and 0.7 and its performance drops the cleaner the data set gets. While it reaches an f-measure of 0.89 for $\theta = 0.5$, (which is the overall best result considering all configurations on all dirty and mixed DSC data sets), it can only achieve a maximum of 0.84 on DSC-100. Star-2 and CCPivot produce similarly good results on DSC-0 but only for the highest match threshold. With a maximum f-measure of 0.87 for Star-2 and 0.85 for CCPivot they still slightly fall behind A-LINK regarding their maximum values on DSC-0. A-LINK benefits from the greater amount of entities and links in DSC-0, which enables a more accurate calculation of inter-cluster similarities. Each edge that has to be assumed to be 0 skews the similarity result because it is considered with the same weight as non-zero edges during the computation of the new average similarity. A larger number of duplicates and thus entities within an already identified cluster helps calculating a precise similarity result that tends less towards zero. Therefore, the chance for cluster merges increases leading to a significant boost in recall. For the lower thresholds on DCS-0 A-LINK almost reaches the same recall as the top-recall algorithms MergeCenter, Connected Compontents/S-LINK and Star-1.

It will now further be analyzed, how cleaning the largest data source 7 affects the clustering results. As presumable there is a visible difference in match quality although at a different degree for each clustering scheme. Figure 5.6 shows the achieved precision, recall and f-measure regarding the data sets DSC-26 and DSC-32, the former still includes duplicates in data source 7 whereas in the latter case the source has been cleaned. The DSC-26 results highly resemble the ones from DSC-0 where data source 7 supplies most of the dirty entities. By removing these duplicates the algorithms Connected Components/S-LINK, MergeCenter and Star-1 are able to improve their precision slightly because they are supported in their poor ability to detect incorrect matches. Algorithms that are already able to determine high

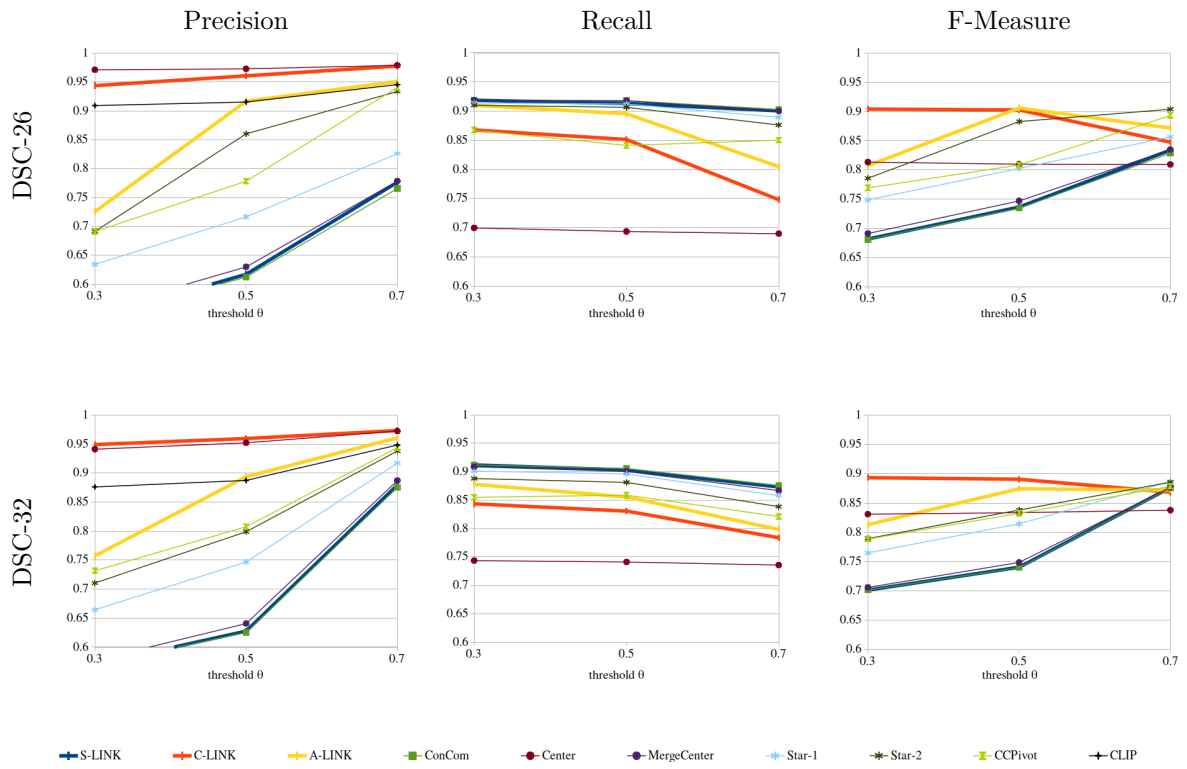


Figure 5.6: MSCD Match Quality Results on DSC-26 and DSC-32

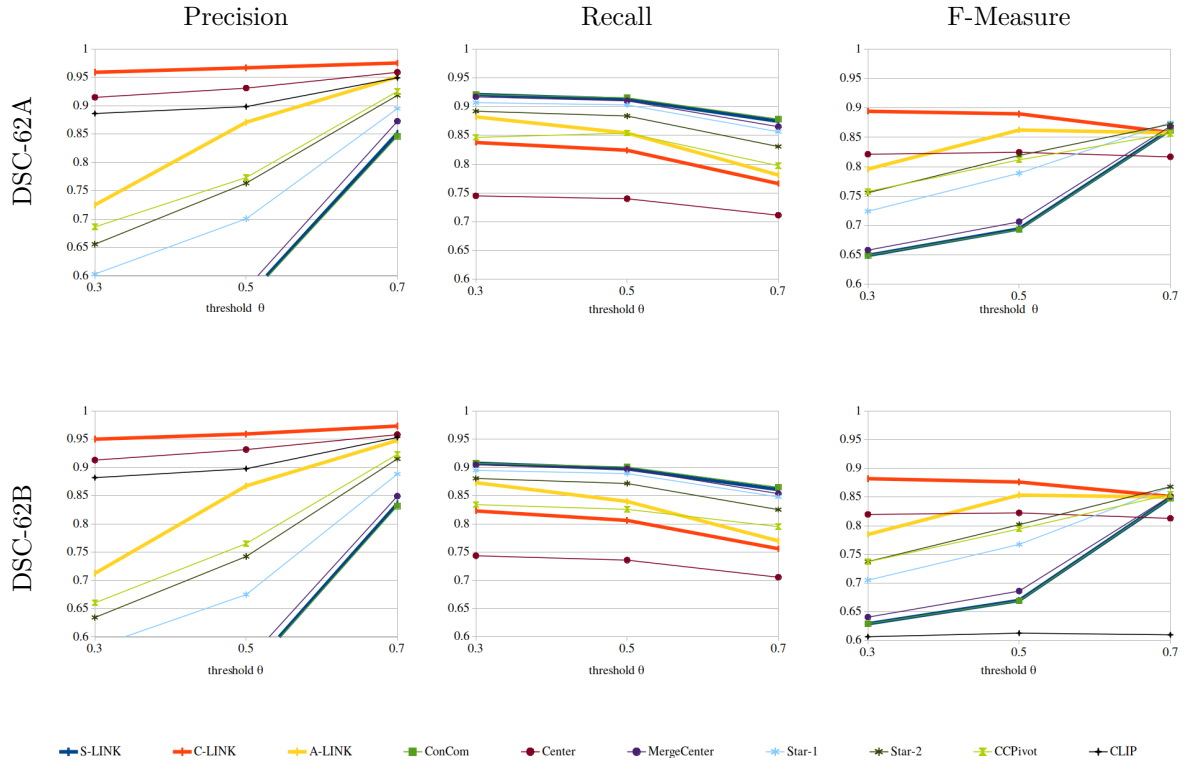


Figure 5.7: MSCD Match Quality Results on DSC-62A and DSC-62B

precision clusters do not benefit from cleaning the source, their results even suffer a little. Besides CLIP, Center and Star-2 this also includes C-LINK and A-LINK HAC. The latter two

further experience a noticeably drop in recall, indicating that the larger amount of duplicates from source 7 supports the detection of larger clusters preventing them from cutting correct match pairs. A-LINK benefits considerably more than C-LINK, underlining the robustness of the C-LINK algorithm not only against the threshold configurations but also regarding the share and choice of dirty data sources.
While the impact of the large data source 7 visibly affected the clustering quality, the choice which of the smaller data sources is considered clean or dirty does not significantly influence the match quality. Figure 5.7 shows the three quality metrics for data set variants DSC-62A and DSC-62B. Both data sets include source 7 in a clean form but all dirty data sources from DSC-62A are clean in DSC-62B and vice versa. It becomes clear that there is almost no considerable difference between the results on the two data sets. The overall precision is improved minimal on DSC-62A while CLIP can achieve a marginal improvement on DSC-62B. It can therefore be reasoned that all data sources except from source 7 are well suited to generate representative data set variants as a base for an informative and meaningful evaluation.

5.4 Runtimes and Speedup

Besides analyzing match quality results, scalability and runtimes have to be considered in order to evaluate how the newly implemented HAC variants perform in a large-scale environment and hence to decide if they are a useful addition to the Famer framework. The applied system configurations will be described before conducting a speedup analysis and comparing the absolute HAC runtimes to the ones measured for the other seven clustering schemes.

5.4.1 System Configurations

All runtimes were determined on a cluster with 16 worker nodes connected via 1 Gigabit Ethernet, each consisting of an E5-2430 6(12) 2.5GHz CPU, 48GB RAM, two 4TB SATA disks and running openSUSE 13.2. The evaluation is based on Hadoop 2.6.0 and Apache Flink 1.9.0, running as a standalone cluster with six threads and 40 GB memory per worker. Due to the recurring appearance of time-out-exceptions, the runtimes for the largest data set comprising ten sources could only be executed for similarity thresholds ≥ 0.8 and all 16 workers. The speedup analysis is hence restricted to the smaller data set consisting of five sources each containing one million records. None of the HAC algorithms could be executed for one worker only due to the occurrence of out-of-memory-errors. Absolute runtimes shown in tables 5.6 and 5.7 represent the average runtime accumulated from multiple independent runs with a merge and match threshold of 0.8.

5.4.2 Speedup Analysis

Figure 5.8 illustrates the speedup for DS3-5M varying the number of workers from 2 to 16 and table 5.6 shows the corresponding runtimes.

	no. of workers			
algorithm	2	4	8	16
S-LINK	2541	1626	1548	1423
C-LINK	614	316	154	108
A-LINK	604	309	153	107

Table 5.6: Runtimes (seconds) for HAC on DS3-5M

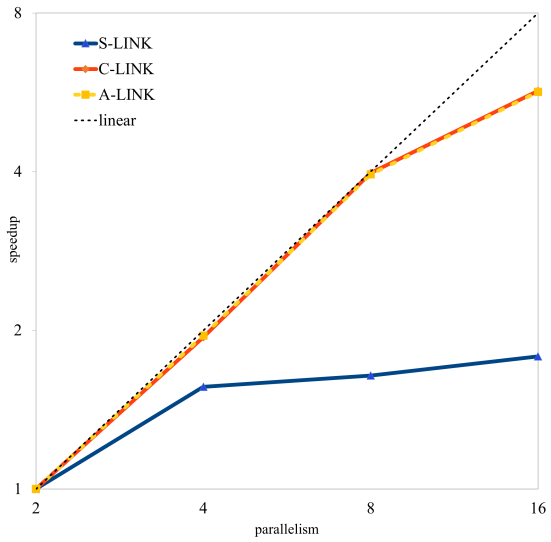


Figure 5.8: Speedup for HAC on DS3-5M

Both A-LINK and C-LINK HAC achieve linear speed-up except for 16 workers, indicating that they are able to effectively exploit the scatter-gather paradigm. The sub-linear speedup for the largest number of workers suggests that the data set DS3-5M might not have been large enough. On the contrary, S-LINK runtimes only slightly improve with growing number of workers, which results in overall poor speedup behavior. Furthermore a great difference in runtimes from the independent runs of the S-LINK algorithm could be observed. Depending on which vertex was chosen as center, the number of messages to send can vary highly: If a cluster only consists of one vertex and it gets chosen cluster center during a merge with a much larger cluster, then the latter has to broadcast the new cluster ID to all its cluster members, whereas the other way round only the single vertex would need to update its cluster ID. A possible way of improving the S-LINK runtime would therefore consist of preferring the cluster center of smaller clusters instead of choosing it dependent on the randomly assigned vertex priorities. This does not apply to C-LINK or A-LINK HAC as they both result in much smaller and more compact clusters and all runtimes from the independent runs were roughly equal. The overall slow runtime and poor speedup for the S-LINK algorithm can also mainly be explained by the fact that S-LINK results in larger and less clusters, hence requiring more merges than A-LINK or C-LINK HAC. As one cluster merge consists of four scatter-gather phases, the number of messages to send and process increases respectively. Furthermore, while the algorithm proceeds and clusters are growing, the number of active vertices decreases since only one vertex per cluster is in charge of computing and maintaining

the inter-cluster similarities. This restricts the degree of parallelization regarding the S-LINK algorithm as visible in the speedup curve.

5.4.3 Comparison of Runtimes

While S-LINK HAC has not only the worst runtime of all HAC algorithms but also the worst runtime over all other clustering schemes, C-LINK and A-LINK are among the faster algorithms. Only the simple Connected Components approach and CLIP achieve better results on DS3-5M. Considering the runtimes on the largest data set DS3-10M, where the input size is doubled compared to DS3-5M, all HAC runtimes are increased by a factor around 3. The runtime of other algorithms like Star-1 and Star-2 increase by a lower degree so that they achieve better runtime results on DS3-10M than the HAC algorithms.

algorithm	DS3 - 5M	DS3 - 10M
S-LINK	1098	3425
C-LINK	108	315
A-LINK	107	334
Connnected Components	37	57
Center	156	603
Merge Center	210	730
Star-1	124	273
Star-2	124	233
CCPivot	426	964
CLIP	81	195

Table 5.7: Runtimes (seconds) for all Clustering Schemes on DS3 using 16 Workers

The poorer results concerning the runtimes on the largest data is a result of the relatively high memory requirements of HAC. Each vertex has to maintain several additional vertex properties, especially the new edges property has to be maintained and updated during all four scatter-gather phases, which is not necessary for any of the other algorithms as they do not require changing the edge values and can therefore send all messages along their in- and outgoing edges. The overall poor performance of the S-LINK algorithm is negligible since the same clustering result can be retrieved with faster algorithms like Connected Components or the naive S-LINK implementation of HAC.

Although not included visually here, other parameters that significantly influence runtimes are the match and merge thresholds. Choosing a higher match threshold configuration usually leads to faster runtimes since the number of links within the input graph and therefore the amount of data to process decreases. A higher merge threshold for HAC results in similar observations. The lower the choice of t, the more cluster merges will be performed, increasing the computation workload and therefore the runtime. As already pointed out in the preceding section, a greater number of cluster merges (hence a smaller number of output clusters) results in less active vertices loaded with a greater computation effort, which also causes increased runtime.

5.5 Evaluation Summary

To summarize the evaluation results obtained during the previous sections, table 5.8 briefly recaps the advantages and drawbacks of the three hierarchical clustering algorithms and rates them on a scale of ++ to -- based on the vertex-centric implementation.

	S-LINK	**C-LINK**	**A-LINK**
runtime	-	+	+
scalability	-	+	+
MSC precision	--	++	+
MSC recall	++	--	-
MSCD precision	--	++	+
MSCD recall	++	--	o
t robustness	--	++	+
θ robustness	--	+	o

Table 5.8: Comparative Rating of HAC Evaluation Results

The performance of S-LINK is overall unsatisfactory, not only poor match quality results and the great dependence on the chosen parameter t but further its long runtimes and weak speedup behavior lead to the fact that it is no useful extension within the Famer framework. The S-LINK quality results can be achieved using the much faster Connected Components algorithm and the scatter-gather model is apparently no appropriate fit, as the naive S-LINK version performed significantly better than the vertex-centric implementation considering runtimes.

A-LINK and C-LINK HAC however, can usefully exploit the scatter-gather model enabling quick runtimes and scalability, while also reaching good match quality results. By making cluster decisions dependent on all links connecting two clusters rather than only considering one edge at a time, hierarchical clustering is even able to exceed the cluster quality of many previously implemented clustering techniques from Famer. C-LINK can be used as a good default strategy when it comes to multi-source clean ER, especially when lower threshold configurations are necessary due to more heavily corrupted data sets. A-LINK on the other hand is useful on completely dirty multi-source data sets. Yet, its results depend on the chosen match and merge threshold to a considerable extent, which is less an issue for C-LINK making C-LINK the overall most convincing HAC strategy.

6 Conclusion and Outlook

Within this study three different types of hierarchical agglomerative clustering have been implemented and integrated into the large-scale ER tool Famer. In contrast to related distributed hierarchical clustering approaches that are mainly based on partitioning, the implementation developed during this study is able to utilize vertex-centric programming by exploiting the concept of Reciprocal Nearest Neighbors. The evaluation using data sets of different sizes and degree of corruption has shown that the A-LINK and C-LINK variants of HAC achieve good match quality results not only on clean multi-source ER data sets but also for a variety of data sets including dirty or a mixture of clean and dirty data sources. Except from S-LINK, the parallel versions of HAC further attained good speedup values and therefore support scalability for large data sets making them a useful addition to the Famer framework. However, due to high memory requirements and complex vertex updates all HAC algorithms encountered difficulties when applied on the largest data set using lower threshold values and could not be executed for some input configurations. Its ability to include similarity values from every vertex within already identified clusters clearly contributes to the great match results, yet complicates the implementation which leads to the mentioned issues. This offers the opportunity to further improve the algorithms performance within future work.
Besides making the implementation more efficient, following studies could include the implementation and evaluation of further linkage strategies or a modification of the existing three by overcoming the drawback of assuming edge weights of 0. For A-LINK these edges could be kept but assigned a lower weight within the computation of the average similarity and for C-LINK setting the inter-cluster similarity to a low (but not zero) default value in case of unconnected vertices between the clusters could be another possibility.
As the implementation proposed in this study requires the merge threshold as an additional input parameter and the decision of when to stop cluster merging affects the clustering quality noticeably, further research regarding the automation of this decision would be particularly interesting. This could include determining a (nearly) optimal threshold value or implementing additional stopping conditions and criteria that could not be studied within this thesis but have been applied to other problem domains.
During the implementation of the naive approach of HAC a significant quality improvement could be observed caused by the removal of weak links prior to clustering. The analysis of this behavior could also serve as matter of future work. Lastly, the cluster repairing approach RLIP, which has already been studied on the six other clustering schemes implemented in Famer, could be applied on S-LINK and A-LINK HAC in the case of clean MSC ER. Since C-LINK is already able to produce source-consistent clusters and HAC generally avoids overlapping clusters, RLIP is not expected to further improve C-LINK HAC.

Bibliography

[1] Emanuele Pio Barracchia, Gianvito Pio, Domenica D'Elia, and Michelangelo Ceci. Prediction of new associations between ncRNAs and diseases exploiting multi-type hierarchical clustering. *BMC bioinformatics*, 21(1):1–24, 2020.

[2] Jon Louis Bentley. A parallel algorithm for constructing minimum spanning trees. *Journal of Algorithms*, 1(1):51 – 59, 1980.

[3] Christoph Böhm, Gerard De Melo, Felix Naumann, and Gerhard Weikum. LINDA: distributed web-of-data-scale entity matching. In *Proceedings of the 21st ACM international conference on Information and knowledge management*, pages 2104–2108, 2012.

[4] Paris Carbone, Asterios Katsifodimos, Sics Sweden, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. Apache Flink: Stream and Batch Processing in a Single Engine. *IEEE Data Engineering Bulletin*, 38, 01 2015.

[5] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.*, 53(6), December 2020.

[6] Elias Dahlhaus. Parallel Algorithms for Hierarchical Clustering and Applications to Split Decomposition and Parity Graph Recognition. *Journal of Algorithms*, 36(2):205 – 240, 2000.

[7] Kristine Daniels and Christophe Giraud-Carrier. Learning the Threshold in Hierarchical Agglomerative Clustering. In *2006 5th International Conference on Machine Learning and Applications (ICMLA'06)*, pages 270 – 278, 2006.

[8] Manoranjan Dash, Simona Petrutiu, and Peter Scheuermann. pPOP: Fast yet accurate parallel hierarchical clustering using partitioning. *Data and Knowledge Engineering*, 61(3):563 – 578, 2007. Advances on Natural Language Processing.

[9] F. Dehne and S. Götz. Practical Parallel Algorithms for Minimum Spanning Trees. In *Proceedings of the The 17th IEEE Symposium on Reliable Distributed Systems*, SRDS '98, page 366, USA, 1998. IEEE Computer Society.

[10] Glenn Fung. A Comprehensive Overview of Basic Clustering Algorithms. *Citeseer*, 01 2001.

[11] Paulene Govender and Venkataraman Sivakumar. Application of k-means and hierarchical clustering techniques for analysis of air pollution: A review (1980-2019). *Atmospheric Pollution Research*, 11(1):40 – 56, 2020.

[12] Oktie Hassanzadeh, Fei Chiang, Renée Miller, and Hyun Lee. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *PVLDB*, 2:1282–1293, 08 2009.

[13] William Hendrix, Md. Mostafa Ali Patwary, Ankit Agrawal, Wei keng Liao, and Alok Choudhary. Parallel Hierarchical Clustering on Shared Memory Platforms. In *2012 19th International Conference on High Performance Computing*, pages 1–9, 2012.

[14] Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok Choudhary. A Scalable Hierarchical Clustering Algorithm Using Spark. In *2015 IEEE First International Conference on Big Data Computing Service and Applications*, pages 418–426, 2015.

[15] Martin Junghanns, André Petermann, Niklas Teichmann, Kevin Gómez, and Erhard Rahm. Analyzing extended property graphs with Apache Flink. In *Proceedings of the 1st ACM SIGMOD Workshop on Network Data Analytics*, pages 1–8, 2016.

[16] Vasiliki Kalavri. *Performance Optimization Techniques and Tools for Distributed Graph Processing*. PhD thesis, KTH Royal Institute of Technology, 2016.

[17] Leonard Kaufmann and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, 2005.

[18] Hisashi Koga, Tetsuo Ishibashi, and Toshinori Watanabe. Fast agglomerative hierarchical clustering algorithm using Locality-Sensitive Hashing. *Knowledge and Information Systems*, 12(1):25–53, 05 2007.

[19] Lars Kolb, Andreas Thor, and Erhard Rahm. Load Balancing for MapReduce-based Entity Resolution. In *2012 IEEE 28th International Conference on Data Engineering*, pages 618–629, 2012.

[20] Hanna Köpcke and Erhard Rahm. Frameworks for entity matching: A comparison. *Data & Knowledge Engineering*, 69:197–210, 02 2010.

[21] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[22] Silvio Lattanzi, Thomas Lavastida, Kefu Lu, and Benjamin Moseley. A Framework for Parallelizing Hierarchical Clustering Methods. In Ulf Brefeld, Elisa Fromont, Andreas Hotho, Arno Knobbe, Marloes Maathuis, and Céline Robardet, editors, *Machine Learning and Knowledge Discovery in Databases*, pages 73–89, Cham, 2020. Springer International Publishing.

[23] Stefan Lerm, Alieh Saeedi, and Erhard Rahm. Extended Affinity Propagation Clustering for Multi-source Entity Resolution. 2021.

[24] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of Massive Datasets*. Cambridge University Press, USA, 2nd edition, 2014.

[25] Grzegorz Malewicz, Matthew H. Austern, Aart J.C Bik, James C. Dehnert, Ilan Horn, Naty Leiser, and Grzegorz Czajkowski. Pregel: A System for Large-Scale Graph Processing. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 135 – 146, New York, NY, USA, 2010. Association for Computing Machinery.

[26] Qi Mao, Wei Zheng, Li Wang, Yunpeng Cai, Volker Mai, and Yijun Sun. Parallel Hierarchical Clustering in Linearithmic Time for Large-Scale Sequence Analysis. In *2015 IEEE International Conference on Data Mining*, pages 310–319, 2015.

[27] Robert Ryan McCune, Tim Weninger, and Greg Madey. Thinking Like a Vertex: A Survey of Vertex-Centric Frameworks for Large-Scale Distributed Graph Processing. *ACM Comput. Surv.*, 48(2), 10 2015.

[28] Boris Mirkin. Choosing the number of clusters. *WIREs Data Mining and Knowledge Discovery*, 1(3):252–260, 2011.

[29] Fionn Murtagh. A Survey of Recent Advances in Hierarchical Clustering Algorithms. *The Computer Journal*, 26(4):354–359, 11 1983.

[30] Fionn Murtagh and Pedro Contreras. Algorithms for hierarchical clustering: an overview. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(1):86–97, 2012.

[31] Clark F. Olson. Parallel algorithms for hierarchical clustering. *Parallel Computing*, 21(8):1313 – 1325, 1995.

[32] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.*, 53(2), 03 2020.

[33] David Powers. Evaluation: From Precision, Recall and F-factor to Roc, Informedness, Markedness & Correlation. *Mach. Learn. Technol.*, 2, 01 2008.

[34] Erhard Rahm. The Case for Holistic Data Integration. In *East European Conference on Advances in Databases and Information Systems*, pages 11–27. Springer, 2016.

[35] David Reinsel, John Gantz, and John Rydning. The Digitization of the World - From Edge to Core. Technical report, International Data Corporation (IDC), 11 2018.

[36] Alieh Saeedi, Markus Nentwig, Eric Peukert, and Erhard Rahm. Scalable matching and clustering of entities with FAMER. *Complex Systems Informatics and Modeling Quarterly*, (16):61–83, 2018.

[37] Alieh Saeedi, Eric Peukert, and Erhard Rahm. Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution. In Mārīte Kirikova, Kjetil Nørvåg, and George A. Papadopoulos, editors, *Advances in Databases and Information Systems*, pages 278–293, Cham, 2017. Springer International Publishing.

[38] Alieh Saeedi, Eric Peukert, and Erhard Rahm. Using Link Features for Entity Clustering in Knowledge Graphs. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web*, pages 576–592, Cham, 2018. Springer International Publishing.

[39] R. Sibson. SLINK: An optimally efficient algorithm for the single-link cluster method. *The Computer Journal*, 16(1):30–34, 01 1973.

[40] John R Talburt. *Entity Resolution and Information Quality*. Elsevier, 2011.

[41] Hongzhi Wang. *Innovative Techniques and Applications of Entity Resolution*. IGI Global, 2014.

[42] Vasilios Zarikas, Stavros G Poulopoulos, Zoe Gareiou, and Efthimios Zervas. Clustering analysis of countries using the COVID-19 cases dataset. *Data in brief*, 31:105787, 2020.

[43] Wei Zhang, Gongxuan Zhang, Xiaohui Chen, Yueqi Liu, Xiumin Zhou, and Junlong Zhou. DHC: A Distributed Hierarchical Clustering Algorithm for Large Datasets. *Journal of Circuits, Systems and Computers*, 28, 06 2018.

[44] Ying Zhao, George Karypis, and Usama Fayyad. Hierarchical clustering algorithms for document datasets. *Data mining and knowledge discovery*, 10(2):141–168, 2005.

List of Abbreviations

API	Application Programming Interface
EPGM	Extended Property Graph Model
ER	Entity Resolution
FAMER	Fast Multi-source Entity Resolution
HAC	Hierarchical Agglomerative Clustering
MSC	Multi-source Clean
MSCD	Multi-source Clean Dirty
A-LINK	average-linkage
C-LINK	complete-linkage
S-LINK	single-linkage
CLIP	Clustering based on Link Priority
RLIP	Cluster Repair based on Link Priority
LSH	Locality-Sensitive Hashing
MST	Minimum Spanning Tree
NN	Nearest Neighbor
RNN	Reciprocal Nearest Neighbor
TLAV	Think like a vertex

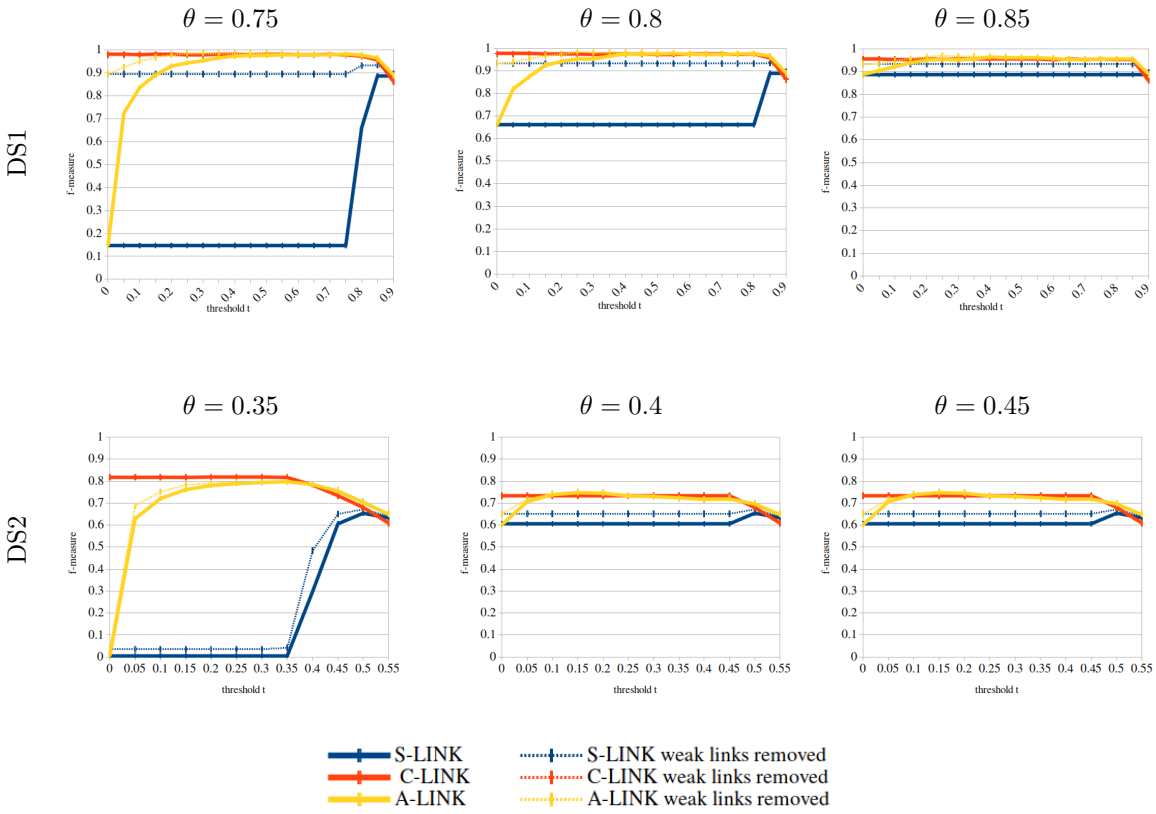
List of Figures

List of Tables

List of Algorithms

Appendix

Appendix A: Match Quality Results Before and After Weak Link Removal applying the Naive Implementation of Distributed HAC



Appendix B: Average Runtimes (seconds) for the Naive Implementation of Distributed S-LINK HAC

workers	DS3-5M	DS3-10M
16	39	78

Erklärung

Ich versichere, dass ich die vorliegende Arbeit mit dem Thema:

„A Distributed Hierarchical Clustering Algorithm for Multi-source Entity Resolution“

selbständig und nur unter Verwendung der angegebenen Quellen und Hilfsmittel angefertigt habe, insbesondere sind wörtliche oder sinngemäße Zitate als solche gekennzeichnet. Mir ist bekannt, dass Zuwiderhandlung auch nachträglich zur Aberkennung des Abschlusses führen kann. Ich versichere, dass das elektronische Exemplar mit den gedruckten Exemplaren übereinstimmt.

Leipzig, den

_____

Lucie David