Clustering Approaches for Multi-source Entity Resolution

Der Fakultät für Mathematik und Informatik
der Universität Leipzig
angenommene

Dissertation

zur Erlangung des akademischen Grades

Doctor Rerum Naturalium
(Dr. rer. nat.)

im Fachgebiet Informatik

vorgelegt von
M. Sc. Informatik Alieh Saeedi
geboren am 18. Mai 1986 in Mashhad/Iran

Die Annahme der Dissertation wurde empfohlen von:
1. Prof. Dr. Erhard Rahm, Universität Leipzig
2. Prof. Dr. Felix Naumann, Universität Potsdam

Die Verleihung des akademischen Grades erfolgt mit Bestehen der
Verteidigung am 1. Dezember 2021 mit dem Gesamtprädikat
Magna cum Laude

UNIVERSITÄT
LEIPZIG

Acknowledgments

I would like to express my sincere gratitude to all parties who supported me doing this PhD. The dissertation is the result of five years of being a PhD student in database group of Leipzig university and is mostly funded within the ScaDS and ScaDS.AI projects. I would like to express my special thanks to Prof. Dr. Erhard Rahm firstly for inviting me to Germany to work under his supervision, and secondly for his assistance at every stage of the research project. Furthermore, I want to thank Dr. Eric Peukert for his unwavering support in my research and illuminating many new basics for me. Their invaluable advice, continuous support, and patience during my PhD study encouraged me in all the time of my academic research and daily life.

I would like to extend my sincere thanks to my colleagues at database group for maintaining the joyful and supportive atmosphere. I would initially name Prof. Dr. Anika Groß and Dr. Ying-Chi Lin for their informational and emotional support when I was new in Germany which helped me feeling almost like home. Furthermore, I thank my former office mate Dr. Markus Nentwig who answered my scientific and technical questions in the first year of the study and for the inspiring discussions on the research topics. I additionally thank Dr. Victor Christen who is always ready to talk about data matching topics and also for his feedback on my dissertation. Furthermore, I would thank all former and current colleagues working on the Gradoop project such as Martin Junghanns, Dr. André Petermann, Kevin Gomez, and Christopher Rost for answering my emails and listening to my complaints. My special thanks go to the former colleague Dr. Mohammad Ali Rostami for the interesting discussions on graph topics and developing the SIMG-VIZ tool which helped me a lot in contributing new approaches.

I would like to thank my friend Dr. Morteza Moradi for giving me the courage to leave my home country and seeking new academic challenges and career opportunities abroad. My deep appreciation certainly goes out to my parents that accepted and endured my distance. I would also like to express gratitude to my sister Atefe who never stops believing in me. Last but not least, I thank my husband Ronald Rist for his support and patience, and his sustained effort for motivating me to finish my dissertation. I also thank him for the proofreading.

Leipzig, 30. Juni 2021 Alieh Saeedi

Abstract

Entity Resolution (ER) or deduplication aims at identifying entities, such as specific customer or product descriptions, in one or several data sources that refer to the same real-world entity. ER is of key importance for improving data quality and has a crucial role in data integration and querying. The previous generation of ER approaches focus on integrating records from two relational databases or performing deduplication within a single database. Nevertheless, in the era of Big Data the number of available data sources is increasing rapidly. Therefore, large-scale data mining or querying systems need to integrate data obtained from numerous sources. For example, in online digital libraries or E-Shops, publications or products are incorporated from large number of archives or suppliers across the world or within a specified region or country to provide a unified view for the user. This process requires data consolidation from numerous heterogeneous data sources which are mostly evolving. By raising the number of sources, data heterogeneity and velocity as well as the variance in data quality is increased. Therefore, multi-source ER, i.e. finding matching entities in an arbitrary number of sources, is a challenging task. Previous efforts for matching and clustering entities between multiple sources (> 2) mostly treated all sources as a single source. This approach excludes utilizing meta data or provenance information for enhancing the integration quality and leads up to poor results due to ignorance of the discrepancy between quality of sources.

The conventional ER pipeline is comprised of blocking, pair-wise matching of entities, and classification. In order to meet the new needs and requirements, holistic clustering approaches that are capable of scaling to many data sources are needed. The holistic clustering-based ER should further overcome the restriction of pairwise linking of entities by making the process capable of grouping entities from multiple sources into clusters. The clustering step aims at removing false links while adding missing true links across sources. Additionally, incremental clustering and repairing approaches need to be developed to cope with the ever increasing number of sources and new incoming entities.

To this end, we developed novel clustering and repairing schemes for multi-source entity resolution. The approaches are capable of grouping entities from multiple clean (duplicate-free) sources as well as handling data from an arbitrary combination of clean

and dirty sources. The multi-source clustering schemes exclusively developed for multi-source ER can obtain superior results compared to general purpose clustering algorithms. Additionally, we developed incremental clustering and repairing methods in order to handle the evolving sources. The proposed incremental approaches are capable of incorporating new sources as well as new entities from existing sources. The more sophisticated approach is able to repair previously determined clusters and consequently yields improved quality and a reduced dependency on the insert order of the new entities.

To ensure scalability, the parallel variation of all approaches are implemented on top of the Apache Flink framework which is a distributed processing engine. The proposed methods have been integrated in a new end-to-end ER tool named FAMER (FAst Multi-source Entity Resolution system). The FAMER framework is comprised of *Linking* and *Clustering* components encompassing both batch and incremental ER functionalities. The output of Linking part is recorded as a similarity graph where each vertex represents an entity and each edge maintains the similarity relationship between two entities. Such a similarity graph is the input of the Clustering component. The comprehensive comparative evaluations overall show that the proposed clustering and repairing approaches for both batch and incremental ER achieve high quality while maintaining the scalability.

Contents

List of Figures

List of Tables

1 Introduction

This chapter starts with motivating the topic of data integration and describing the corresponding workflows. The relevant subjects and background of data integration are discussed in Section 1.1.1 and Section 1.1.2. Section 1.1.3 then introduces the corresponding requirements and challenges. On this basis, scientific contributions of this dissertation are specified in Section 1.2. Finally, Section 1.3 gives an overview of the remaining work.

1.1 Motivation

Data integration is the practice of consolidating data from disparate sources into a single, unified view [100]. The main aim of data integration is delivering data to meet the information needs of different applications and processes. Data integration, for instance, is the initial step in getting the whole picture of an individual patient's electronic health record in health care. It furthermore enables analytics tools to produce businesses intelligence for companies, and facilitates better governing and managing the public sector for governments. Additionally, popular tools we use on a daily basis, such as search engines and query answering systems, integrate data from numerous sources in order to generate knowledge from data. Analogously, advanced AI tools that allow the user to turn a single query into an ongoing conversation, build and store a general knowledge of the world in a so-called knowledge graph [166] by collecting and combining entities and their relationships from multiple data sources [130].

Integrating heterogeneous data residing in different sources is a costly process that requires several steps [100]. The conventional data integration workflow depicted in

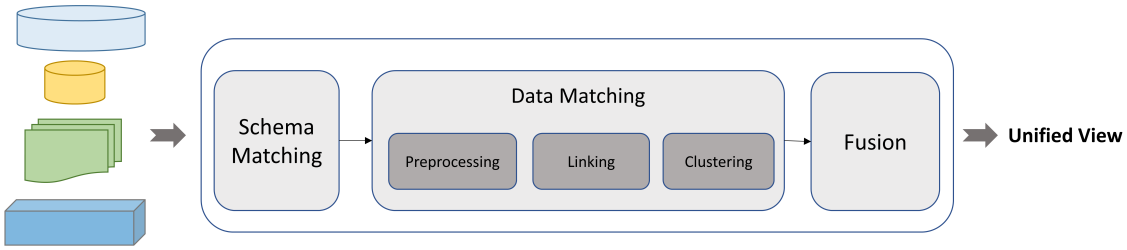


Figure 1.1: Data integration workflow

Figure 1.1 consists of three main steps [27]. The first step is *schema matching*. Schema consists of several attributes for describing data instances and schema matching is the process of identifying semantically identical schema elements [147]. For example, in two schemas camera1 (Manufacturer, Model, Resolution, Weight, Zoom) and camera2 (Manufacturer, ModelNo, Megapixels, Optical Zoom), schema matching generates the possible matches of camera1.Manufacturer ≈ camera2.Manufacturer, camera1.Model ≈ camera2.ModelNo, camera1.Resolution ≈ camera2.Megapixels, and camera1.Zoom ≈ camera2.Optical Zoom. The second step is *data matching* which is the task of identifying and matching entities that refer to the same entity across different sources or within a single source. A great amount of literature under different titles such as Entity Resolution (ER), record linkage and etc. has been devoted to data matching issues [20, 32, 58, 59]. Referring to the camera example, ER aims at finding identical cameras such as *Fujifilm FinePix S6800* with the same manufacturer, model number and exactly the same specifications within or across different sources. *Data fusion* [17] as the last step of data integration, merges groups of matched (identical) entities into a clean and consistent entity.

As illustrated in Figure 1.1, data matching or entity resolution includes several main steps. After some data cleaning actions refereed as *preprocessing*, the *linking* step determines matches by creating similarity links between instances (or/and uses the existing links). Then *clustering* is applied as a postprocessing step, as for more than two sources a binary linking of entities is not sufficient. Clustering additionally facilitates fusion so that all matches of the same entity that are grouped together can derive a fused entity representation [146]. Moreover, semantic metadata such as provenance (see [22] for the role of provenance in data quality) can be considered in ER-specific semi-supervised [6] clustering approaches that leads to improved quality of matches.

With the emerge of Big Data, all conventional data-related approaches and algorithms including data matching methods and techniques must be adapted or advanced to meet the relevant requirements. Big Data entails new challenges to the whole process of data matching. Therefore, we discuss the Big Data characteristics as well as the requirements and challenges of data matching for Big Data in the next section.

1.1.1 Big Data

The term Big Data refers to the great variety of large data sources that grow at increasing rates. The concept of big data has been articulated under three main V's [161]:

- Volume: refers to the sheer volume of data. The number and size of the datasets that need to be analyzed and processed continues to increase strongly [32, 67, 150].
- Velocity: points to the speed of data generation. The data sources are evolving continuously i.e. new data sources as well as new entities from new or existing data sources are incrementally disseminated.
- Variety: introduces all the structured and unstructured data that have the possibility of being generated either by humans or by machines.

Likewise, other V's such as Veracity and Value were defined to describe more aspects. The former is equivalent to quality and the latter refers to the ability to transform data to knowledge. Due to the significant role of Big Data applications in today's technology and economics, the story of defining new V's is still continuing.

Considering the above mentioned features, it is difficult or impossible to perform any computation including data matching process on data when the data is so large, fast and complex using traditional methods and technologies. To tackle this problem the data matching pipeline needs to be built on top of the available distributed storage and processing platforms. Moreover, in order to handle the evolving data, novel approaches should be designed and developed to integrate new entities to the already existing knowledge graph. The next problem with conventional data matching methods is that the majority of them consider at most two data sources which is insufficient to integrate data from numerous different data sources. For transcending this limitation, the last step of data matching workflow (clustering) must be thoughtfully considered. Therefore, in the next section we discuss the functionality of clustering in data matching process.

1.1.2 Clustering

Clustering aims at grouping entities into clusters, such that entities belonging to the same cluster are more similar to each other than they are to entities in the other clusters. A large number of clustering algorithms have been proposed for different applications. Detailed surveys of clustering methods can be found in [46, 182, 183]. Generally, clustering is an unsupervised task that determines entities of the clusters by estimating the

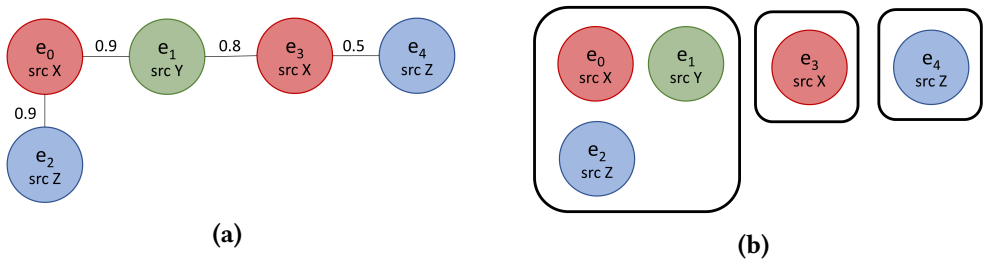


Figure 1.2: Clustering example

similarity or distance between them. In addition to the similarity information, the metadata or the available knowledge about the sources can guide the clustering process as a limited form of supervision. The resulting approach is called semi-supervised clustering [65].

Clustering is an essential component for multi-source data matching, because the linking (matching) step of data matching workflow determines the final matching status of each pair of entities individually. As a consequence, the entities that are connected via indirect links (transitive closures) and are assumed to be identical, may break some priori provenance-related constraints. Moreover, a long chain of connecting entities may form so that entities at the two ends of the chain have a very low similarity [27]. Therefore, an appropriate clustering algorithm as the last step of data matching process is applied in order to cut irrelevant and incoherent direct and indirect links while keeping the correct ones. Figure 1.2a depicts five entities from three sources X, Y and Z. Based on the prior knowledge all entities of source X are unique. Thus, none of the entities of source X are allowed to be matched together. But a long chain makes indirect connection between entity e_0 and e_3 (both from source X). Moreover, the entity e_0 in the middle of the chain and the entity e_4 at the end, are most probably dissimilar. A good clustering algorithm splits the chain in such a way that the constraints are satisfied (the constraint in this example is that each cluster must contain at most one entity from the duplicate-free source of X) and entities with low similarities get grouped in different clusters. In Figure 1.2b, a clustering algorithm grouped entities into three clusters so that each cluster contains at most one entity from source X and entities within each cluster are highly similar.

1.1.3 Challenges and Requirements

As mentioned above, data matching is the central component of the data integration process. This dissertation is focused on data matching solutions for multiple data sources using Big Data technologies. In the following different challenges are listed.

Grouping instances into clusters effectively and efficiently Integrating data from multiple (> 2) data sources, makes clustering an essential part for unsupervised data matching. Clustering should be applied as a postprocessing step after classification phase with the aim of grouping linked entities into clusters of entities so that each represents a unique entity. Dealing with numerous different sources, a clustering method furthermore should utilize the provenance information of entities in order to improve the match quality. In this thesis, it is assumed that duplicate-free sources are known prior to data matching. Therefore, the clustering approaches set constraints in order to avoid grouping entities of the same duplicate-free sources within the same cluster. Additionally, for further improvement of final clustering results, repairing approaches need to be designed and developed.

Support for dynamic data and data sources Velocity is one of the main aspects of Big Data. Thus, new technologies should handle incoming and processing of the new data in an efficient and effective way. Constant incoming of new data sources and new entities from existing sources need to be supported by developing incremental linking and incremental clustering and repairing approaches. The incremental approaches should be competent to add entities to an in-use knowledge graph i.e. the already integrated entities that are largely unaffected by new entities should not have to be re-integrated for every update. The incremental repairing approach furthermore should be able to use the new entities and sources to repair already determined clusters by reconsidering the former match decisions.

Scalable and distributed processing of data matching workflows With growth of data and the increase in number of data sources, building data matching pipelines on top of Big Data frameworks has become inevitable. The frameworks in the Big Data ecosystem facilitate distributed data storage and processing across arbitrary number of machines. They additionally provide scaling up to a desired degree by simple addition of machines to the cluster. Therefore, developing conforming algorithms and programs for an optimized and scalable execution of data integration pipeline has become an state-of-the-art research topic.

1.2 Scientific Contributions

The following contributions are derived from the initial analysis of challenges and requirements (mentioned above). All contributions are both peer-reviewed and published in scientific journals or proceedings of conferences.

Clustering approaches for entity resolution Performing data matching process across

multiple sources entails using clustering approaches in order to grouping entities from different sources in the same cluster. For this reason, we implemented existing generic-purpose clustering algorithms for our entity resolution task and compared them for our multi-source datasets in terms of both their efficiency (speed-up curves) and effectiveness. The results are published in ADBIS 2017 [158]. Due to the fact that data sources have diverse characteristics, awareness and utilizing provenance information by the clustering algorithm improves the matching result. Therefore, we developed the clustering scheme CLIP (Clustering based on LInk Priority) that assumes all sources as duplicate-free. Moreover, in order to correct the results of clustering methods such as overlapping clusters, we additionally developed the repairing method RLIP (cluster Repair based on LInk Priority). Both approaches produce superior results compared to generic-purpose clustering schemes and were published and presented in ESWC 2018 [160]. The publication succeeded to win the best research paper award. All mentioned approaches were further compared to novel clustering schemes for grouping entities of multiple clean sources (named as *Split* and *SplitMerge*) and the results were published in CSIMQ paper in 2018 [157]. Furthermore, we developed clustering algorithms that are able to consider the degree of dirtiness (duplicate existence rate) of datasets to determine the output. The approaches MSCD-AP (Multi-Source Clean Dirty Affinity Propagation) and MSCD-HAC (Multi-Source Clean Dirty Hierarchical Agglomerative Clustering) are extended variations of basic Affinity Propagation clustering and Hierarchical Agglomerative clustering that are able to cluster datasets of combined clean (duplicate-free) and dirty sources. The MSCD-AP approach is published and presented in BTW 2021 [102] and MSCD-HAC approaches are submitted to KEOD 2021 [156].

Incremental clustering approaches In the era of Big Data, data sources are evolving. Moreover, new data sources constantly need to be integrated with the existing knowledge graph. For handling this dynamic growth, we developed approaches for incremental addition and integration of data without recalculating already existing matches. One approach called MAX-Both inserts the new incoming entity either to the most similar existing cluster or creates a new singleton cluster. However, this approach typically suffers from a strong dependency on the order in which new entities are added. In particular, wrong cluster decisions, e.g., due to data quality problems, will not be corrected and can lead to further errors when new entities are added. The overall quality can thus be much worse than for batch entity resolution where all entities are simultaneously integrated. We therefore propose and evaluate a novel approach called n-depth reclustering (nDR) for incremental entity clustering and repairing. The letter n refers to the portion of the existing graph that is modified. The approach reduces the dependency on the order in which the new entities and sources are added. Moreover, it is capable of repairing the

previously matched decisions while integrating the new entities and sources. The incremental approaches were published and presented in ESWC 2020 [159].

FAMER framework on top of Apache Flink Handling data from numerous sources entails developing scalable methods. Therefore, this dissertation focuses on developing scalable approaches for multi-source entity resolution. We developed an end-to-end framework for entity resolution on top of Apache Flink called FAMER (FAst Multi-source Entity Resolution system). The input of FAMER is data from multiple sources and the output is sets of clusters, each representing a unique entity. FAMER constitutes of two main modules of Linking and Clustering that both support batch and incremental data matching. As an open-source software, FAMER is available on GitLab server of the informatics institute / Leipzig university[1]. FAMER additionally benefits from a visualization tool that enables the user to analyze precomputed similarity graphs and clusterings. The tool is presented in EDBT 2018 [155]. Furthermore, FAMER succeeded to win the KDD DI2KG data matching challenge 2019 [131].

1.3 Structure of Thesis

This dissertation consists of five further chapters as follows:

Chapter 2 gives an introduction to the entity resolution topic and the conventional pipeline. We then give an overview to the current entity resolution tools and their comparison. The chapter further defines the relevant concepts and categorizes and explains ER-specific clustering algorithms briefly. It additionally elaborates on the concept, background, and relative tools of distributed data processing. In the end, quality measures to evaluate the effectiveness of clustering results are introduced.

Chapter 3 introduces FAMER framework. It explains all modules and sub-modules of FAMER in detail. The chapter elaborates on both batch and incremental pipelines. Finally, it discusses the technical details and functionalities of SIMG-VIZ tool. Some screen-shots of the visualization output are shown for illustration.

Chapter 4 presents the existing clustering algorithms as well as our innovative approaches for clustering entities from multiple clean (duplicate-free) sources. In this chapter we comprehensively evaluate the cluster quality and scalability of the new approaches for different datasets and compare them with previously proposed clustering schemes.

[1]FAMER repository https://git.informatik.uni-leipzig.de/dbs/FAMER

Chapter 5 proposes innovative clustering approaches for grouping entities from a combination of clean (duplicate-free) and dirty data sources. In order to provide the scalability, the parallel variations of the new approach are developed as well. We comprehensively evaluate match quality, runtimes and scalability of the new approaches for different datasets and compare them with previous clustering schemes.

Chapter 6 focuses on handling evolving data sources. It proposes two incremental approaches to integrate data from dynamically changing data sources. One naive approach inserts entities to the existing clusters or creates new clusters and a sophisticated approach that repairs existing clusters with the new incoming entities and sources. The incremental approaches are evaluated for datasets of four domains in terms of cluster quality and runtime efficiency. Additionally, a comparison to a previous approach for incremental cluster repair and with batch entity resolution is provided.

Finally **Chapter 7** - Conclusion and Outlook - concludes the results of this dissertation and provides concepts and ideas for future work.

2 Background

This chapter extends the required background and related works of the dissertation. Section 2.1 elaborates on Entity Resolution (ER) topic and illustrates the challenges and issues by an example. The formal definition of the problem is stated in Section 2.1.1. Section 2.1.2 furthermore explains the conventional pipeline of the ER and narrates the details of each step. Finally, an overview of the current ER tools and a comparative comparison between them is described in Section 2.1.3.

Section 2.2 introduces the clustering topic by stating a formal definition. Then, Section 2.2.1 introduces the clustering algorithms that are already used in ER applications.

Section 2.3 gives an overview of the architecture of distributed systems and the programming models. The current distributed storage and processing frameworks and their programming paradigms as well as a brief comparison of them are presented in Section 2.3.1 and Section 2.3.2. Finally, Section 2.3.3 introduces Apache Flink[1] and the relevant libraries which have been used for implementing contributions of this dissertation. The chapter is closed with description of the quality measurements for evaluating the methods (Section 2.4).

2.1 Entity Resolution

Data matching known as Entity Resolution (ER) is the task of identifying and matching individual entities from disparate data sources that refer to the same real-world entities or objects. Other names of data matching are listed as record or data linkage given by

[1] https://flink.apache.org/

health researchers and statisticians. The terms data, record or object matching, entity resolution, co-reference resolution, object identification, data reconciliation, citation or reference matching, and deduplication are denominated by computer scientists from different fields [27].

Christen [27] summarized the long history of using computers for data matching by both database community [73, 74, 110, 111, 147] and statisticians [47, 123, 124, 133, 181]. With the growth of data volume and increased importance of data, data matching has become more pervasive and influential for organizations. In the last decades, machine learning approaches as well as natural language processing and graph-based approaches have been developed to tackle the challenges of data matching for millions of entities and furthermore improve the matching quality [15, 58, 59]. Performing entity resolution leads to less usage of storage and computational effort [27]. Moreover, deduplicated data is needed in order to performing data mining tasks and inferring useful information toward finding improvement points for organizations and businesses [40].

Generally, ER is a challenging task due to several reasons. The first reason that was existing from the early days of emergence of data deduplication is solving the computational complexity problem. Matching each entity of one source with all other entities of the same source and all entities of the other sources is an exhaustive computation. On the other hand, with the advent of big data, data volume has been grown in both aspects of the number of sources and the size of each source. Furthermore, with dynamic data sources, newly collected data or even a new source might be continuously added to the data. The second reason is that data sources have heterogeneous schema which makes identifying duplicates of different sources a very challenging task. Thirdly, the data sources are extremely different in quality. The accuracy of data may significantly differ from source to source. Additionally, there are plenty of other issues like lack of training data containing the match status or privacy and confidentiality problems which are out of scope of this dissertation [27, 43].

As a motivating example, we consider a price comparison website that skim through multiple e-commerce sites to collect data about products and services like prices, descriptions, features, reviews and etc. The buyer can then compare the listings based on price, features, shipping costs and other desired features to find the best deal. In order to provide such a listing, it is needed to collect data from many sources, perform the data matching process and present a specific product with all the providers. To illustrate the data quality and source heterogeneity problems in a multi-source ER scenario, we show in Table 2.1[2] three matching *Fujifilm* camera products from different sources. As shown

[2]The camera data is from the dataset of the ACM SIGMOD 2020 Programming Contest. http://www.inf.uniroma3.it/db/sigmod2020contest/index.html

Table 2.1: Camera entities from three data sources with challenges for data matching

source	**www.ebay.com**
”<page title>”	”Fujifilm FinePix S6800 16 2 MP Digital Camera Black 074101020977 | eBay”
”brand”	”Fujifilm”
”megapixels”	”16.2 MP”
”model”	”S6800”
”mpn”	”4004857”
”optical zoom”	”30x”
”type”	”Point & Shoot”
”upc”	”074101020977”
source	**www.pcconnection.com**
”<page title>”	”Buy Fujifilm FinePix S6800 Digital Camera, 16MP, 30x Zoom, Black Cameras - Digital - Point & Shoot 16303014 today at PC Connection”
”returns policy”	”This product is subject to our return policy. Please see our complete return policy for details.”
”warranty labor”	”Call for Warranty”
”warranty parts”	”Call for Warranty”
lcd viewer	3 Inch
effective megapixels	16
source	**www.mypriceindia.com**
”<page title>”	”Fujifilm FinePix S6800 Price In India, Bangalore, Hyderabad, Delhi, Chennai, Mumbai, Pune, Kolkatta”
”aperture range”	”F3.1 (W) - F5.9 (T)”
”auto focus”	”Yes, Single, Continuous, Center, Tracking, Multi, Area, TTL contrast AF, AF assist illuminator”
”built in flash”	”Yes”
”camera resolution”	”16.2 MP”
”color filter”	”Yes, Primary (RGB) Color filter”
”conitnous shots”	”Yes, High (8.0 fps (max. 10 frames)), Medium (5.0 fps (max. 10 frames)), Low (3.0 fps (max. 10 frames))”
”delete function”	”Yes”
”digital zoom”	”2x”
”external flash”	”No”
”face detection”	”Yes”
”flash range”	”0.4 - 7.0 m (W), 2.5 - 3.6 m (T)”
”image format”	”JPEG (Exif Ver 2.3)”
”image stablizer”	”Yes, CMOS Shift Type”
”iso rating”	”100 - 12800”
”lens type”	”Fujinon 30x Optical Zoom Lens”
”macro mode”	”Yes, 5 cm 300 cm (W), 180 cm 300 cm (T)”
”maximum shutter speed”	”1/2000 sec”
”metering”	”Yes, TTL 256-zone metering, Multi, Spot, Average”
”minimum shutter speed”	”0.25 sec”
”optical zoom”	”30x”
”other focus features”	”Lens Construction (11 groups, 15 Lenses) Normal Focus Range (15cm to infinity(W), 3.0m to infinity(T))”
”red eye reduction”	”Yes”
”self timer”	”Yes, 10 sec. / 2 sec. Delay”
”video format”	”AVI (Motion JPEG)”
”video resolution”	”1920 x 1080 pixels at 60 fps”
”white balancing”	”Fine, Shade, Fluorescent light (Daylight) Fluorescent light (Warm White), Fluorescent light (Cool White) Incandescent light, Custom”
”screen size”	3” (more than 62%)

in the table, there is a significant difference in the set of properties and property values. For example, the second entity owns the property *returns policy* while the other two cameras do neither contain this property nor the corresponding value. Moreover, the same property values are not represented similarly in different entities. For example, in the first camera the property *megapixels* with the value *16.2 MP* is represented as *”effective megapixels”: ”16”* for the second camera and *”camera resolution”: ”16.2 MP”* for the

third camera. The difference in the values may arise from misspelling or wrong values either different ways of data representations like different measuring units or different symbols. For example, the properties *lcd viewer* and *screen size* in the second and third entities have different values of *3 Inch* and *3" (more than 62%)*.

As mentioned in [27], data matching does neither include the extraction of entity information from unstructured documents nor performing schema matching. It is assumed that records (entities) to be matched are stored in well-defined files or tables. Although in Chapter 4, we utilize a few schema matching heuristics in order to be able to perform the task of ER on unstructured datasets as well. Nevertheless, schema matching is not the main topic of this dissertation but the focus is on designing and developing methods and techniques for clustering entities from different sources with different quality. Furthermore, incremental addition of new entities from new or existing sources are investigated in this dissertation.

2.1.1 Problem Definition

The input of Entity Resolution (ER) is data from a set of different data sources S_1, S_2, ..., S_m. Each data source S_i consists of entities e_1, ..., e_n where each entity e_j is characterized by a set of attributes $\mathcal{A}$. For instance, the first entity listed in Table 2.1 is characterized by the attributes *<page title>*, *brand*, *megapixels*, *model*, *mpn*, *optical zoom*, *type*, and *upc*.

The output of ER is a set of clusters C_1, C_2, ..., C_k where each cluster consists of entities representing the same real-world entity.

2.1.2 General approach

Figure 2.1 depicts the general conventional workflow of the ER task. The input is data from one or multiple sources that may differ enormously in size and quality and the output is a set of clusters, each of which containing the same real-world entity. The shown preprocessing step entails data cleaning actions such as handling missing values, smoothing noisy values, and identifying and correcting inconsistent values [27]. Furthermore, schema matching can be applied to identify matching properties that can be used for determining the similarity of entities for ER. To match the cameras shown in Table 2.1, preprocessing may include transforming values into the same unit, lowercasing strings, applying canonical abbreviations to harmonize property values, and assigning the same name to matching properties to facilitate similarity computations. The blocking step prevents comparing irrelevant entities with each other. For instance, in our running camera example (Table 2.1), cameras with different manufacturers will

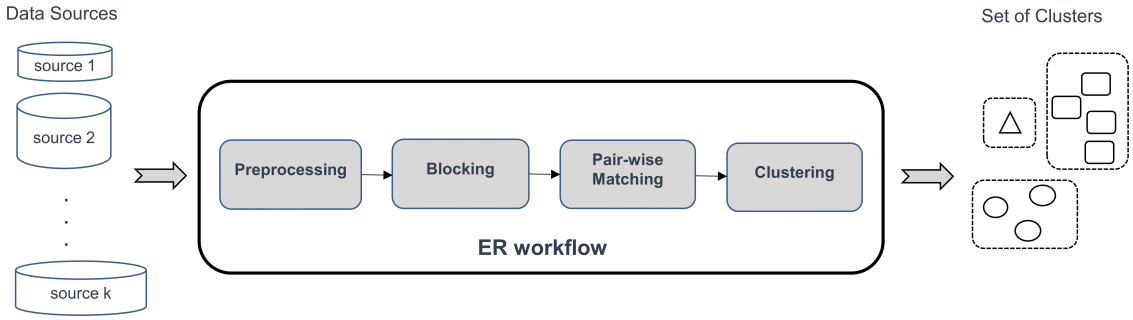


Figure 2.1: Entity resolution workflow

be placed in different blocks in order to avoid comparing *Fujifilm* cameras with cameras from other manufacturers such as *Canon* or *Nikon*. Then in the pair-wise matching step the similarity of candidate pairs are computed by applying a set of similarity methods on the property values of the entities. Finally, the clustering step uses computed similarities to group the same entities in the same cluster. Clustering facilitates fusion of the same entities into one unique representative entity. The main ER steps of blocking, matching and clustering will be discussed in the rest of this section.

Most of the contributed approaches for ER almost fit into this workflow. However, some deep learning based ER approaches [10] contemplate schema matching as an implicit part of ER process. They consider a single step as a neural network for both feature extraction and pair-wise matching. Additionally, another line of research called collective ER [14] performs matching almost in an iterative manner, because matching status of each pair is influenced by status of neighbour pairs or other information from neighbors.

Blocking aims at improving performance and scalability by avoiding that every entity has to be compared with every other entity for determining matching entity pairs. Such a naive approach has a quadratic complexity with $\frac{n\cdot(n-1)}{2}$ comparisons for n entities. Therefore, blocking methods intend to restrict the comparisons only to those pairs that are more likely to match. Standard Blocking (SB) [47] and Sorted Neighborhood (SN) [74] are two popular blocking methods that both utilize a so-called *blocking key* to group entities. The key is mostly specified by a human expert and is the result of a function on one or several property values, e.g. the initial three letters of the <*page title*> property for the camera example (Table 2.1). SB restricts the comparisons to the entities with the same blocking key while SN moves a sliding window of a fixed size over the entities and considers the entities within a window as candidate matches. Since real data is noisy, generating one blocking key per entity may not allow finding all matches. Hence,

it can be necessary to generate multiple blocking keys per entity, leading to multi-pass blocking [73, 90] that can find more matches and thus improve recall over the use of single blocking key. Some methods also try to improve precision along with efficiency by limiting the maximal block size or the number of candidate matches [48]. Since determining suitable blocking keys can be a tedious and difficult task, approaches based on both supervised [16, 60] and unsupervised [86] Machine Learning (ML) have been proposed to learn blocking keys. Papadakis et al. [140] gives a comprehensive overview of blocking techniques.

To further improve runtime, both SB-based and SN-based ER methods can be parallelized to utilize multiple machines in a cluster. The major challenge in parallelizing a blocking technique is achieving good load balancing because the sizes of the output blocks can be highly skewed. Kolb et al. propose the load-balanced SB [89] and SN [91] based on MapReduce framework [39]. For SB, two methods named BlockSplit and PairRange are proposed and implemented. BlockSplit dedicates an equal number of entities to each *reducer* by breaking large blocks into smaller sub-blocks while PairRange distributes the comparisons on the reducer nodes evenly. For load-balanced SN, JobSN and RepSN methods are proposed [91]. The former approach employs a second MapReduce job to generate boundary entities while the latter enumerates the entities to assign evenly-sized entity ranges to every *reducer*.

For semi-structured, textual data or in absence of an aligned schema across sources, schema-agnostic token-based blocking approaches have been proposed. The basic Token Blocking (TB) [137] generates a candidate match based on the common tokens of property values of a pair. Like with traditional blocking methods, scalability can be improved by a MapReduce-based implementation [140]. For load balancing, Chu et al. [33] propose an approach that distributes blocks among computing units. Since the basic TB may create too many candidate pairs, newer schema-agnostic approaches reduce them by pairing tokens from synthetically similar properties, considering only selected properties, or comparing only the entities of the same type [140]. Furthermore, block post-processing approaches such as meta-blocking [139, 165] can largely reduce the number of candidate matches. A very different approach [129] totally ignores property values but determines candidate matches based on relations between entities.

Pair-wise Matching The decision on whether a pair of entities is a likely match is based on the similarity of the two entities which is determined by one or multiple similarity functions. These functions mostly determine the similarity of property values depending on the data type (string, numerical, date, geographical coordinates etc.). Typically, several such similarity values need to be combined to derive a match or non-match deci-

sion. Traditional approaches such as threshold-based or rule-based methods classify the matching status for each pair independently. In threshold-based classification, a specified threshold considers all pairs with similarity above a certain value as matches. On the other hand, in rule-based classification, a rule specifies a match predicate consisting of property-specific similarity conditions that are combined with logical operations [27]. For the camera example (2.1), the match decision may be based on the similarity of the properties *page title* and *megapixels* although the latter property is not present for all entities shown.

Manually determining the properties to match, similarity functions and similarity thresholds is a complex task, especially for heterogeneous and noisy data. Hence, a better alternative is often to apply supervised ML approaches to find optimal match configurations to determine matching entity pairs. These approaches can utilize traditional ML techniques such as SVM, logistic regression or random forests [94] but also newer approaches based on deep learning. Barlaug et al. [10] provides an overview about ER proposals utilizing deep neural networks including the approaches DeepER [45], DeepMatcher [113] and Hi-EM [188]. These approaches typically utilize embeddings for textual property values by transforming either words or their characters to numerical representations that preserve the semantic similarity between property values. Word embeddings are able to convert a long sequence to a short one, but they can not necessarily cover all possible words for specialized domains. The generation of embeddings can make use of pretrained models such as word2vec [109], GloVe [143] or fastText [18] that are derived from large corpora such as Wikipedia [10]. Another line of research called collective ER [14] uses both property value similarity and relational information for determining the similarity of two entities. Here, the ER process is mostly iterative because changes in similarity or matching status of one pair affects the similarity value of the neighbouring pairs. Such approaches are more difficult to scale than the standard approaches where candidate pairs are compared independently. To better scale collective ER, Rastogi et. al. [149] propose a generic approach that executes multiple instances of the matching task and constructs the global solution by message passing.

Clustering The matches determined by the pair-wise similarity calculations are often contradicting and therefore only match candidates. The final matches are determined by applying a clustering approach on the set of candidate match pairs that form a similarity graph where matching entities are linked with each other. The baseline approach for entity clustering is to determine the transitive closure or connected components over the match links. Note, that general clustering algorithms like K-means that need a predefined number of clusters are not suitable for ER.

Connected components does not consider the strength or similarity of candidate matches and can thus cluster even weakly similar entities. There is a large spectrum of alternatives that are suited when the input constitutes of two duplicate-free sources [56, 96]. For deduplicating a single source, Hassanzadeh et al. [68] comparatively analyzed several clustering algorithms. A comprehensive overview of ER-specific clustering approaches is given in Section 2.2.

2.1.3 Overview of The Current Tools

Different ER systems developed by research community or industry consider different scenarios for input data sources. A traditional ER problem mostly solved by database community is matching entities of two tables against each other where each table is duplicate-free. Another scenario is deduplicating a single source. But in Big Data applications, data is in fact from multiple sources (> 2) that some of them are duplicate-free and some are dirty.

The methods deployed for each step of ER highly influence the result quality. Selecting among the variety of existing methods or proposing new methods is highly dependent on input data features. Therefore, any meta data about input entities such as provenance facilitate the ER process. Interestingly, different input scenarios can be converted to each other. For example, in case of having multiple dirty sources, each source can be separately deduplicated and then the problem is reduced to integrating multiple duplicate-free sources. However, the effort is immense and the result is not necessarily successful [131], because a wrong decision in one source is propagated to all other sources which lowers the precision significantly.

Table 2.2: Comparison of ER Frameworks

		Big Data aspects				#Input sources		
system	Birth	Scalability	Incremental	Clustering	Open source	1	2	>2
Febrl [28]	2008	-	-	-	✓	✓	✓	-
FEVER [93]	2009	-	-	-	-	-	✓	-
Silk [171]	2009	✓	✓	-	✓	-	✓	-
Dedoop [88]	2010	✓	-	-	-	-	✓	-
Limes [125]	2011	✓	-	-	✓	-	✓	-
Woo [12]	2013	✓	✓	✓	-	✓	✓	✓
FAMER	2016	✓	✓	✓	✓	✓	✓	✓
Magellan [41]	2016	✓	-	-	✓	-	✓	-
JedAI [141]	2017	✓	-	✓	✓	✓	✓	-

The focus of this dissertation is to cover all possible input scenarios in clustering step of the ER pipeline for Big Data applications. To give an overview of the currently available ER frameworks, Table 2.2 lists a number of known ER tools with a focus on Big Data requirements such as scalability, providing clustering algorithms, supporting dynamic data and different input scenarios.

As listed in Table 2.2, the more recent systems aim at encountering Big Data and therefore they are scalable. Woo supports both batch and incremental ER and along with JedAI provides the user with clustering algorithms. Some of the systems provide deduplication inside one single dirty source while others match entities across two sources. The frameworks that perform binary matching assume that both sources are clean. Therefore, they do not link entities inside each source but it may happen that one entity from one source is linked to several entities form the other source. Exceptionally, in the binary matching mode, JedAI guarantees one to one mapping by its binary clustering algorithms such as Hungarian Algorithm [96], Unique Mapping Clustering [98], and Best Assignment Clustering[3]. Woo is designed in industry to perform ER for hundreds of millions of entities from multiple input sources. Obviously, in special cases the number of input sources can be one or two, however it does not consider the information about cleanness status of the input sources. Woo is not open source and there is little information in details in the corresponding publications. The refinement phase of Woo which performs clustering uses generic-purpose clustering schemes. Correlation clustering [3] is specifically mentioned as the most effective clustering approach they used. They did not report any contribution in developing a novel clustering algorithm.

In Chapter 3, we introduce the framework FAMER that is able to perform scalable ER for entities from multiple sources. It further considers cleanliness status of the sources in both *pair-wise comparison* and *clustering* phases. The clustering step deploys novel clustering and repairing algorithms for grouping entities from multiple duplicate-free sources as well as a combination of duplicate-free and dirty sources. Moreover, FAMER benefits from incremental approaches that are capable of incorporating new sources as well as new entities from existing sources.

2.2 Clustering

Clustering or cluster analysis is the general task of separating a finite unlabeled dataset into a finite and discrete set of natural, hidden data structures [183]. Clustering aims at

[3]It is an efficient, heuristic solution to the assignment problem in unbalanced bipartite graphs [1]

putting objects or entities in the same cluster such that they are more similar to each other than to those in other groups.

Given a set of data points $X = x_1, x_2, ..., x_n$, a clustering algorithm F groups data points into a set of clusters $C = c_1, c_2, ..., c_k$ such that $\bigcup_{i=1}^{k} c_i = X$. The number of clusters, k is not necessarily predefined. But a group of clustering algorithms like *k-means clustering* need k as an input parameter. Thus, they produce different clustering results with different values of k. Similarly, some clustering algorithms set a maximum or minimum size constraint for the clusters. If $\bigcap_{i=1}^{k} c_i \neq \emptyset$, then the clustering algorithm F generates overlapping clusters. Ideally, the overlapping clusters are not desired, because in most applications a clear separation of entities is demanded.

Clustering has a data-driven nature. There is a large number of clustering algorithms proposed to group data points in different applications. Evidently, the definition of the cluster as well as finding the similarity measure of data points vary in different applications. Therefore, there is no clustering algorithm that correctly finds the clusters in all given data. Due to these facts, having any external or side information beside the similarity measure between data points improve the clustering results extremely [77].

As depicted in Figure 2.1, the clustering in ER process is considered as a post-processing step after pair-wise matching. Therefore, the input to the clustering is usually deemed as a similarity graph. Furthermore, considering cleanliness status of the input sources, the output of clustering should satisfy source-consistency constraint. These concepts are defined as follows:

Similarity graph A similarity graph $\mathcal{G}(\mathcal{V}, \mathcal{E})$ is a graph in which the vertices $\mathcal{V}$ represent the entities and the edge $(v_i, v_j) \in \mathcal{E}$ exists only if the similarity θ of v_i and v_j is higher than a minimum threshold value. There is no direct link between entities of the same clean source. Figure 2.2a depicts seven entities from three sources *X*, *Y* and *Z*. The entities are linked via edges which contain a similarity value higher than 0.75. Since it is assumed that source *X* is duplicate-free, entities of source *X* are not directly linked.

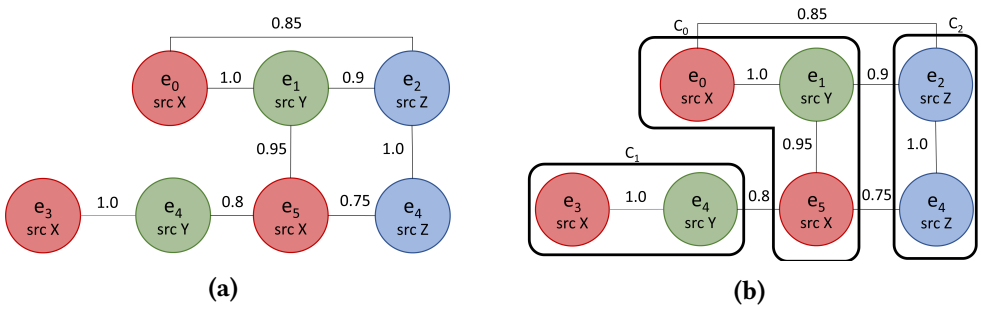


Figure 2.2: Clustering example

Source consistent cluster A cluster that contains at most one entity from each clean source is called a source-consistent cluster. In Figure 2.2b, the cluster c_0 is source-inconsistent since it contains two entities (e_0 and e_5) from source *X*. The other two clusters are source-consistent.

The rest of this section focuses on the clustering algorithms that were proposed or applied by research community in order to solve the ER problem.

2.2.1 ER Clustering Algorithms

Since the number of unique real-world entities is not defined or specified, the clustering algorithms that require the predefined number of clusters are not applicable for ER process. Hassanzadeh et. al [68] deploy string similarity join on datasets of single sources in order to generate the input similarity graph. Then as the final step of ER process, they apply several unconstrained general-purpose clustering algorithms on the similarity graphs with different minimum threshold values. They finally sort the chosen clustering algorithms based on different efficiency and effectivity criteria.

Christen [27] explains that instead of specifying a global minimum threshold for similarity graphs, all computed similarity values should incorporate in the final match decisions. Therefore, clustering algorithms similar to density clustering that group entities based on all similarity values are preferable. Other possibilities are the algorithms that iteratively remove edges till a constraint is satisfied or the approaches that decide about matches based on not only the single similarity value but a vector comparison.

JedAI [141] is an open source ER tool. It has implemented clustering methods for binary matching as well as matching inside a single source surveyed in [68]. Another Big Data ER tool, named Woo [12] refines the matches in the last step by Correlation Clustering [3]. In the following the clustering approaches for ER are explained in details.

Clustering two clean sources

Having two clean sources, the clustering algorithm should make a [0,1][0,1]-mapping. In this case the problem is usually solved by the solutions to assignment problem. Popular representatives are Hungarian Algorithm [96] and the solution to the Stable Marriage problem [56]:

Hungarian Algorithm [96] is an optimization algorithm that solves the *assignment problem* in polynomial time. The problem consists of finding a way to allocate certain available resources (machines or people) to carry out certain tasks at the lowest cost. It is assumed that each resource is allocated to a single task and each task is executed by only one of the resources. Assignment problem is one of the fundamental problems of

combinatorial optimization in the field of optimization or operations research in mathematics.

Stable Marriage [56] matches the entities of two equal size sets. The algorithm guarantees a stable matching such that for entity x and y that are not matched together, it is impossible that both prefer each other over their current matches.

Clustering multiple clean sources

SplitMerge Nentwig et. al [120, 121] propose a novel holistic approach for clustering-based link discovery for multiple duplicate-free data sources. The approach utilizes existing links for initial connected components clustering. It then in the two subsequent phases of cluster decomposition and merging refines the clustering results.

Clustering single dirty source

Connected components also referred as *transitive closure* considers two entities in the same cluster if there is a path between them. The algorithm does not remove any edges from the input graph. Therefore, it results in the highest recall with the cost of very low precision.

Center [71] processes edges in descending order of similarity values. For each edge, the unclustered entities at both ends are grouped together as a cluster and one of them is selected to be the center of the cluster randomly. In all the subsequent decision, if any entity is connected to a center will be assigned to the cluster of that center and if it is connected to a clustered entity which is not center, the decision about that entity is left to the next edges. The Merge Center algorithm [69] performs the same like Center but it merges two clusters if an entity in one cluster is directly linked to the other cluster center as well.

Star [4] is based on the idea of finding a minimum clique cover with maximal cliques on the resulting graph. Since the problem is NP-complete, a heuristic approach promises to produce the same results by forming dense star-shape clusters. The algorithm initially computes the degree for each vertex of the similarity graph. Then in each iteration, the unassigned vertex with the highest degree becomes center and all its direct neighbors are assigned to its cluster. The algorithm terminates when all vertices are assigned to a cluster. The clustering can result in overlapping clusters.

Correlation Clustering [8] considers + and - edges depending on whether the entities at the two ends of the edge are rated as similar or dissimilar. The algorithm aims at a clustering that maximizes the number of + edges within clusters and the number of - edges between clusters. It can be identically formulated as a minimization problem. Since it is an NP-hard problem, many approaches approximated the optimal solution.

Markov Clustering [169] clusters entities based on simulation of stochastic flow in graph. It assumes that a region with strong connectivity is a cluster. Therefore, it strengthens the amount of flow there and analogously weakens it where the connectivity is poor. The iterative process of manipulating the flow causes the underlying cluster structure to appear.

Cut Clustering is based on the max flow-min theorem [50]. It implies splitting a graph into two partitions with a minimum cut. The approach is further utilized for partitioning a graph into multiple clusters [49].

Articulation Point For an undirected graph, an articulation point is a vertex that removing it, increases the number of connected components. The Articulation Point clustering [9], finds all articulation points of the graph and clusters the graph by removing the incident edges of them. It may result in overlapping clusters.

Maximum Clique In graph theory, a clique is a subset of vertices of an undirected graph such that every two distinct vertices in the clique are adjacent. The clique with the largest possible number of vertices is called a maximum clique. The maximum clique clustering [44], finds the maximum cliques and removes them from the graph. Each maximum clique is assumed as a cluster. It repeats the process in an iterative way until all vertices are removed from the graph. The Extended Maximum Clique Clustering (EMMC) [44], relieves the clique definition to near-clique. The cliques with many edges to outside vertices are extended and assumed as a cluster. A vertex outside of a clique must have a minimum connectivity percentage to other cluster members or to the core clique. Similarly, GCluster [172] defines a new concept called δ-clique in which all vertices are connected to at least $\delta(v - 1)$ vertices. It additionally considers cohesion which is the sum of weights of the edges in the subgraph. Another clustering algorithm that utilized the concept of clique is called Global Edge Consistency Gain [44]. It considers all possible triangles in a graph and assumes all three as a cluster only if there are three duplicate edges between them. If there is any no-duplicate edge, the algorithm tries to switch the edges status in order to make an consistent triangle or splitting it. The process of switching edge status is continued until a clique is formed or a singleton is remained which is assumed as cluster and removed from the graph.

Hierarchical Clustering [79] groups data by creating a multi-level hierarchy tree. The tree allows to generate clusterings at different threshold degrees without running the algorithm again. Yan et al. [184] propose a novel variant of the hierarchical clustering that avoids *hard-conflict* inside clusters. The cluster-level *hard-conflict* is determined by comparing attributes of candidate merging clusters. If the similarity value for any attribute is less than the predefined threshold a *hard-conflict* is detected and therefore merging the candidate cluster pair is avoided.

Affinity Propagation [53] groups entities by identifying so-called exemplars. An exemplar is the entity that best represents all the entities of a cluster. The non-exemplar entities are assigned to the most appropriate exemplar. The goal of AP is to find exemplars and cluster assignments in a way that the sum of similarities inside clusters are maximized. In [63], AP is solved by the iterative max-sum algorithm on a factor graph. The factor graph is a bipartite graph between the exemplar assignments (variable nodes) and so-called factor nodes representing two constraints, called the g- and h-constraints. This method allows to add more constraints in order to have a novel algorithm that fits an specific application.

FAMER supports the generic-purpose clustering schemes such as connected components as the basic method, Center, Merge Center, two variations of Star, CCPivot which is a variation of Correlation clustering, Hierarchical clustering, and Affinity Propagation. SplitMerge is additionally considered for the comparative evaluations.

2.3 Distributed Data Processing

A distributed system is comprised of independent computers (with no shared memory) that coordinate their actions to achieve a common goal. The computers communicate by passing messages to each other [167]. Distributed computing exploits distributed systems to solve a problem with massive amounts of data and computation. It divides the computation into many tasks, so that each task is small enough to be solved by one or more computers that work at the same time (in parallel). Distributed systems are used for practical and scalability reasons. Moreover, a distributed system can provide more reliability than a non-distributed system, as there is no single point of failure.

In the following we elaborate on the most poplar software techniques and models that implement distributed computing systems. The advantages and drawbacks as well as a comparison of them is explained. Finally, a brief overview of the Apache Flink framework [23] is given. The distributed algorithms of this dissertation are implemented on top of Apache Flink.

2.3.1 Apache Hadoop

Apache Hadoop[4] is an open-source software that facilitates using a distributed system (cluster of computers) for performing massive amount of computations. The framework guarantees automatic handling of hardware failures. Apache Hadoop comprises of two main parts. Firstly, the storage part, known as Hadoop Distributed File System (HDFS)

[4] https://hadoop.apache.org/

that supports storing very large files across nodes (computers) in a cluster of many computers. The HDFS splits files into blocks with a configurable size and distributes them across nodes in a cluster. By default each block has multiple replicas to ensure reliability. Secondly, the processing engine which is based on the MapReduce [39] programming paradigm. In order to ensure fast and efficient parallel processing, Hadoop transfers the packaged code into the nodes. This approach is known as data locality [175].

Hadoop Distributed File System HDFS is a highly available file system for storing very large amounts of data on the file systems of several computers. The Hadoop cluster computers are organized as master and slave nodes. The incoming data requests are processed by the master node known as *NameNode*. In addition, it manages storing the metadata as well as organizing the storage of files in the slave nodes (*DataNodes*). Hadoop achieves reliability by replicating data across multiple hosts with the default replication value of three. The data is stored in three nodes: two in the same rack (a collection of nodes connected to a same network switch), and one in a different rack. Data nodes can talk to each other to rebalance data, move copies, and keep data replication high [19].

MapReduce-based processing engine MapReduce [39] is a programming model that facilitates performing parallel or distributed algorithms on big datasets on a cluster of computers [2].

The MapReduce model is mainly composed of the following three operations:

- Map: the user-provided Map() code is applied to the local data by each worker. The result is written down to a temporary storage.
- Shuffle: the data is redistributed on the workers such that all data with the same key is located on the same worker.
- Reduce: the user-provided Reduce() code is performed by the worker nodes per key.

A MapReduce program usually has a high communication cost often dominating the computation cost [164, 168]. Therefore, the programmer should consider a good trade off [168] between them. Even though that MapReduce programming model ensures scalability and fault tolerance, it has some weaknesses due to its architecture [99]. In summary the disadvantages are listed as follows:

- Materialization of intermediate results between Map and Reduce function
- Necessity of manual coding even for common operations

- Difficulty of maintenance due to hidden semantics inside the Map and Reduce functions
- Disability of utilizing global state information

Considering the above mentioned deficiencies, more capable and less disk-oriented general purpose techniques and frameworks have been developed. In the following two popular current frameworks are presented and compared with each other in order to remark the improvements.

2.3.2 State of The Art

Current distributed systems perform in-memory computations in order to achieve higher speed and lower latency compared with Hadoop MapReduce engine. They furthermore have the functionality to process streaming workloads. Moreover, they provide an extended set of transformation operators in order to making ease of use. Apache Flink [23] and Apache Spark [185] are two introduced distributed systems that are designed based on in-memory computation. They are not assumed as a replacement for Apache Hadoop rather their processing engine runs on top of the HDFS. In the following, we summarize some fundamental differences of them with Hadoop MapReduce engine according to [23, 99, 185].

Low latency processing It is achieved by avoiding unnecessary read/write operations to the disk. Apache Flink and Apache Spark store datasets in the memory that results in reducing the time consuming factor of writing into and reading from the disc.

Programming model The programming model of Apache Flink and Apache Spark generalizes the concepts of the MapReduce programming model. In addition to Map and Reduce functions, they offer transformations akin to the operations of relational database query languages (Join, CoGroup, Filter, Distinct, etc) and Iterations. Consequently, their programming model is a super set of the MapReduce programming model. Furthermore, the data model is not limited to the key-value pair model. Thus, programming is more convenient and can be done in a much more concise way.

Execution model Similar to the programming model, Apache Flink and Apache Spark employ the concepts of parallel relational database systems in their execution model. They reduce the need to materialize intermediate results on file systems by promoting a pipeline-based processing model. It further makes them capable of doing real-time stream processing. Despite the fixed execution pattern of MapReduce programs, the execution plan of Apache Flink and Apache Spark programs are figured out by an optimizer in a lazy manner (only the necessary transformations are evaluated). The execution plan

can be mapped to a directed acyclic graph (DAG) and is computed such that expensive operations are avoided.

Data streaming Apache Flink and Apache Spark support for both batch and streaming workloads. Therefore, for fast processing of data and when low latency is required, stream processing capability can be exploited.

Useful libraries Apache Flink and Apache Spark are multi-purpose frameworks for data analytics. They facilitates complex data analytics through libraries for machine learning algorithms, graph processing, streaming live structured data, etc.

Although Apache Hadoop and Apache Spark are similar in many ways, they differ in several aspects [57, 104, 170]. There is a substantial difference between the two frameworks in ingesting streams of data. Apache Flink features an operator-based streaming computational model. It uses the streaming model for all workloads including batch processing i.e. batch processing is assumed as an special case of stream processing. In contrast, the computational model of Apache Spark is built upon the micro-batch model. Thus, stream processing is considered as batch processing for chunks of data known as Resilient Distributed Datasets (RDDs). To conclude, Apache Flink would be the right choice when large streams of data need to be processed in real-time. However, Apache Spark provides the user, convenient switching between streaming and batch mode (because both have the same API). Apache Spark furthermore facilitates iterations through explicit caching. In Apache Spark for all iterations the same set of instructions is generated while Apache Flink optimizes iterative processes by generating an unique schedule for each iteration. Apache Flink additionally offers delta iterations in order to optimize the process when only part of data needs to be changed [57]. Li et al. [104] compared the runtimes of several popular algorithms for both Apache Flink and Apache Spark. For the non-iterative WordCount example, Apache Flink is slower than Apache Spark while for executing the iterative algorithm PageRank, Apache Flink shows better performance compared with Spark. In executing SSSP, Apache Flink manages to finish the iterations while Apache Spark fails due to the large number of iterations. Veiga et al. [170] confirm the previous results about WordCount and PageRank algorithms. They prove that for PageRank, Flink acquires execution time up to 3.6x faster than Spark. They additionally show that with maximum cluster size, Spark obtains better results for K-Means, while both frameworks come by similar results for Grep and connected components. Apache Spark however is more mature and provides high level of support due to a large community behind.

2.3.3 Apache Flink

The implementation of the Entity Resolution workflows of this dissertation relies on Apache Flink [23]. As mention in Section 2.3.2 compared to Apache Spark, Apache Flink features faster execution of iterative algorithms as well as the more optimized mechanism for performing iterations. Due to the fact that clustering schemes are mostly iterative algorithms, choosing Apache Flink has been figured out as the rational decision. This section explains the Flink batch APIs as well the Gelly library which has been employed for implementation of the clustering algorithms. In Apache Flink, the batch APIs implement transformations on datasets. Certain sources such as files or local collections can initially create a dataset. After one transformation or a sophisticated assembly of them is applied on the dataset, the result is written down on the distributed files, or to standard output via sink operations. A Flink program can run as an standalone program or embedded in other programs. Moreover, a program can execute on a local JVM or on a cluster of many computers. Table 2.3[5] gives a brief overview of a subset of the available transformations.

Gelly as the graph API of Flink offers a set of utilities for graph transformations and modifications. It further provides a library of graph algorithms and iterative graph processing. A Gelly graph is represented by a dataset of vertices and a dataset of edges. Gelly features a variety of methods for retrieving various graph properties, applying transformations (such as map, filter, join, reverse, union, difference, intersect, etc.) on the vertices and edges, neighborhood aggregation, and graph validation[6].

Additionally Gelly facilitates large-scale iterative graph processing by exploiting efficient iteration operators of Flink. It currently provides vertex-centric, scatter-gather, and gather-sum-apply models[7]. In the following each model is explained briefly.

Vertex-Centric model The model is known as ”think like a vertex” where the computations are expressed from the perspective of vertices. In each step of the computation known as superstep, each vertex executes a user-defined function. The communication between vertices is done through messages [84].

Scatter-Gather model Similar to the vertex-centric iterations, the model expresses the computation from the perspective of a vertex in the graph. In this model, a vertex sends messages to the other vertices in one superstep and updates its value in the next super-

[5] https://ci.apache.org/projects/flink/flink-docs-stable/dev/batch/

[6] https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/libs/gelly/graph_api.html

[7] https://ci.apache.org/projects/flink/flink-docs-release-1.12/dev/libs/gelly/iterative_graph_processing.html

step based on the messages it receives [84].

Gather-Sum-Apply model It is very similar to the scatter-gather model but each superstep consists of three phases of Gather, Sum, and Apply. In Gather phase a partial value is produced by a user-defined function applied on the edges and neighbors of each vertex. Then, in Sum phase, the produced partial values are aggregated to a single value and finally in Apply phase, vertex values are updated by using the current value and the aggregated value of the previous phase [84].

The implementations of the parallel clustering algorithms (Chapter 4 and Chapter 5) mainly relies on Gelly Scatter-Gather iterations. For the rest of Entity Resolution pipeline (explained in Section 2.1.2), the batch APIs of flink are employed.

Table 2.3: Apache Flink dataset transformations

Operation	Description
Map	Takes one element and produces one element.
FlatMap	Takes one element and produces zero, one, or more elements.
Filter	Evaluates a boolean function for each element and retains those for which the function returns true.
Reduce	Combines a group of elements into a single element by repeatedly combining two elements into one.
ReduceGroup	Combines a group of elements into one or more elements.
Distinct	Removes the duplicate entries from the input dataset, with respect to all fields of the elements, or a subset of fields.
Join	Joins two data sets by creating all pairs of elements that are equal on their keys.
OuterJoin	Performs a left, right, or full outer join on two data sets.
CoGroup	Groups each input on one or more fields and then joins the groups.
Cross	Builds the Cartesian product of two input datasets.
Union	Creates the union of two data sets having the same type.
First-n	Returns the n arbitrary elements of a data set.
Range-Partition	Range-partitions a dataset on a given key.

2.4 Quality Measurements

The result of clustering is a set of clusters. Each cluster constitutes entities that are assumed as matches e. g. a cluster of size m contains $\frac{m \cdot (m-1)}{2}$ paired entities. In order to measure the quality of different approaches, we use Precision, Recall and F-Measure in our experiments. Firstly, we count true positives (TP) which are the number of pairs that are correctly determined as matches. False positives (FP) analogously comprises non-matching pairs that are incorrectly identified as matches by the method. The matching pairs that could not be matched by the method are called false negatives (FN). On this basis, Precision is determined by computing the ratio between true positives (TP) and all pairs determined by the methods while recall is the ratio of TP and all existing pairs in the ground truth. Figure 2.3 clarifies the concept of TP, FP, and FN. It illustrates the result of a clustering that constitutes of three clusters representing three real-world entities. In Figure 2.3 entities with the same color represent duplicated entities. The cluster c_0 contains four blue entities, therefore it has six ($\frac{4 \cdot (4-1)}{2}$) true positives. On the other hand, one red entity is incorrectly grouped with the blue entities. Thus, c_0 creates four false positives as well. Similarly, the cluster c_1 has ten true positives while in cluster c_2, one red and one green entity are incorrectly clustered that makes one false positive. The red entities of c_0 and c_2 should have been clustered within c_1 to make a perfect clustering. Grouping them in separated clusters creates eleven false negatives. On this basis, Precision and Recall are computed as follows:

$$Precision = \frac{TP}{TP + FP} \quad (2.1) \qquad Recall = \frac{TP}{TP + FN} \quad (2.2)$$

F-Measure is used as the harmonic mean between Precision and Recall. It is computed as follows:

$$F - Measure = \frac{2 \times Precision \times Recall}{Precision + Recall} \quad (2.3)$$

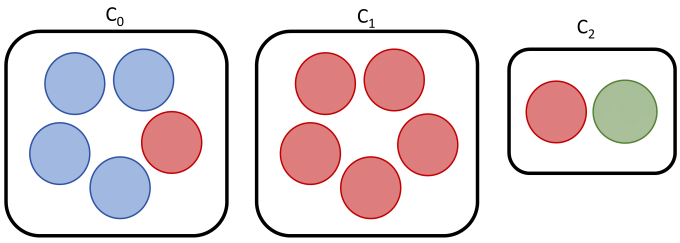


Figure 2.3: Quality measurement example

3 FAMER

3.1 Motivation

FAMER is an open source research framework[1], for scalable data deduplication in multiple sources. The name FAMER stands for **FA**st **M**ulti-source **E**ntity **R**esolution system. As an *Entity Resolution system*, it possesses all components of a general ER pipeline described in Section 2.1.2. The details about each component are explained in Section 3.3 and Section 3.4. The word *FAst* refers to the fact that FAMER is implemented on top of the distributed processing engine of Apache Flink[2]. Finally, the term *Multi-source* emphasizes that FAMER can handle data from multiple (> 2) input data sources. This feature is to serve Big Data requirements. In Big Data applications, data is from many sources that need to be integrated and deduplicated. The number of sources and the quality of them influence the ER process. We investigate these issues in the later chapters (Chapters 4 and 5). Considering additionally the velocity aspect of Big Data, new incoming entities or even data sources need to be handled. In this situation, static approaches are not sufficient to add entities to an in-use Knowledge Graph where the majority of already integrated entities is largely unaffected by new entities and should not have to be re-integrated for every update. Therefore, FAMER supports incremental ER (see Chapter 6) as well as Batch ER.

In Section 3.2, we initially point out the data model and the Graph analytics software Gradoop which FAMER is implemented on top of it. We then explain each component of

[1]FAMER repository https://git.informatik.uni-leipzig.de/dbs/FAMER
[2]Apache Flink website https://flink.apache.org/

FAMER (version 0.1.0) for both batch (Section 3.3) and incremental pipelines (Section 3.4). Finally Section 3.5, introduces the Famer GUI which is developed in a joint work with Mohammad Ali Rostami.

3.2 Data Model and Data Structures

FAMER considers a set of k data sources $S = S_1, \ldots, S_k$, each containing an arbitrary number of entities E such as e_1, e_2. Each entity e_i encompasses a set of property values including the source information that describes the entity. Two entities can be connected by a link with a similarity value sim. The similarity value indicates the degree (probability) that they represent the same real-world object. According to this, candidate pairs are matched by a binary equivalence mapping $M_{i,j} = (e_1, e_2, sim)|e_1 \in S_i, e_2 \in S_j, sim[0, 1]$. If sources are duplicate-free, then $i \neq j$.

To achieve the goal of integrated data sources, FAMER determines clusters of entities that can be denoted as $C = c_1, c_1, \ldots, c_n$. Each c_i constitutes duplicates of the same real-world object and is identified by a unique id named as $cluster\_id$ which is stored as a property value in all cluster members. The number of clusters $|C|$ or the size of them are not predefined. If all sources are duplicate-free, then the maximum possible cluster size equals to the number of sources k, because each cluster can at most keep one entity from each source.

FAMER is implemented using Apache Flink and the extension for graph analytics called Gradoop [80]. Gradoop is an open source research framework[3] for scalable graph analytics built on top of Apache Flink. It offers the Extended Property Graph Model (EPGM) [82] that extends the well known property graph model [151]. In Gradoop the extended property graph is named logical graph. Moreover, the introduced concept of graph collection represents a combination of several logical graphs. Therefore, FAMER stores the attribute values of entities as key value properties. Analogously, the similarity values of matching entity pairs are represented as edge properties.

Gradoop can be easily integrated in a workflow which already uses Flink operators and Flink libraries (e.g. Gelly)[4]. Each Gradoop element such as graph collections, logical graphs, vertices and edges have an unique identifier, one label and a number of key-value properties. Each element can contain an arbitrary number of properties because no fixed schema is involved[5].

[3]Gradoop repository https://github.com/dbs-leipzig/gradoop
[4]Gradoop wiki https://github.com/dbs-leipzig/gradoop/wiki
[5]Gradoop data model
https://github.com/dbs-leipzig/gradoop/wiki/Data-Model

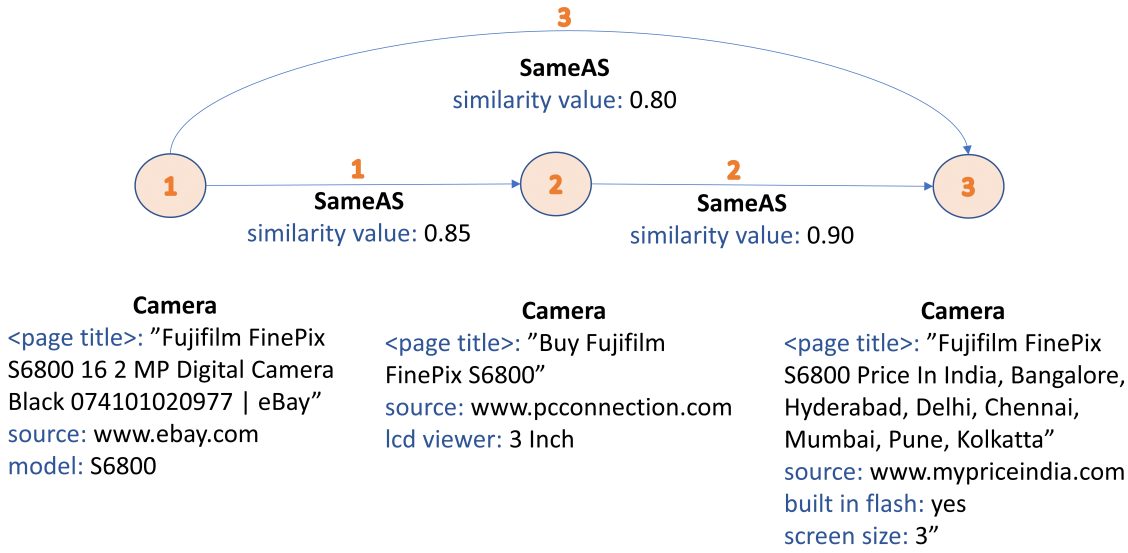


Figure 3.1: An example of FAMER similarity graph implemented using Gradoop logical graph

Figure 3.1 depicts an example of a similarity graph represented as a Gradoop logical graph. The computed similarity value between each pair of vertices is stored as an edge property value while the properties of each entity are represented as vertex property values. Each logical graph is comprised of three *Apache Flink DataSets* for vertices (EPGMVertex), edges (EPGMEdge), and graph heads (EPGMGraphHead).

3.3 FAMER Batch Pipeline

The FAMER framework is depicted in Figure 3.2. The input of FAMER is data from multiple sources and the output is a set of clusters. Each output cluster contains entities that represent a unique real-world entity. FAMER consists of two main modules. The *Linking* module generates a similarity graph which is given to the *Clustering* module as input. The clustering module groups entities into clusters by removing existing false links of the similarity graph and adds missing true matches to the final output. Both *Linking* and *Clustering* modules are configured by a *json* configuration file. The configuration file lists the selected methods for different parts of linking and clustering as well as their input parameters. Listing 3.1 depicts the overview configuration of one complete FAMER job[6]. It constitutes the configuration of *Linking* and *Clustering*. The input is read from the path specified in *Preprocessing* part while the *Postprocessing* identifies writing the output graph on the disk or evaluates the results. One job can perform both

[6]FAMER overall configuration https://git.informatik.uni-leipzig.de/dbs/FAMER/-/wikis/Home/Configuration/Overall-Configuration-(JSON)

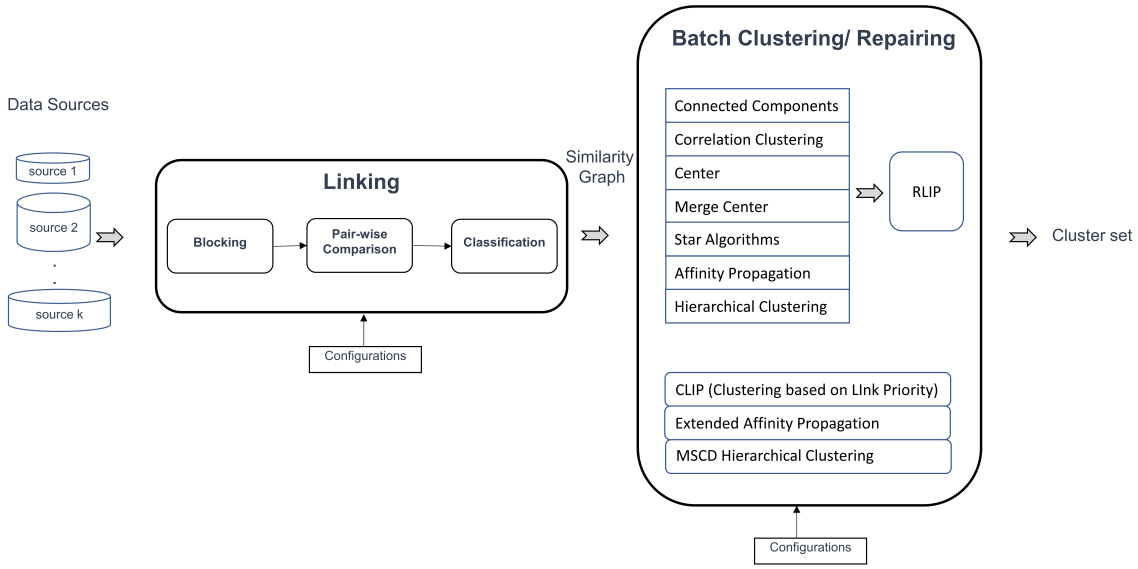


Figure 3.2: FAMER batch workflow

linking and clustering or any of them. Moreover, the configuration file can include an array of jobs (multiple jobs) in order to enable FAMER to execute arbitrary number of consecutive jobs in one run.

Listing 3.1: FAMER batch configuration file

```
[
  [
    {
      "task": "PREPROCESSING",
      "config":{}
    },
    {
      "task": "LINKING",
      "config":{}
    },
    {
      "task":"CLUSTERING",
      "config":{}
    },
    {
      "task":"POSTPROCESSING",
      "config":[]
    },
  ]
]
```

3.3.1 Preprocessing

The preprocessing task aims mainly at reading data from disk or modify an EPGM element. The list of different preprocessing tasks are as follows[7]:

- READ: reads Gradoop graph data as logical graph (LOGICAL_GRAPH) or graph collection (GRAPH_COLLECTION) from a given folder path.
- COMBINE: combines any number of logical graphs into one graph collection.
- BENCHMARK: reads provided benchmark data and returns it as a specified type such as PERFECT_MAPPING, PERFECT_CLUSTERING, GRAPH_COLLECTION, or LOGICAL_GRAPH. Below is the list of benchmark sets[8] and the available return types:
  - ABT_BUY (PERFECT_MAPPING, GRAPH_COLLECTION)
  - AMAZON_GOOGLE (PERFECT_MAPPING, GRAPH_COLLECTION)
  - DBLP_ACM (PERFECT_MAPPING, GRAPH_COLLECTION)
  - DBLP_SCHOLAR (PERFECT_MAPPING, GRAPH_COLLECTION)
  - AFFILIATIONS (PERFECT_MAPPING, GRAPH_COLLECTION)
  - GEOGRAPHIC (PERFECT_MAPPING, GRAPH_COLLECTION, PERFECT_CLUSTERING)
  - NC_VOTERS (PERFECT_MAPPING, GRAPH_COLLECTION)
  - MUSICBRAINZ (PERFECT_MAPPING, GRAPH_COLLECTION)
  - CAMERA (PERFECT_MAPPING, GRAPH_COLLECTION)
- Check a logical graph for the mandatory vertex property ”graphLabel”[9]. It can be applied to the preprocessing READ tasks.
- Transforms vertex properties of the logical graph or the graph collection. The properties of a vertex in a logical graph or a graph collection can be transformed

[7] FAMER preprocessing configuration https://git.informatik.uni-leipzig.de/dbs/FAMER/-/wikis/Home/Configuration/PreProcessing-Configuration-(JSON)

[8] https://dbs.uni-leipzig.de/de/research/projects/object_matching/benchmark_datasets_for_entity_resolution

[9] The ”graphLabel” property specifies the data source of the entity.

to meet special needs for linking or clustering. There are two possibilities, which are optional and can be applied to the READ and COMBINE tasks which return a logical graph or a graph collection:

- Combines arbitrary number of properties into a new property.
- Renames one or more properties.

3.3.2 Linking

The Linking module of FAMER constitutes of three main parts. Initially, entities are grouped into blocks in the *Blocking* part to limit the number of comparisons. Then the matching candidates are compared in the *Pair-wise Matching* part and a similarity value is generated for each pair. Finally, the *Match Classifier* decides about the final matches.

In FAMER wiki[10], FAMER linking configuration is explained comprehensively.

Blocking

FAMER implements the Standard Blocking (SB) and Sorted Neighborhood blocking (SN) methods in a distributed fashion.

Both SB and SN methods can be parallelized using MapReduce framework [39]. Basic SB is implemented by only one MapReduce job. In mapping phase, each *mapper* converts each entity to the key value pair of blocking-key and entity-id. Then, every single *reducer* processes all entities with the same blocking key. In SN implementation, the *mapper* acts like the *mapper* of SB. Then, sorting entities based on their blocking keys is done by the *partitioner* that follows the *mapper*. The *reducer* function, slides down a window of size w on the entities of each *reducer* node. Generating duplicates from windows boundary entities must be additionally implemented [140].

As mentioned in Section 2.1.2, since the sizes of the output blocks can be highly skewed, achieving good load balancing is the major challenge in parallelising a blocking technique. Kolb et al. proposed the load-balanced SB [89] and SN [91] based on MapReduce framework. For SB two different methods named as BlockSplit and PairRange are proposed and implemented. BlockSplit dedicates equal number of entities to each *reducer* by breaking large blocks into smaller sub-blocks while PairRange distributes the comparisons on the reducer nodes evenly. Figure 3.3 shows the sequence of Flink transformations for the PairRange approach. The input of blocking is the DataSet of vertices

[10]FAMER linking configuration https://git.informatik.uni-leipzig.de/dbs/FAMER/-/wikis/Home/Configuration/Linking-Configuration-(JSON)

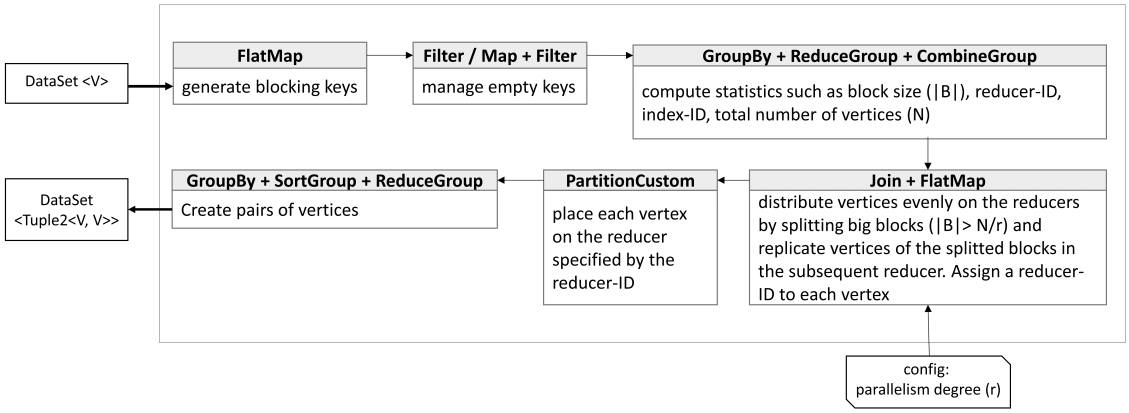


Figure 3.3: Sequence of transformations for the SB

and the output is a DataSet of paired vertices. Initially, blocking keys are generated for each vertex using the FlatMap transformation, because some key generation methods may generate multiple keys for a single entity. When data is dirty and some property values are missing in some entities, then a group of vertices may have empty keys that results in forming a bucket comprised of non-relevant vertices. FAMER manages empty keys by the user preferred strategy of removing the empty block (REMOVE strategy), or attaching all keys to each vertex (ADD_TO_ALL strategy), or leaving the empty block as it is (EMPTY_BLOCK strategy). Then, a set of GroupBy and ReduceGroup transformations are applied to compute the statistics of each block, because distributing comparisons evenly on the reducers requires knowing the size of each block and total number of vertices. In the next step big blocks (blocks bigger than $\frac{total\ number\ of\ vertices}{number\ of\ Flink\ reducers}$) are splitted into multiple blocks. Therefore, the entities of the splitted blocks are replicated in the subsequent blocks as well. A reducer-ID is assigned to each vertex such that vertices are evenly distributed on the reducers. The transformation PartitionCustom relocates the vertices and puts them on the specified reducers. Finally, on each reducer the GroupBy transformation pairs vertices together. The replicated vertices are not paired with each other, because they are getting paired in other reducers.

For load-balanced SN, Kolb et al. [91] proposes JobSN and RepSN methods. The former approach employs a second MapReduce job to generate boundary entities while the latter enumerates the entities to assign evenly-sized entity ranges to every *reducer*. Figure 3.4 shows the sequence of Flink transformations for the RepSN approach. After key generation, a reducer-ID is assigned to each vertex such that vertices are evenly distributed on the Flink reducers. Then, $windowsize - 1$ number of vertices of each reducer are replicated for the next reducer. Finally, the vertices are grouped by the as-

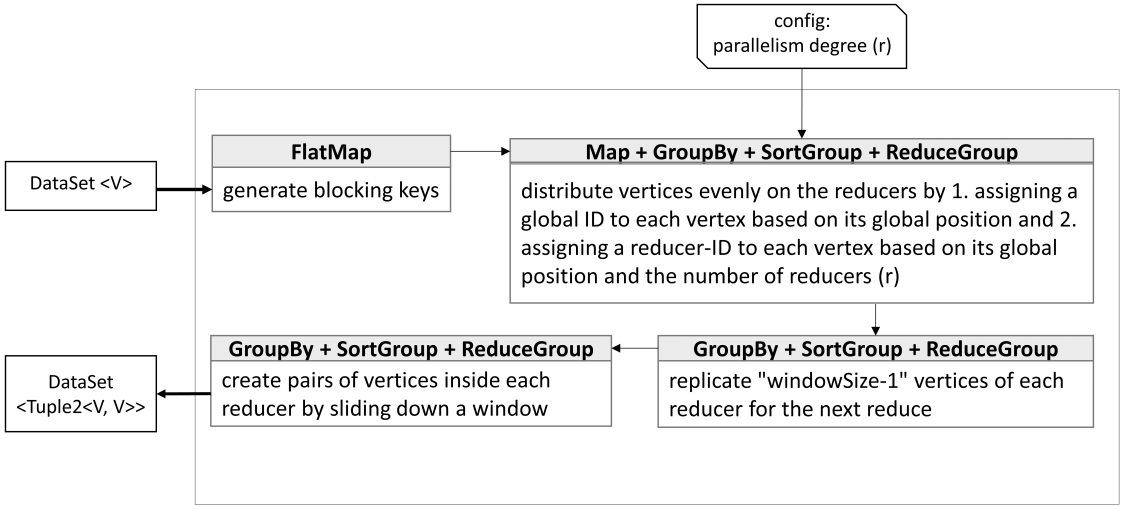


Figure 3.4: Sequence of transformations for SN

signed reducer-ID and are paired together on each reducer by sliding down a window. Similar to BlockSplit, the replicated vertices are bypassed to be paired together.

To improve the recall of blocking, FAMER additionally supports multi-pass blocking which is using more than one blocking phase. Then, the candidate pairs are the union of candidate pairs of the multiple blocking phases. The improved recall is gained by the cost of less efficiency due to performing multiple passes of blocking and removing repetitive candidate pairs.

All blocking methods need to generate at least one blocking key for each entity. Considering each property value as a string, FAMER supports the following key generation methods:

- Full property value: assumes the complete value of the specified property as key.
- Prefix length: assumes the value of initial n characters of the specified property as key.
- QGrams: generates a key using the qGrams method on the attribute value. It splits the attribute value to substrings with the given length q. The keys are generated by concatenating the substrings with different combination orders. The minimum number of substrings is calculated using the threshold. If the attribute value length is less than q, then the whole attribute value is returned as one single key.
- Word tokenizer: splits the attribute value on the given *tokenizer* string into a list of generated keys. If the attribute value does not contain the *tokenizer* then the attribute value itself is returned as a single key.

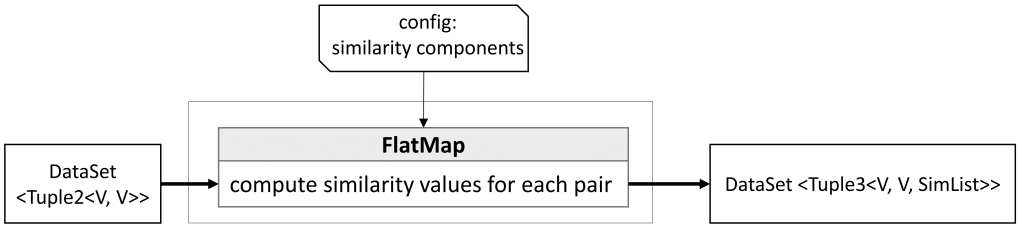


Figure 3.5: The transformation for the pair-wise comparison

The last two methods may generate more than one key per entity. Therefore, the output of blocking may contain repetitive candidate pairs. FAMER removes the repetitive pairs to have a clean output for blocking.

Pair-Wise Comparison

In this part a similarity degree is computed for each candidate pair. Thus, a selected set of corresponding property values of the pair are compared using different similarity methods. Figure 3.5 shows that for each pair of blocked vertices, a list of similarity values are computed. The FAMER configuration enables the user to specify the list of expected similarities between the desired property values (see Appendix A for more details about similarity configurations).

The similarity methods supported by FAMER are listed below:

- Edit distance Levenshtein: The smallest number of edits (character insertions, deletions and substitutions) that are required to convert string s_1 into string s_2 is defined as Levenshtein distance of the two strings. The distance can be converted into a similarity degree in [0, 1] by $1 - \frac{distance}{max(|s_1|,|s_2|)}$ [27].

- Extended Jaccard: Firstly two strings are tokenized using a *tokenizer*. Then, all tokens of string s_1 are compared with all tokens of string s_2 using a secondary similarity value (FAMER uses Jaro Winkler similarity). The set of pairs S are the pairs that their similarity value is bigger than a threshold θ. The unique tokens from s_1 and s_2 that are not included in S as an element of pair are noted by U_1 and U_2 respectively. Then, the similarity degree is computed by $1 - \frac{|S|}{|S|+|U_1|+|U_2|}$ [119].

- Jaro Winkler: Jaro distance counts the number of matching characters (c) as well as the number of transpositions (t) in two comparing strings s_1 and s_2. The Jaro similarity is computed by $\frac{1}{3}(\frac{|c|}{|s_1|} + \frac{|c|}{|s_2|} + \frac{|c-t|}{|c|})$. Then, considering a threshold θ, the Jaro Winkler similarity is computed by $sim_{Jaro} + \theta \cdot l \cdot (1 - sim_{Jaro})$. The length

of common prefix at the beginning of the two strings up to a maximum of four characters is noted by l [27].

- List similarity: It computes the similarity degree of two list of values using the user specified similarity computation method. The computed similarities for each pair of compared values are aggregated into one single similarity degree.

- Longest common substring: The common substrings of the two comparing strings s_1 and s_2 are found and then the total summed length of all found common substrings noted by l_c is computed. The similarity degree can be computed as $sim_{dice} = \frac{2 \cdot l_c}{|s_1|+|s_2|}$, or $sim_{overlap} = \frac{l_c}{min(|s_1|+|s_2|)}$, or $sim_{jaccard} = \frac{l_c}{|s_1|+|s_2|-l_c}$ [54].

- Monge-Elkan: It splits two input strings s_1 and s_2 into tokens using a *tokenizer*. Then, using a secondary similarity computation function (FAMER uses JaroWinkler), the similarity between the set of tokens of s_1 (noted by A) and the set of tokens of s_2 (noted by B) is computed. The maximum similarity between each token of $|A|$ and $|B|$ is found. These maximum similarities are summed up (noted as sum). Then, the aggregated similarities of all tokens is computed by $\frac{l}{|A|} \cdot sum$ [111, 112].

- Numerical similarity with maximum distance/percentage: The difference of the two numbers are computed (noted as $diff$) and if it is less than a predefined *distance*, then the similarity of the numbers is computed as $1 - \frac{diff}{distance}$. The method can be used by a predefined *percentage* instead of a predefined absolute *distance* [27].

- Q-grams: The two input strings s_1 and s_2 are split into small substrings of length q. Then, the number of common substrings of the two strings c_{common} is computed. Assuming the number of all substrings of s_1 and s_2 as c_1 and c_2 respectively, the similarity degree between s_1 and s_2 is computed as $sim_{overlap} = \frac{c_{common}}{min(|c_1|+|c_2|)}$, or $sim_{jaccard} = \frac{c_{common}}{|c_1|+|c_2|-c_{common}}$, or $sim_{dice} = \frac{2 \cdot c_{common}}{|c_1|+|c_2|}$ [97].

- Truncate Begin/End: It considers only the first/last n characters of the two strings. If the considered substrings are equal, then the similarity of the two input strings is 1 and otherwise 0.

- Geographic distance: Using the latitude and longitude of geographic locations, the distance between them can be computed in kilometer. Two locations can be assumed as similar if the distance of them is less than a defined threshold β.

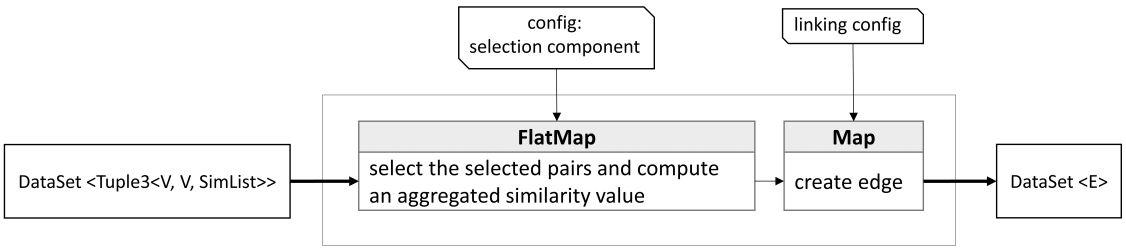


Figure 3.6: Sequence of transformations for the match classification

Match Classification

FAMER offers both threshold-based and rule-based classification methods as well as the combination of them. Figure 3.6 shows that the DataSet of paired vertices and the corresponding list of similarity values are the input of match classification and the output is a DataSet of created edges. Initially, the similarity values of each pair are evaluated against the selection rules specified in the selection configuration (see Appendix A for more details about selection configurations) and then an edge is created in accordance with each selected pair.

FAMER benefits from ML-based classification approaches such as decision trees, methods from the Weka library (J48, IBk, RandomForest), Flink implementation of Random Forest and word embedding training with JFastText which are not a part of this dissertation.

3.3.3 Clustering

If the candidate matching pairs are classified individually, it may lead to a long chain of entities that are related to each other through transitive closures. Thus, two entities with a very low or even no similarity are considered as matches. Moreover, the special restrictions of some input data sources such as one-to-one mapping or source-consistency constraints can not be satisfied. Therefore, clustering is applied on the similarity graph

Table 3.1: ER clustering algorithms classification

2 clean sources	Single source	Multiple (>2) clean sources	Multiple (>2) combined sources (clean & dirty)
Max-Both, Hungarian Algorithm, Stable Marriage, ...	Correlation clustering, Center, Merge Center, Star, Hierarchical Agglomerative Clustering (HAC), Affinity Propagation (AP), ...	SplitMerge, CLIP, ...	Extended AP, MSCD HAC, ...

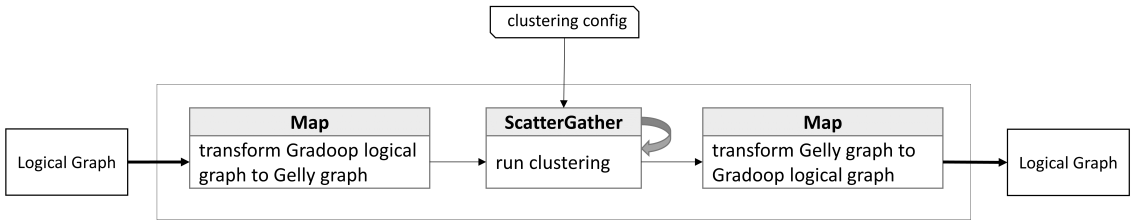


Figure 3.7: Sequence of transformations for clustering approaches implemented with Gelly

as the last step of the ER pipeline. Clustering aims at removing false edges as well as cutting transitive closure chains such that only true missing edges are added.

Table 3.1 classifies the clustering algorithms that have already been employed in ER applications. For two clean sources only one-to-one mapping is allowed. Therefore, only algorithms that generate clusters with maximum size of two (each cluster contains at most one entity from each source) are applicable. In contrast, generic-purpose algorithms are leveraged for deduplicating in a single source [68]. The contribution of this dissertation is on clustering for multiple (> 2) clean sources as well as multiple combined clean and dirty sources. FAMER however includes the single source clustering methods implemented in both sequential and parallel fashion. The clustering algorithms for two clean sources and single source deduplication listed in Table 3.1 are explained in Section 2.2.1. Furthermore, Chapter 4 and Chapter 5 focus on the clustering algorithms for multi-source ER.

For implementing a majority of clustering algorithms we use Gelly library which is the graph processing engine of Apache Flink. In particular, we employ Gelly graphs that are comprised of a DataSet of vertices and a DataSet of edges. Listing 3.2 depicts the Gelly graph class. Each vertex the same as each edge is comprised of a key K and a value. As shown in Figure 3.7 for exploiting the Gelly ScatterGather iterations, FAMER initially transforms the Gradoop logical graph to a Gelly graph. Then, the clustering algorithm specified in clustering configuration (see Appendix A for more details about clustering configurations) is executed through a number of ScatterGather iterations and finally the resulted Gelly graph is again transformed to a Gradoop logical graph.

Listing 3.2: Gelly graph

```
class Graph<K, VV, EV> graph {
        DataSet<Vertex<K, VV>> vertices;
        DataSet<Edge<K, EV>> edges;
}
```

3.3.4 Postprocessing

The postprocessing task mainly aims at writing data to the disk or computing the quality of output results. The list of different preprocessing tasks are as follows:[11]

- WRITE_GRAPH: Writes Gradoop graph data to disk.
- QUALITY: Evaluates the quality of a similarity computation or a clustering and writes the results as a comma-separated file to a given file path.

A postprocessing configuration can include an array of tasks e.g. it is possible to store a similarity graph as well as calculating and storing its quality measures in the same postprocessing job.

3.4 FAMER Incremental Pipeline

The incremental FAMER workflow is indicated in Figure 3.8. The input of the workflow is a stream of new entities from existing sources or from a new source plus the already determined clustered similarity graph from previous iterations. The *Linking* part here focuses on the new entities and does not re-link among previous entities. We also support the linking among new entities to provide additional links in the similarity graph that may lead to better cluster results. The output of the linking is a *grouped similarity graph* composed of existing clusters and the group of new entities and the newly created links (the light-blue colored group in the middle of Figure 3.8).

The *Incremental Clustering/Repairing* part supports two methods for integrating the group of new entities into clusters. In the base (non-repairing) approach the new entities are either added to a similar existing cluster or they form a new cluster. The repairing approach however is able to repair existing clusters to achieve a better cluster assignment for new entities. The details of the incremental clustering approaches are explained in Chapter 6.

Listing 3.3 depicts the configuration overview of one complete job including one incremental repairing task. The configuration file can include multiple jobs as well. It should be noted that FAMER can not be initiated with an incremental task. It is important that at list the first task is a normal batch ER that produces a clustered similarity graph which can be given into the input of the framework again. The preprocessing task reads the clustered similarity graph and at least one more raw graph (not linked) as input. In the

[11] https://git.informatik.uni-leipzig.de/dbs/FAMER/-/wikis/Home/Configuration/PostProcessing-Configuration-(JSON)

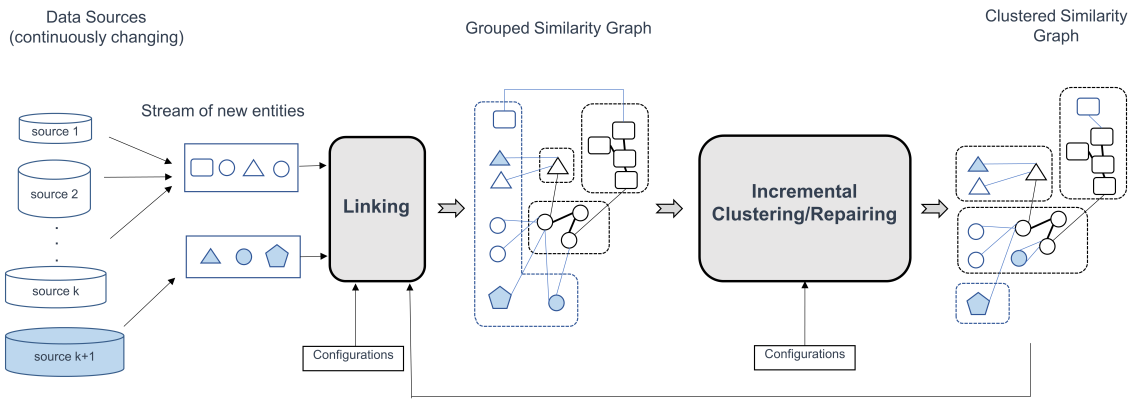


Figure 3.8: FAMER incremental workflow

Linking part the new entities are linked to the old ones and optionally with each other. Then the *incremental repairing* task applies one of the repairing methods specified by user to integrate the newly linked entities with the rest of similarity graph. Finally, the *Postprocessing* task writes the output down on HDFS.

Listing 3.3: FAMER incremental configuration file

```
[
  [
    {
      "task": "PREPROCESSING",
      "config":{}
    },
    {
      "task": "LINKING",
      "config":{}
    },
    {
      "task":"INCREMENTAL_REPAIRING",
      "config":{}
    },
    {
      "task":"POSTPROCESSING",
      "config":[]
    },
  ]
]
```

3.5 Visualization Tool

During the continuous development of FAMER, it was challenging to investigate the correctness and efficiency of the certain algorithms and to understand the problems. Such an investigation may lead to introduce improved matching and clustering algorithms or even an extra postprocessing step of repair. However, to identify (or debug) any occurring issues with limited effort and time we see the need for a comprehensive and powerful approach to visually analyze similarity graphs and ER clusterings. Unfortunately, general purpose graph visualization tools like Gephi[12] or Graphviz[13] have limited capabilities to analyze ER clusterings. They additionally are not capable of proper visualization of large (Big Data) similarity graphs with vertex and edge properties.

In this section we introduce SIMG-VIZ, the visualization system for entity resolution and clustering that allows us to investigate different match and clustering techniques for multi-source entity resolution. SIMG-VIZ offers the following key features:

- SIMG-VIZ offers analyzing precomputed similarity graphs and clusterings from existing ER tools. It additionally supports executing and analyzing ER match tasks directly with FAMER.
- Different graph and ER cluster visualization techniques and layouts can be applied to choose the best visualizations.
- To increase performance, some layouts can be precomputed on a server with either parallel or serial computation. This provides a significant optimization potential in particular for force-directed layouts [11].
- To support visualization of large graphs, preprocessing techniques such as sampling (also executed in parallel on the server) can be selected to obtain a fast overview of large similarity graphs and their clustering results.
- Clusters and their overlaps as well as edges annotated with their type and similarity are visualized by using a simple but useful cake-like visual metaphor. Users can interact with clusters and select individual clusters for investigation.

[12] https://gephi.org
[13] http://www.graphviz.org

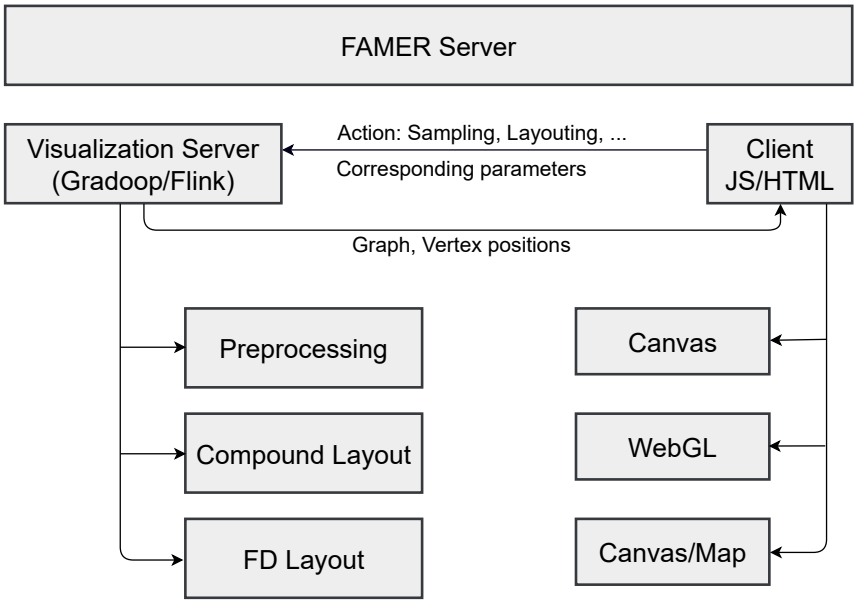


Figure 3.9: SIMG-VIZ architecture

3.5.1 Overview

The SIMG-VIZ system consists of three modules: (1) the FAMER server, (2) a visualization server in JAVA and (3) a web-based GUI-client written in JavaScript (see Figure 3.9).

The FAMER server is used to link several sources and executes defined matching tasks. However, SIMG-VIZ allows the user to load similarity graphs and clustering results computed by other tools as well. The visualization server offers several preprocessing (e.g. for sampling) and layouting algorithms. The preprocessing algorithms are implemented in distributed fashion on top of Gradoop and Apache Flink whereas graph layoutings are currently only implemented as non-distributed algorithms. The web-based client, provides an interactive visualization of similarity graphs and ER-clusterings. SIMG-VIZ offers investigating vertex and edge properties as well as triggering server-side components like matching, clustering, preprocessing and layouting. The user is able to trigger server based REST-interfaces through a client-side GUI. The visualization server and the FAMER server both respond with JSON results. In the following sections each component is described in detail.

3.5.2 Client (Web-based HTML/JS Frontend)

Figure 3.10 indicates an overview of the web-based client of SIMG-VIZ. The top part of the GUI shows the options that enable the user to choose between different clustering and match configurations. Moreover, the user can select layouting options, i.e. which

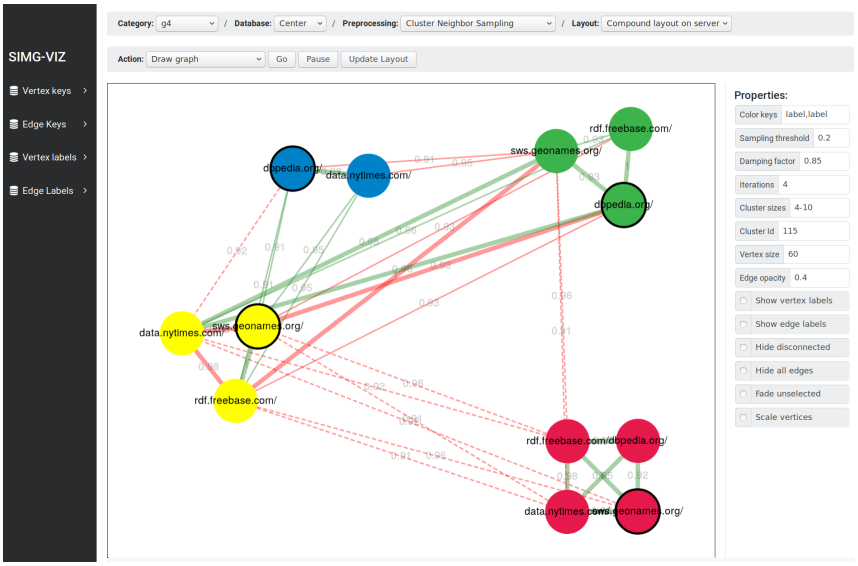


Figure 3.10: An overview of SIMG-VIZ

layout to use and where the computation should take place. Computing the layout on the server gives significant improvements for large graphs. Additionally, the visualization parameters can be set and tuned via the GUI (the right side in Figure 3.10). SIMG-VIZ offers a couple of preprocessing algorithms such as sampling that can be further applied on the similarity graph before visualization. To improve interactivity, additionally styling tasks such as changing vertex or edge sizes are performed on the client. Table 3.2 lists the drawing task types along with the statistics computations and other operations supported by SIMG-VIZ.

Table 3.2: Actions in SIMG-VIZ

Action	Description
Draw graph (Cytoscape)	Draws a sim-graph in Cytoscape[14].
Draw graph (WebGL)	Draws a sim-graph in WebGL based on VivaGraph.
Compute only	Executes preprocessing and sampling without visualization.
Compute labels/keys	Computes all labels and property-keys of the vertices and edges for filtering in the left part of the UI.
Save as image	Exports an image of the drawn graph.
Remove selected node	Removes a selected node.
Degree Distribution	Computes the degree distribution of the graph.
Graph Statistics	Computes additional basic statistics of a graph.

Table 3.3: Preprocessing algorithms in SIMG-VIZ

Preprocessing	Description
Graph sampling	Computes a statistical sampling of a graph. Currently SIMG-VIZ implements vertex, edge and page rank sampling.
Graph summary	Computes a graph summary by grouping dense subgraphs of a graph to generate a compact overview of a large graph [83]. In SIMG-VIZ the Flink implementation is used.
Cluster neighbor filtering	In particular for ER-Clustering a filtering to neighbors of clusters is needed. The user specifies a cluster ID (at the right part of UI) and that cluster together with its neighbor clusters is visualized.
Cluster sizes filtering	Only the clusters with specific sizes are visualized.
Cluster Aggregation	It visualizes a graph in which the cluster vertices are grouped together.

3.5.3 Visualization Server

The visualization server offers preprocessing and layouting services. The preprocessing algorithms are implemented in a distributed fashion on top of Gradoop and Apache Flink. Table 3.3 lists currently implemented preprocessing components.

All layouting algorithms are available both for the client and the server as non distributed algorithms. We observed that executing layouting algorithms that need iterations like the force directed layout [55] should not be executed on the client within a browser. Executing it on a server brings significant runtime improvements, even without distributing the computation to multiple processing nodes. When the layout computation is done on the server, the positions of the vertices along with the graph are sent to the client. Implementing parallel versions of layouting to be run on top of Apache Flink or Gradoop are left as future work.

3.5.4 Cluster Visualization

In this section, we describe some specific features of SIMG-VIZ which are designed particularly for the visualization of ER clusters.

These features are explained along a real world ER-example of integrating four duplicate-free data sources namely Freebase (Fb), New York Times (nyt), DBpedia (db), and Geon-

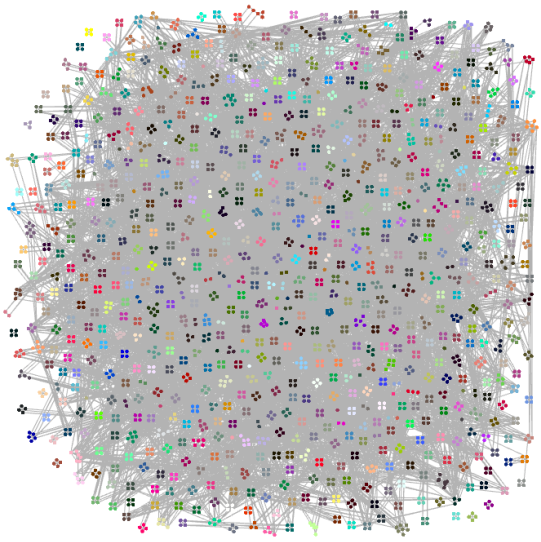
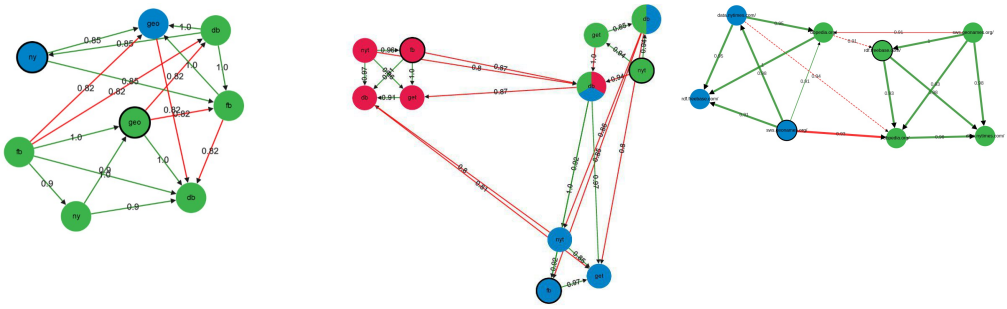


Figure 3.11: A visualization of all clusters

ames (geo). We initially compute a similarity graph and apply different clustering techniques with FAMER. A result cluster includes only vertices from these four sources which are most probably the same real world entities. An edge connects two vertices which have a high value of computed similarity measure. In Figure 3.11 we initially visualize the complete clustering result. Clusters are shown with different colors to indicates cluster membership. Users can identify clusters that may warrant a closer inspection, e.g., clusters with more than four vertices or singleton clusters. It is possible to zoom in and inspect the properties of each vertex. Since generic graph layouting algorithms often have problems in visualizing large similarity graphs (e.g., problem of edge cluttering) we applied a compound layout for cluster visualization. Such compound layouts of graphs like CoSE-Bilkent [42] visualize vertices in a cluster (referred to as compound in the paper) close to each other while the whole graph is visualized by using a modified force-directed algorithm. Figure 3.12b shows a visualization of such a compound layout that is computed on the server. Vertices of a cluster share the same color and are closely grouped together to form a compound.

To get a cleaner picture, a user can interactively select a specific cluster or enter a cluster ID to only visualize a specific cluster for closer inspection. Often we also need to visualize a cluster together with its neighboring clusters (see Figure 3.12a). The vertex labels here refer to the corresponding data source for that entity. The scenario illustrates a problem case since there are more than four cluster members with some data sources having two entities in the same cluster which should not be possible for duplicate-free sources. There is also a singleton cluster that might have to be merged with another



(a) Each cluster is indicated with a unique color

(b) Multicolored vertices belong to more than one cluster

(c) The strength degree of edges are visualized by different colors and styles

Figure 3.12: SIMG-VIZ output

cluster. Based on such observations we are now able to re-assess the used clustering algorithms and investigate new approaches for cluster repair.

We also provide support for visualizing clustering result of specific clustering algorithms like Star[68]. The Star clustering computes cluster representatives and all neighbors of those representatives are assigned to the corresponding cluster. In SIMG-VIZ, those cluster representatives are highlighted with a black outline (see Section 3.5.4). It happens that a vertex is a neighbor of several cluster representatives so that such vertex will belong to multiple clusters. These multi-assignments are represented as pie charts on nodes. Each piece of a pie chart which has a specific color specifies a cluster assignment. For example, Figure 3.12b contains a pie chart with three pieces which means that node (entity) is assigned to three clusters. Obviously, some cluster postprocessing is needed to select the best cluster for each entity that has been assigned to several clusters.

SIMG-VIZ provides special visualization support for evaluating clusters when the perfect cluster result is available for comparison. As shown in Figure 3.12c there are different edge colors: green for edges that link two perfect match entities and red for (wrong) edges based on the ground truth. Hence, clusters containing red edges should be investigated more closely. Finally, we implemented a map-visualization of geo-referenced data so that entities can be plotted onto a map. With the help of this map-visualization we are able to identify false matches within a given dataset.

3.5.5 Web-based Visualization Libraries

Drawing large graphs within a browser is problematic. We investigated several different Javascript-based visualization libraries and observed that there are significant performance differences. Three groups of libraries can be found: (1) SVG-based libraries

compute SVG-nodes and tags. They are often feature rich but do not scale well due to many generated SVG-Elements. (2) The second group relies on HTML-Canvas. These libraries are typically faster but interactivity is harder to realize. Still they mostly do not scale well for very large graphs. (3) WebGL-based libraries offer the best scalability but are still not as feature rich as existing libraries in the other two categories. We finally decided to use the Canvas-based Cytoscape[15] library for small graphs up to 2000 vertices, because it gives more flexibility regarding the style of edges and vertices. For example, the feature of drawing a pie chart on vertices is already available in Cytoscape. For large graphs, we use VivaGraph[16], which is a WebGL-based library with less support for vertex and edge attributes, styling and coloring. However, for very large graphs the user anyway would not be able to see those details.

[15] http://js.cytoscape.org
[16] https://github.com/anvaka/VivaGraphJS

4

Multi-source Clean Clustering

This chapter is based on [131, 157, 158, 160]. In [158] a comparative evaluation of different existing clustering algorithms for ER is provided. Additionally, in [160] new clustering and repairing schemes are proposed. The clustering algorithm named as CLIP intends to create source-consistent clusters and the repairing scheme (RLIP) modifies the output of other clustering algorithms so that overlapping or source-inconsistent clusters are resolved. CLIP and RLIP were presented at the ESWC 2018. All approaches are integrated in FAMER and provided as open source[1]. Additionally, another novel clustering scheme for multiple clean sources is compared with the existing approaches of FAMER [157]. Later, FAMER was applied for the KDD DI2KG data matching challenge 2019 [131] and succeeded to win the competition.

4.1 Motivation

While entity resolution and the corresponding problems such as link discovery are intensely investigated research topics, this problem is still not sufficiently solved for the large-scale integration of data from many sources as needed for knowledge graphs. In our research, we aim at scalable ER approaches for Big Data that are able to deal with large data volumes and multiple data sources. We therefore need ER approaches that support clustering matching entities.

[1] https://git.informatik.uni-leipzig.de/dbs/FAMER/-/tree/master/famer-clustering

Evidently for more than two sources a binary linking of entities is not sufficient but all matches of the same entity should be clustered together to derive a fused entity representation in the knowledge graph. There are several known approaches for such an entity clustering [68] that FAMER integrated parallel implementations of them. In this chapter six clustering schemes, including connected components as the baseline, Correlation clustering [7], two variations of Star clustering [4], Center [69] and Merge Center clustering[69] are introduced.

As explained in Chapter 3, clustering is applied on a *similarity graph* where entities are represented as vertices and edges link pairs of entities with a similarity above a predefined threshold. The clustering schemes use this graph to determine groups of matching entities aiming at maximizing the similarity between entities within a cluster and minimizing the similarity between entities of different clusters. In fact clustering tries to find additional links by considering indirect matches and to eliminate weaker links in favor of more plausible ones. This chapter focuses on duplicate-free (clean) sources. For clean data sources without duplicates[2] each cluster should contain at most one entity per source. As mentioned in Chapter 2 (Section 2.2), we call clusters violating this restriction *source-inconsistent*.

Analyzing the clusters determined by the different clustering schemes, we observed some common problems in particular overlapping clusters and source-inconsistent clusters. Algorithms like Star clustering can associate entities to more than one cluster leading to cluster overlaps and thus wrong clusters. Moreover, while the similarity graphs determined by FAMER never link entities from the same clean source, the transitive clustering of linked entities, e.g., with the baseline approach connected components, can easily lead to source-inconsistent clusters which should be avoided or repaired. Therefore, we proposed and evaluated new algorithms to create high-quality entity clusters or to repair clusters determined by other approaches so that the observed cluster problems are avoided.

In addition to illustrating the problems and solutions mentioned above, this chapter outlines the use of the tool FAMER for the DI2KG 2019 challenge[3]. FAMER could reasonably well solve the entity resolution task of the challenging dataset and be the challenge winner. The challenge task additionally entailed schema matching which was solved by Daniel Obraczka.

Specifically, this chapter provides the following contributions:

- Integration of six previously existing clustering algorithms.

[2] If necessary, the individual sources could be deduplicated before the entity resolution with other sources (see Section 4.5).

[3] http://di2kg.inf.uniroma3.it/2019/#challenge

- The proposal of a new clustering approach called CLIP (Clustering based on LInk Priority) to determine high quality, overlap-free and source-consistent entity clusters. Its cluster decisions are based on a link prioritization considering not only link similarities but also the so-called link strength and link degree.
- The proposal of an approach called RLIP (cluster Repair based on LInk Priority) to repair entity clusters such that the overlap and source inconsistency problems are resolved. It includes a component to resolve overlapping clusters and uses CLIP to produce source-consistent clusters.
- Comprehensive evaluation of the cluster quality and scalability of the former approaches as well as the new approaches for different datasets. Moreover, The new approaches are compared with another multi-source clean clustering algorithm SplitMerge [120].
- Application of FAMER for the schema and entity matching tasks of the DI2KG 2019 challenge [131].

Section 4.2 describes the considered clustering algorithms and their distributed implementation. In Section 4.3 we define concepts and describe the new algorithms which are evaluated in Section 4.4. In Section 4.5, we describe FAMER solutions for the DI2KG 2019 challenge and we report the results. Finally, we briefly discuss related work in Section 4.6 and conclude in Section 4.7.

4.2 Generic Clustering Schemes

In this section, we present the considered generic clustering approaches for entity resolution and their parallel implementation. The parallel implementations are based on the vertex-centric programming model to iteratively execute a user-defined program in parallel over all vertices of a graph. In particular, we use the two-step Scatter-Gather model of Gelly (see Section 2.2.1). Section 2.2.1 gave a brief explanation of the following algorithms. In this section we focus on the vertex-centric implementation of them. The implementation for one of the clustering schemes (Center) is explained in detail; the other implementations follow similar approaches.

4.2.1 Connected components

The subgraphs of a graph that are not connected to each other are called connected components. Having the input similarity graph, the connected components are easy

to determine in a vertex-centric way by letting every vertex iteratively add all its direct neighbors to its cluster. The approach is therefore easy to implement with Scatter-Gather (as shown in [81]). In the evaluation, we use this approach as a baseline for the comparison with the other clustering schemes.

4.2.2 Center clustering

In contrast to connected components, the Center clustering algorithm [69] utilizes the similarity values (weights) of the edges in the similarity graph (see Section 2.2.1).

We proposed and implemented a parallel version of the Center algorithm (see Algorithm 1). In each round of the algorithm for all unassigned vertices, the outgoing edge with the highest weight must be found. The vertices on both sides of this edge are then processed. If one of them is a cluster center, the other will belong to the cluster of that vertex (line 6-line 8). In case one of them is assigned to another cluster (line 9), i.e, both vertices belong to different clusters, the edge between these two vertices is removed

Algorithm 1: Parallel Center

```
   Data: G=(V, E)
1  assignVertexPriorities (V)
   /* priority according to a random permutation of
      vertices */
2  Center ← {}
3  for v_i ∈ V in Parallel do
4      repeat
5          v_nn ← argmax_j (e_(v_i,v_j))
6          if (v_nn ∈ Center) then
7              v_i.setClusterId (nn)
8              V ← V−{v_i}
9          else if (v_nn ∉ V) then
10             E ← E −{e_(v_i,v_nn)}
11         else
12             v_k ← argmax_j (e_(v_nn,v_j))
13             if ((i = k ∧ i > nn) ∨ (v_nn = Null)) then
14                 Center ← Center ∪ {v_i}
15                 v_i.setClusterId (i)
16                 V ← V−{v_i}
17     until (v_i ∈ V)
```

(line 10). When both vertices are unassigned and the edge between them is for both the outgoing edge with the highest weight (line 13, $i = k$), then one of them is assumed as center (line 14) and the other will belong to the same cluster in the next round. For selecting the center in this case we make use of initially assigned (line 1) vertex priorities as done in the sequential algorithm. Hence, the vertex with higher priority is considered as center (line 16, $i > nn$). If a vertex is not connected to any other vertex (line line 13, $v_{nn} = Null$), it is a singleton. The algorithm iterates until all vertices are assigned to a cluster (line line 17).

We implemented parallel Center using the Scatter-Gather model (see Algorithm 2). The algorithm applies two phases that are iteratively executed for all vertices. Phase one (Scatter1, Gather1) finds the nearest neighbour vertex for each vertex v_i (the neighboring vertex with the currently highest edge weight), and phase two (Scatter2, Gather2) processes the status of the found vertex and assigns v_i to an existing cluster or considers it as a center. Again, we initially assign a priority per vertex (line 2). In phase one, for each vertex v_i the neighbor with the K-highest edge weight (nearest neighbor NN) is found (line 14-line 18). K is a helper variable that prevents choosing already assigned

Algorithm 2: Scatter-Gather Center

```
   Data: G=(V ,E)
1  Algorithm Center
2      assignVertexPriorities (V)
3      for (v_i ∈ V) do
4          v_i.K ← 1
5      end
6      repeat
7          Phase1: Scatter1 (Vertex)
8                  Gather1 (Vertex, MessageIterator)
9          Phase2: Scatter2 (Vertex)
10                 Gather2 (Vertex, MessageIterator)
11     until (V≠ {})
12 end
13 Procedure Scatter1 (Vertex v)
14     for (e ∈ getOutEdges()) do
15         msg.Src ← v.getId()
16         msg.Weight ← e.getWeight()
17         sendMessageTo(edge.target(), msg)
18     end
19 end
20 Procedure Gather1 (Vertex v, MessageIterator messages)
21     Array ← messages.Sort()
       /* Messages are sorted based on their weights
          descendingly */
22     v.NN ← Array[v.K].getSrc()
23 end
24 Procedure Scatter2 (Vertex v)
25     msg.Src ← v.getId()
26     msg.NN ← v.getNN()
27     msg.Priority ← v.getPriority()
28     for (e ∈ getOutEdges()) do
29         msg.Weight ← e.getWeight()
30         sendMessageTo(edge.target(), msg)
31     end
32 end
33 Procedure Gather2 (Vertex v, MessageIterator messages)
34     Array ← messages.Sort()
       /* sorted based on weights descendingly */
35     for (i : v.K → Array.Size()) do
36         m ← Array[i]
37         if (m.getSrc().isCenter()) then
38             v.ClustereId ← m.getId()
39             v.assigned ← true
40             break
41         end
42         else if (m.getSrc().isAssigned()) then
43             v.K + +
44         end
45         else if (v.NN= Null ∨ (v.NN = m.getSrc() ∧ v.Priority
             > m.getPriority()) ) then
46             v.ClustereId ← m.getSrc()
47             v.center ← true
48             v.assigned ← true
49             break
50         end
51     end
52 end
```

vertices as neighbor. It is attached to each vertex and initialized with 1 (line 3-line 5). It will be incremented in phase two when a vertex neighbor has been assigned to a cluster (line 42-line 44). In phase two, all neighbors of a vertex v_i are sorted and processed in descending order of the edge weights (for the edge to v_i) (line 34-line 41). Then, vertex v_i is set as center similar to Algorithm 1 (line 45-line 50).

4.2.3 Merge Center

The Merge Center clustering algorithm [69] is a modified version of Center. In contrast to Center, it merges two clusters if a vertex in one cluster is similar to the center of another cluster (see Section 2.2.1). Our parallel implementation for Merge Center is very similar to parallel Center but applies an extra iteration for merging clusters. This iteration is initiated right after all vertices are assigned to a cluster. The merge processing is repeated until there are no further cluster changes.

4.2.4 Star clustering

The Star clustering algorithm [4] initially computes the degree for each vertex of the similarity graph. Then, in each iteration, the unassigned vertex with the highest degree becomes center and all its direct neighbors are assigned to its cluster. The algorithm terminates when all vertices are assigned to a cluster. In contrast to all other clustering approaches, Star clustering can result in overlapping clusters (see Section 2.2.1). As a consequence, it introduces the need of a post-processing to select the best cluster for entities that have been assigned to several clusters.

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

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

Algorithm 3: Parallel Star

```
Data: G=(V, E)
1  V ← {v_1, ..., v_n}
   /* A random permutation of vertices */
2  Center ← {}
3  repeat
4      for (v_i ∈ V) in Parallel do
5          v_max ← argmax_{v_j ∈ {v_j | e(v_i,v_j) ∈ E} ∪ {v_i}} (computeDegree(v_j)))
6          if (v_i = v_max) then
7              Center ← Center ∪ {v_i}
8      for (v_i ∈ V) in Parallel do
9          for (e(v_i, v_j) ∈ E) do
10             if (v_j ∈ Center) then
11                 v_i.addClusterId(v_j.getId())
12                 V ← V − {v_i}
13 until (V ≠ {})
```

4.2.5 Correlation Clustering

As explained in Section 2.2.1, correlation clustering aims at finding a clustering that either maximizes agreements (sum of positive edge weights within a cluster plus the absolute value of the sum of negative edge weights between clusters) or minimizes disagreements (absolute value of the sum of negative edge weights within a cluster plus the sum of positive edge weights across clusters). Gionis et al. [61] propose an approximate and iterative solution for this optimization problem that randomly selects an unassigned vertex as a cluster center in each round. Then, all unassigned neighbors of the selected center are added to the cluster and marked as assigned. The algorithm terminates when there is no unassigned vertex left.

This simple algorithm suffers from too many rounds making it unsuitable for very large graphs. Some studies therefore proposed parallel solutions [26, 135] that select multiple centers in each round. They also address the newly introduced concurrency problem to avoid that a vertex is assigned to more than one center at a time. We implemented the parallel pivot approach of [26], called CCPivot, since it fits well the Scatter-Gather paradigm. In each round of this algorithm, several vertices are considered as active nodes, i.e., as candidates for becoming a cluster center (or pivot). In the next step, active nodes that are adjacent to each other are removed from the set of active nodes; the remaining vertices become centers. Then, adjacent vertices of each center are assigned

to that center and form a cluster. If one vertex is adjacent of more than one center at the same time, it will belong to the one with higher priority. As in the other algorithms, the vertex priorities are determined in a preprocessing phase.

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

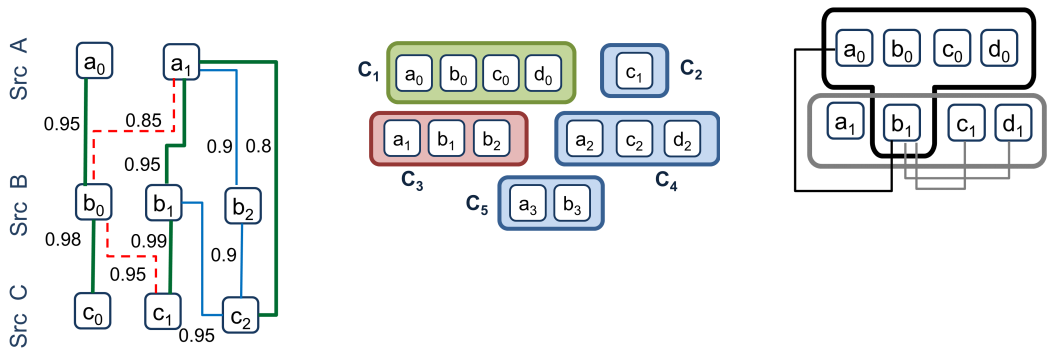
4.3 Clean Clustering Algorithms

This section introduces the novel approaches for clustering multi-source clean clustering. We first define the main concepts and then describe the CLIP and RLIP algorithms.

4.3.1 Concepts

Maximum link: An entity from a *source A* may have several links to entities of a *source B*. From these links, the one with the highest similarity value is called maximum link. For example, for entity a_1 in Figure 4.1a the maximum link with respect to source B is the one with similarity 0.95 to entity b_1. Based on this concept we define the strength of links and classify links into *strong*, *normal*, and *weak* links. Considering a link ℓ between entity e_i from source A and entity e_j from another source B we define these link types as follows:

Strong link: Link ℓ is classified as a *strong* link, if it is the maximum link from both sides, i.e. for e_i to source B and for e_j to source A. In Figure 4.1a, entity a_1 from source A has a strong link, colored in green, to b_1 in source B. Note that an entity can have



(a) Link features

(b) Complete cluster

(c) Cluster association degree

Figure 4.1: Clean clustering concepts

several strong links to different sources; e.g., a_1 is also strongly linked to c_2 from source C.

Normal link: Link ℓ is called a *normal* link, if it is the maximum link for only one of the two sides. In Figure 4.1a, the link between a_1 and b_2 is a normal link (colored in blue) as it is the maximum link from b_2 to source *A*, but not the maximum link from a_1 to source *B*.

Weak link: Link ℓ is a *weak* link, if it is not the maximum link for any of the two sides. In Figure 4.1a, the link between a_1 and b_0 is such a weak link and shown with a red dashed line.

Link degree: The link degree is the minimum vertex degree of its two end point vertices. In Figure 4.1a, the vertex degree of a_1 is 4 and the vertex degree of b_1 is 3, so that the link degree between a_1 and b_1 is $min(4, 3) = 3$.

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

Complete cluster: A source-consistent cluster that contains entities from all sources is called a *complete* cluster. The green-colored cluster in Figure 4.1b is a complete cluster for four sources *A*, *B*, *C*, and *D* as it contains one entity from each source.

Cluster association degree: An entity e that is shared between two or more clusters will be in some cases assigned to the cluster with the highest association degree. The association degree of e for cluster C of size k corresponds to the average similarity of e to the $k-1$ other entities e_i in C, i.e., it is determined by the ratio of the sum of similarity values of the intra-cluster links involving e and $k-1$. In Fig. 4.1c, entity b_1 is member of the gray and black clusters of sizes 4 and 5, respectively. Assuming a link similarity of 1 for the shown links, the association degree for b_1 is $2/3$ for the gray cluster and $1/4$ for the black cluster. Hence, b_1 will be preferably assigned to the gray cluster.

4.3.2 Entity clustering with CLIP

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

Algorithm 4: CLIP

Input : $\mathcal{G} = (\mathcal{V}, \mathcal{E})$
Output: Cluster set $\mathcal{CS}$

/* PHASE 1 */
1 `determineLinkStrength` $(\mathcal{E})$
/* Links are classified so that $\mathcal{E} = \mathcal{E}_{Strong} \cup \mathcal{E}_{Normal} \cup \mathcal{E}_{Weak}$ */
2 $\mathcal{G}' = (\mathcal{V}, \mathcal{E}_{Strong})$
3 $\mathcal{CS}' \leftarrow$ `connectedComponents` $(\mathcal{G}')$
4 $\mathcal{CS} \leftarrow$ `getCompleteClusters` $(\mathcal{CS}')$
/* PHASE 2 */
5 $\mathcal{V}' \leftarrow \mathcal{V}$- $\mathcal{V}_{Complete}$, $\mathcal{E}' \leftarrow (\mathcal{E}_{Strong}$- $\mathcal{E}_{Complete}) \cup \mathcal{E}_{Normal}$
6 $\mathcal{G}' = (\mathcal{V}', \mathcal{E}')$
/* Vertices and links of the complete clusters are removed from the current graph $\mathcal{G}'$ */
7 $\mathcal{CS}' \leftarrow$ `connectedComponents` $(\mathcal{G}')$
8 **for** $(cluster_i \in \mathcal{CS}')$ ***in Parallel*** **do**
9 **if** (`isSourceConsistent` $(cluster_i)$) **then**
10 $\mathcal{CS} \leftarrow \mathcal{CS} \cup cluster_i$
11 **else**
12 *Assume* $v_t \in \mathcal{V}_{clusteri}$ *as a cluster* $t = 1, 2, ..., n$
13 $\mathcal{CS}_i \leftarrow \{c_1, c_2, ..., c_n\}$
14 $intraLinks \leftarrow \mathcal{E}_{clusteri}$
15 **repeat**
16 $\ell_{c_i,c_j} \leftarrow$ `getMaxPriority` $(intraLinks)$
17 `merge` (c_i, c_j)
18 `updateClusterSet` $(\mathcal{CS}_i)$
19 $removedLinks \leftarrow$ `removeConflictingLinks` $(intraLinks)$
20 $intraLinks \leftarrow intraLinks - removedLinks$
21 **until** $(intraLinks \neq \{\})$
22 $\mathcal{CS} \leftarrow \mathcal{CS} \cup \bigcup_{i=1}^{m} \mathcal{CS}_i$

The pseudocode of CLIP is shown in Algorithm 4. Its input is a similarity graph $\mathcal{G}$ and the output is the cluster set $\mathcal{CS}$. Figure 4.2 illustrates the algorithm for entities from four data sources *A*, *B*, *C*, and *D*. Entities with the same index are assumed to belong to the same cluster, e.g. entity a_0 from source A and b_0 from source B. The sample similarity graph in the example already links most matching entities but also contains wrong links, e.g. (b_0, c_1). In phase 1, we start with determining the strength of all links (line 1 of Algorithm 4). Then we only use strong links to determine graph $\mathcal{G}'$ (line 2). We then apply *connectedComponents* on $\mathcal{G}'$ to identify complete clusters and add these to the output (line 3-line 4). In the example of Figure 4.2, the second graph in the upper

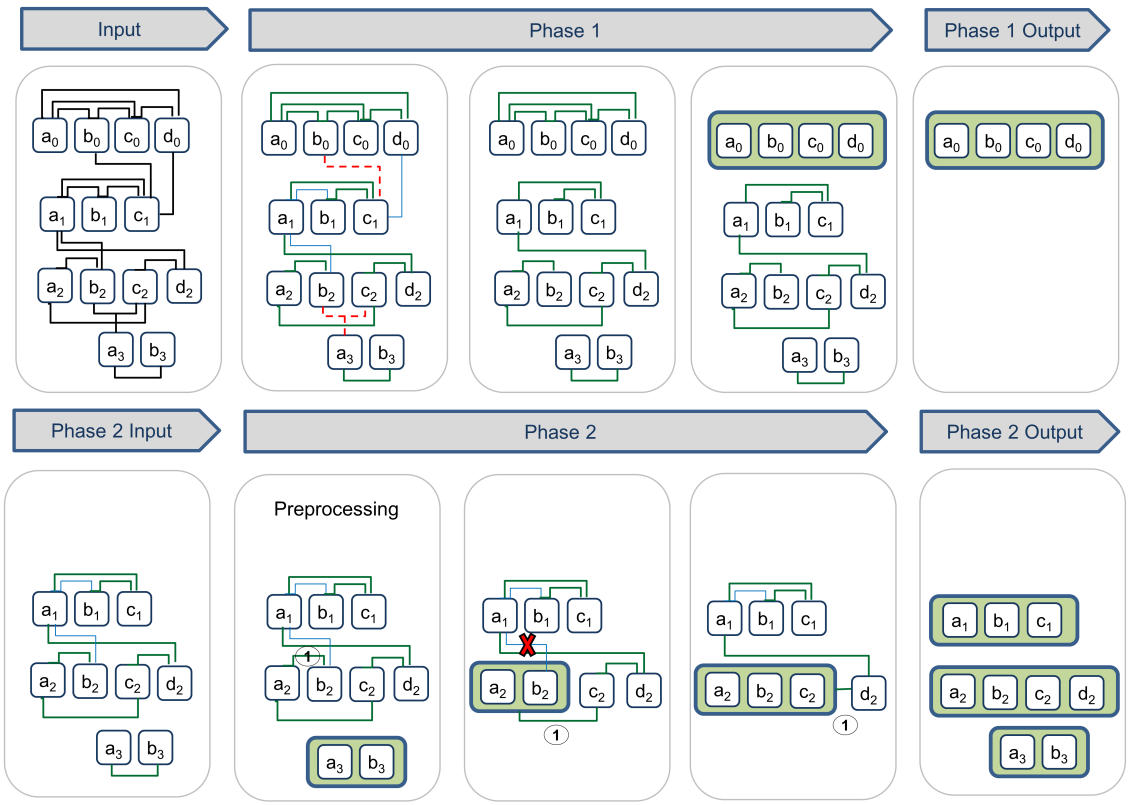


Figure 4.2: CLIP example

half differentiates between strong, normal, and weak links by showing them as green, blue and dashed lines, respectively. Focusing on strong links, we obtain four connected components in the example, one of which (for index 0) results in a complete cluster that is added to the output of phase 1.

For phase 2, we remove the vertices and edges from the complete clusters. Furthermore, we ignore weak links and only consider strong and normal links in the updated graph $\mathcal{G}'$ (line 5-line 6 of Algorithm 4). Again we use $connectedComponents$ to consider the resulting connected components as possible clusters (line 7). Afterwards these components $cluster_i$ are processed in parallel (line 8). If the cluster $cluster_i$ is already a source-consistent cluster, it is directly added to the CLIP output (line 9-line 10). Otherwise the component/cluster is source-inconsistent and will be iteratively processed as outlined below. In the example of Figure 4.2, phase 2 is illustrated in the lower part which starts with a reduced similarity graph that has no longer the entities from the complete cluster determined in phase 1 and that only contains strong and normal links. We then obtain two connected components one of which (with index 3) is already a source-consistent cluster that is thus added to the output. The remaining source-inconsistent component/cluster needs further processing.

In the processing of source-inconsistent clusters/components we initially consider each entity of component $cluster_i$ as a cluster c_i of its own (lines line 12-line 13). We then iteratively process the intra-component links (line 15-line 21) in the order of their maximal link priority and merge linked entity pairs from different sources into larger clusters such that no source inconsistency is created (line 17). For merged clusters, the cluster set is updated accordingly (line 18). Links from the newly formed cluster to entities of the same sources already present in the formed cluster (conflicting links) are removed from the intra-component link set (line 19-line 20). The process for each component terminates when the corresponding intra-component link set is empty (line 21). The union of all cluster sets $\mathcal{CS}_i$ determined in this way for the different components combined with the previously determined clusters in phase 1 form the final output of CLIP (line 22). In the example of Figure 4.2, we start with the link between a_2 and b_2 in the third graph for phase 2 and merge these entities into a new cluster. The link to a_1 from this newly formed cluster is considered as a conflicting link and therefore removed. In the next iterations the link priorities are updated and a new link with maximum priority is selected and clusters are merged. In the example this leads to adding entities c_2 and d_2 to the previously determined cluster while the link of this cluster to entity a_1 is in conflict and will be removed. Similarly, the cluster with index 1 can be generated. Together with the output of phase 1, four clusters are found in the example.

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

4.3.3 Cluster repair with RLIP

As explained in Section 4.1, RLIP aims at repairing the output of clustering algorithms by first resolving overlapping clusters, if necessary, and second by using CLIP to repair source-inconsistent clusters. We thus focus on overlap resolution.

The RLIP approach to resolve overlapping clusters also uses the intra-cluster links between entities[4] and favors strong links to select the cluster to which an overlapped entity should be assigned. In particular, overlapped entities that have only strong links to one cluster are assigned to this cluster and for overlapped entities with strong links to several clusters we choose the cluster with the highest association degree for this entity. Overlapped entities with no strong link are kept as singletons. The cluster decision cannot be made directly if an overlapped entity is only strongly linked to another

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

overlapped entity since the best result will depend on the cluster decision for the other overlapped entity. We therefore treat such cases in a second iteration of the algorithm. If in the second iteration the entity still is linked to only overlapped entities, all of them will become singletons.

Algorithm 5 outlines overlap resolution in more detail. The input is a set of cluster graphs $\mathcal{CS}$ and the output is a set of disjoint clusters $\mathcal{CS}_{output}$. The cluster graphs in the output can be merged into a similarity graph as input for the subsequent execution

Algorithm 5: Overlap Resolution

Input : $\mathcal{CS} = \bigcup_{i=1}^{m} \mathcal{CS}_i\,(\mathcal{V}_i, \mathcal{E}_i)$
Output: $\mathcal{CS}_{output}$

$\mathcal{CS}_{output} \leftarrow$ `determineLinkStrength` $(\mathcal{CS})$
/* Links are classified so that $\mathcal{E}_i = \mathcal{E}_{i(Strong)} \cup \mathcal{E}_{i(Normal)} \cup \mathcal{E}_{i(Weak)}$ */
for $(iterationNo := 1$ ***to*** $2)$ **do**
  $\mathcal{OV} \leftarrow$ `getOverlappedVertices` $(\mathcal{CS}_{output})$
  /* $\mathcal{OV}$ Vertices that belong to more than one cluster */
  for $v \in \mathcal{OV}$ ***in Parallel*** **do**
    $adjacentVertices \leftarrow$ `getStronglyLinkedPairs` (v)
    if $(adjacentVertices.size() = 0)$ **then**
      `updateClusterSet` $(\mathcal{CS}_{output}, v)$
      /* remove v and its associating links from all clusters. */
      $\mathcal{CS}_{output} \leftarrow \mathcal{CS}_{output} \cup$ (`formCluster`(v))
      /* v is a singleton. */
    else
      $associatedClusters \leftarrow \{\}$
      for $(v_n \in adjacentVertices)$ **do**
        if $(v_n \notin \mathcal{OV})$ **then**
          $associatedClusters \leftarrow$
           $associatedClusters \cup$ `getCluster`(v_n)
      if $(associatedClusters.size() = 0 \wedge iterationNo > 1)$ **then**
        `updateClusterSet`(v)
        $\mathcal{CS}_{output} \leftarrow \mathcal{CS}_{output} \cup$(`formCluster`$(v)$)
      else
        $resolvedCluster \leftarrow$
          $\underset{associatedClusters}{\text{argmax}}$ (`association`$(cluster_i, v)$)
        `updateClusterSet` (v)
  $iterationNo++$
$return\ \mathcal{CS}_{output}$

of CLIP. In line line 1, we first determine and store the strength of links in the input cluster graphs. Then, we determine the overlapped entities and process them in parallel in one or two iterations. For overlapped entity v, we store all strongly linked entities in *adjacentVertices* (line 5). If there is no such entity, the overlapped entity is kept as a singleton (line 6-line 8). Otherwise, the clusters of non-overlapped entities (line 11) of *adjacentVertices* are determined and stored in the set *associatedClusters* (line 10-line 13). If there is no such cluster, i.e., all strongly linked entities are overlapped entities, and we are in the first iteration we wait and this entity will become a singleton in the second iteration (line 14-line 16). Otherwise, the cluster association degree of the overlapped entity v to all members of *associatedClusters* is determined and v is assigned to the cluster with the maximal association degree (line 17-line 19). Obviously, if there is only one element in *associatedClusters*, v will go to this cluster.

Figure 4.3 illustrates overlap resolution for four input clusters (C_1, C_2, C_3, and C_4) where entities a_0, a_5 and d_4 belong to two clusters and entity b_2 even to three clusters. The algorithm starts by determining the strength of the links. In the second box of Figure 4.3, strong, normal and weak links are shown by green, blue and dashed red lines. The output of the first iteration (third box) shows that entity a_5 is considered as a singleton because it is not strongly linked to any other entity. Entity a_0 is assigned to cluster C_1 because of a higher association degree to C_1 than to C_2. Entity d_4 is strongly linked only to the overlapped entity b_2 so we do not decide about d_4 in this iteration. Entity b_2 has strong links to non-overlapped entities only to cluster C_3 so it is removed from clusters C_2 and C_4. In the second iteration (last box), the remaining overlapped entity d_4 is also resolved. It is linked to entity b_2 which has been assigned to cluster C_3 in the previous iteration, so d_4 is now also assigned to C_3 and removed from cluster C_4. We have thus resolved all overlaps although the resulting clusters are not necessarily

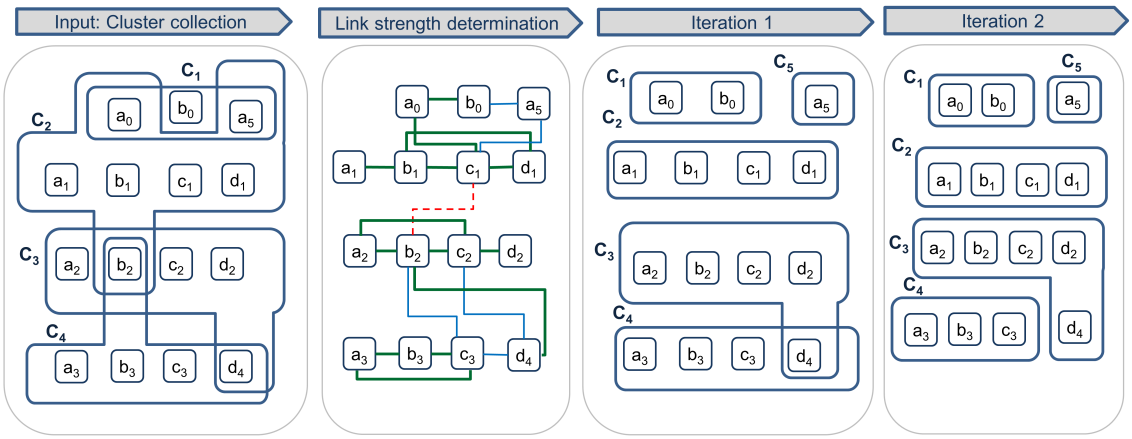


Figure 4.3: Overlap resolution (example)

source-consistent (e.g., cluster C_3 has two entities from source D). So the output of overlap resolution is then processed by CLIP to obtain both disjoint and source-consistent clusters.

4.4 Evaluation

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

4.4.1 Datasets and configuration setup

For our evaluation we use datasets from three domains for different numbers of duplicate-free sources. Table 4.1 shows the main characteristics of the datasets in particular the number of clusters and match pairs of the perfect ER result. The smallest dataset DS-G contains geographical real-world entities from four different data sources (DBpedia, Geonames, Freebase, NYTimes) and has already been used in the OAEI competition[5]. For our evaluation we focused on a subset of settlement entities as we had to manually determine the perfect clusters and thus the perfect match pairs.

For the two larger evaluation datasets DS-M and DS-P we applied advanced data generation and corruption tools [76] to be able to evaluate the ER quality and scalability for larger datasets and a controlled degree of corruption. DS-M is based on real records about songs from the MusicBrainz database but uses the DAPO data generator to create duplicates with modified attribute values [76]. The generated dataset consists of five sources and contains duplicates for 50% of the original records in two to five sources.

Table 4.1: Overview of evaluation datasets

	General information				Perfect result	
	domain	entity properties	#entity	#src	#clusters	#links
DS-G	geography	label, longitude, latitude	3,054	4	820	4,391
DS-M	music	artist, title, album, year, length	19,375	5	10,000	16,250
DS-P1	persons	name, surname, suburb, postcode	5,000,000	5	3,500,840	3,331,384
DS-P2			10,000,000	10	6,625,848	14,995,973

[5]OAEI 2011 IM: http://oaei.ontologymatching.org/2011/instance/

Table 4.2: Default blocking and match configuration for different datasets

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

All duplicates are generated with a high degree of corruption to stress-test the ER and clustering approaches. DS-P is based on real person records from the North-Carolina voter registry and synthetically generated duplicates using the tool GeCo [31]. We consider two configurations with either 5 or 10 sources each having 1 million entities; i.e. we process up to 10 million person records. Each source is duplicate-free, but 50% of the entities are replicated in all sources without any corruption. Moreover, 25% of entities are corrupted and replicated in all sources, and the remaining 25% are corrupted but present in only some sources. For the generation of corrupted records we applied a moderate corruption rate of 20%, i.e., most attribute values remained unchanged. The datasets are available on our website[6].

To generate the similarity graphs for the different datasets as the input of the clustering schemes, we experimented with a large spectrum of blocking and match configurations. Table 4.2 lists the default configurations that resulted already in good match quality even without clustering. All configurations apply standard blocking with different blocking keys. The match rules specify the conditions when a pair of entities is considered a match. As shown in Table 4.2, we use different similarity functions (string similarity functions or geographical distance) to compute attribute similarities and require the similarities to reach or exceed a minimal fixed or variable similarity threshold θ.

4.4.2 Match quality of clustering approaches

CLIP quality: We first evaluate the cluster quality achieved with the new CLIP clustering scheme in comparison with six known clustering schemes for the three datasets. For this purpose we assume that all entities in the determined clusters match with each other and determine the precision, recall and F-measure compared to the matches of the perfect cluster result. Figure 4.4 shows the achieved results for these metrics and different

[6] https://dbs.uni-leipzig.de/de/research/projects/object_matching/benchmark_datasets_for_entity_resolution

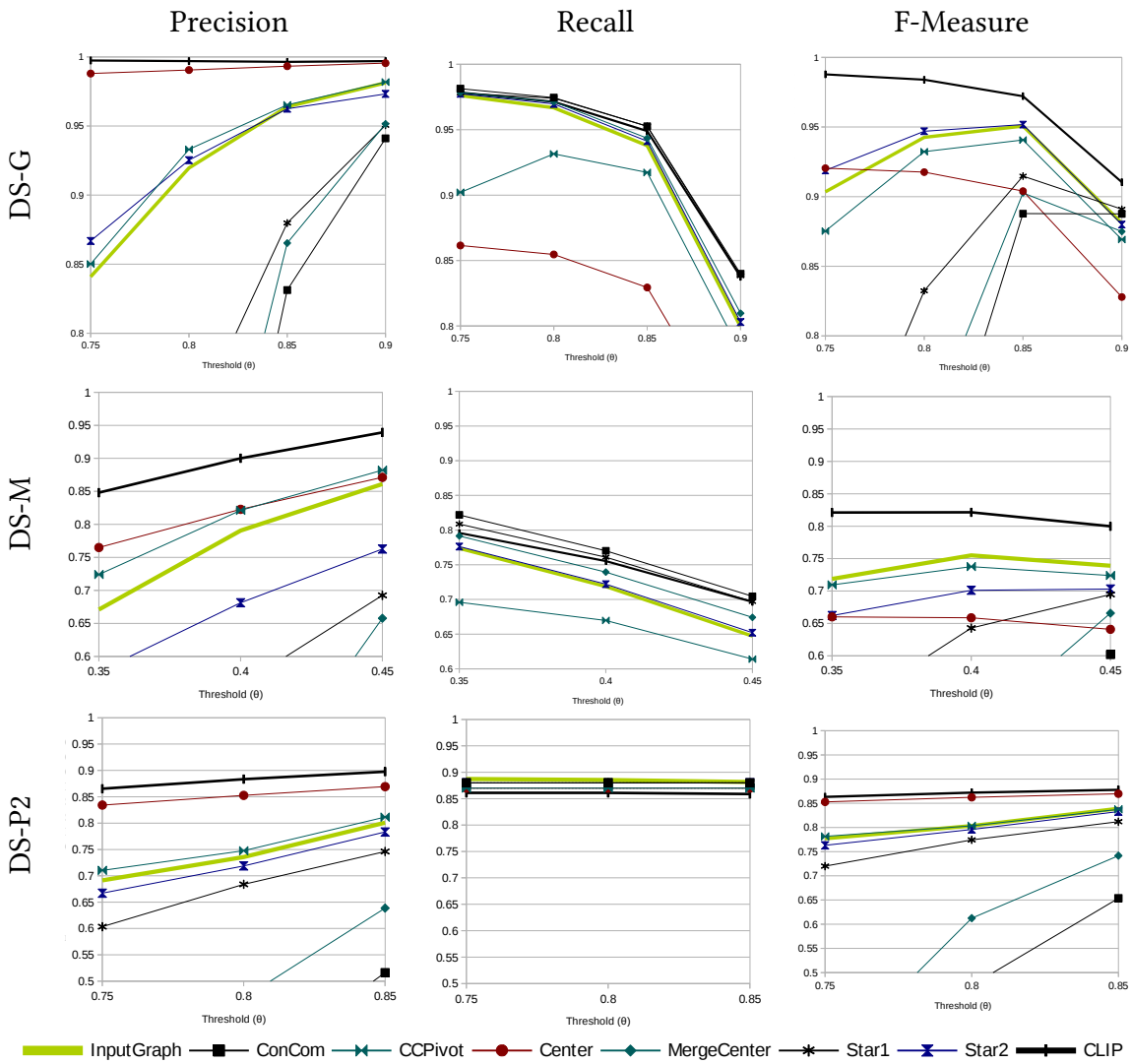


Figure 4.4: Cluster quality of CLIP vs other clustering approaches

similarity thresholds θ for the seven clustering schemes (CLIP, Connected Components, CCPivot, Center, MergeCenter, Star-1, Star-2) as well as for the similarity graph used as input to the clustering schemes (although this graph has only links but no clusters). The results for the six previous approaches correspond to those reported in [158].

We observe that CLIP achieves an excellent quality result and outperforms all previous algorithms in terms of precision and F-measure for all three datasets. Recall is comparable to the best approaches and only slightly worse compared to approaches such as Connected Components achieving the best possible recall (albeit at the expense of the poorest precision). A closer inspection of the CLIP behavior showed that its good recall is already achieved by determining the connected components for finding complete clusters and source-consistent clusters involving only strong and normal links. By

contrast, the CLIP iterations to split source-inconsistent components into several source-consistent clusters is primarily helpful to improve precision. The excellent precision of CLIP, even for lower similarity thresholds, is further due to the ignorance of weak links. This behavior is especially helpful for the relatively dirty dataset DS-M where we had to use very low similarity thresholds to achieve a sufficient recall. CLIP here achieves a F-Measure of 82% compared to only 65-75% for the other clustering schemes. Interestingly, the previous clustering schemes had even problems to outperform the link quality of the similarity graph due to wrong clustering decisions while CLIP achieves clear improvements compared to the similarity graphs by correctly clustering matching entities and removing wrong links between non-matching entities.

We additionally compared the CLIP quality with SplitMerge algorithm. The SplitMerge approach [120, 121] is a multi-source clustering algorithm that cluster entities in two phases of splitting and merging. It needs extra linking configuration parameters such as blocking configuration and similarity function for computing additional links between entities and similarity thresholds for the split and merge phases. The SplitMerge approach consists of three main phases: (1) determining initial clusters by applying connected components and making the components source-consistent, (2) splitting clusters to ensure a high intra-cluster similarity and (3) merging similar clusters.

In Figure 4.5, we compare the obtained precision, recall and F-Measure results for CLIP and SplitMerge algorithms as well as the results for a SplitMerge variation called *Split* that leaves out the merge phase for faster processing. We experimented SplitMerge with different values for the split and merge thresholds and we found that the split threshold should be chosen lower than the similarity threshold θ so that clusters are only split when there are links with a low similarity. By contrast, the merge threshold should be higher than θ so that only very similar clusters should be merged. The shown results refer to a fixed setting per dataset, e.g., a split threshold of 0.4 and a merge threshold of 0.8 for DS-G.

The recall of SplitMerge is always better than the recall of Split because it is based on connected components and the final merge phase helps to find additional links. A closer inspection of the CLIP behavior showed that its good recall is already achieved by determining the connected components for finding complete clusters and source-consistent clusters involving only strong and normal links. Comparing Split and SplitMerge, SplitMerge always achieves a slightly better F-Measure because its merge phase leads to a better recall than for Split that more than outweighs a somewhat reduced precision. For DS-M, Split and SplitMerge are better than CLIP due to the higher precision and recall while CLIP outperforms SplitMerge for DS-P due to a better precision.

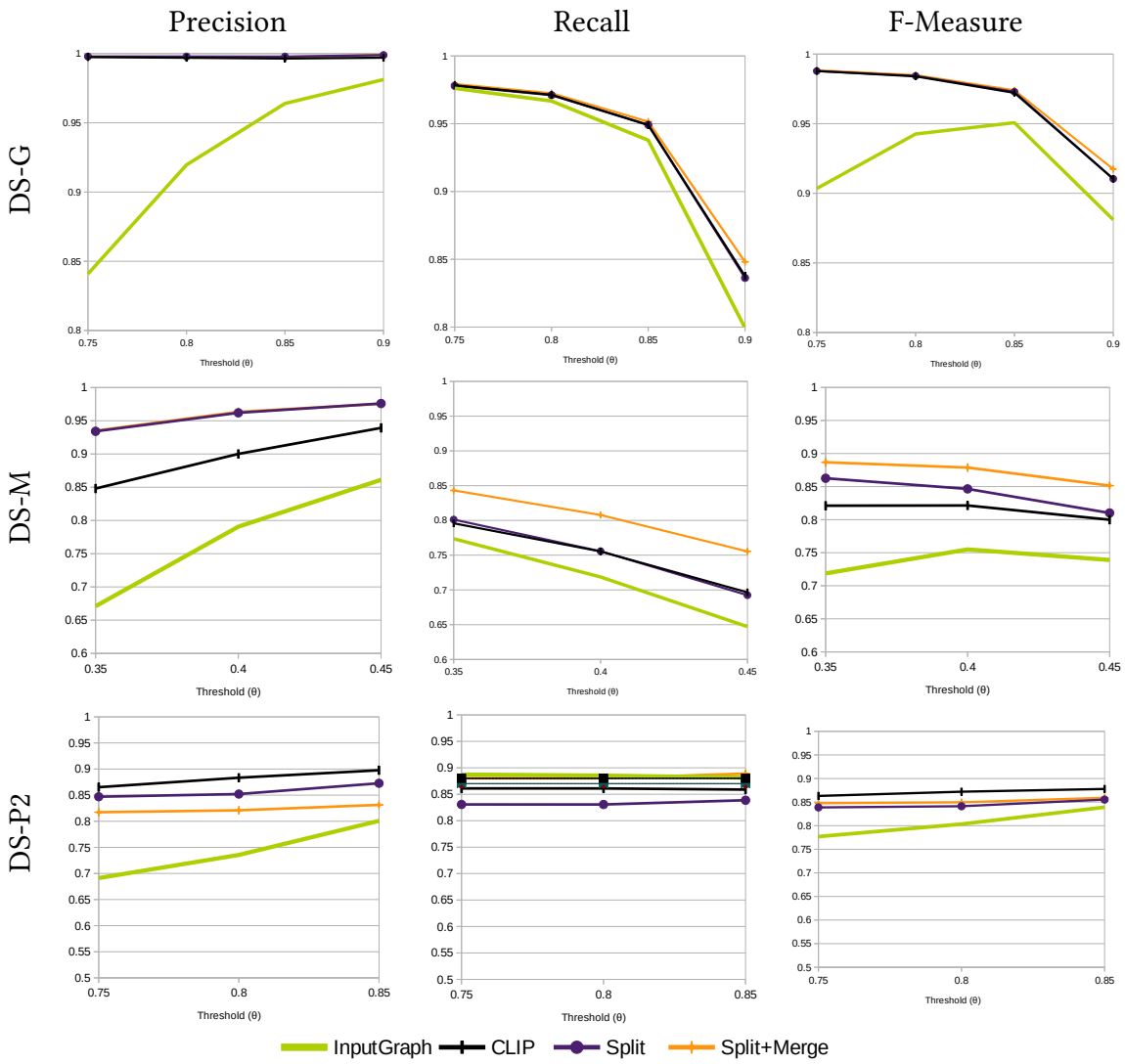


Figure 4.5: Cluster quality of CLIP vs Split/SplitMerge

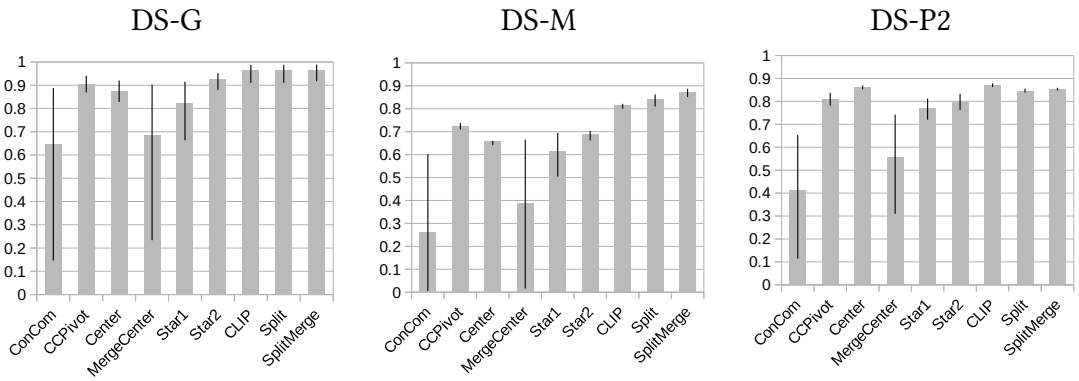


Figure 4.6: Average F-Measure results with range between minimal and maximal values

These observations are confirmed by Figure 4.6 showing the average F-Measure results of the clustering schemes over all threshold configurations. The vertical lines show the F-Measure spread between the minimal and maximal value for the different threshold values used to determine the input similarity graphs. We again observe the low and highly variable match quality of connected components and MergeCenter. By contrast, the remaining algorithms including the top-performing SplitMerge and CLIP algorithms are more robust and achieve much better F-Measure values. CLIP is similarly effective as Split and SplitMerge but it is easier to configure since it does not require the specification of additional similarity thresholds for splitting and merging.

RLIP quality: We now study the cluster quality achieved with the proposed repair approach RLIP when applied to the cluster results of the six previous clustering schemes. Figure 4.7 summarizes the achieved F-Measure results (averages over all considered values for similarity threshold θ) for the three datasets with the original clustering schemes only (blue bars on the left) and with additionally applying RLIP (red bars on the right). On the right we also show the average F-measure results for CLIP. We observe that RLIP can improve F-measure for all algorithms indicating an excellent effectiveness of the proposed cluster repair. The biggest improvements are achieved for the two poorest performing clustering schemes, Connected Component and MergeCenter, that both achieve a high recall but low precision. Here the CLIP component of RLIP achieves a substantial improvement in precision; the already high recall of the input enables that the repaired

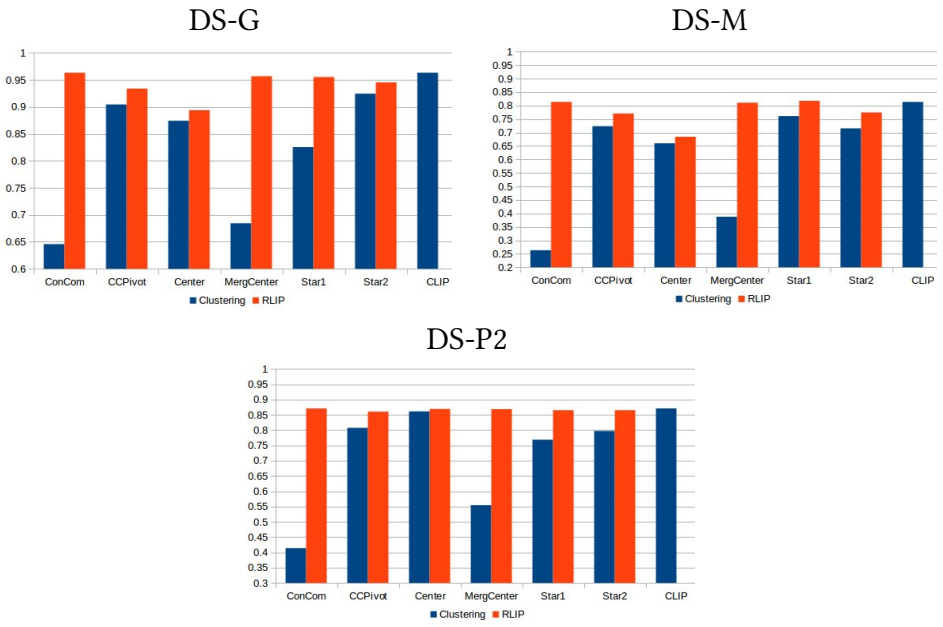


Figure 4.7: Cluster quality without and with repair using RLIP

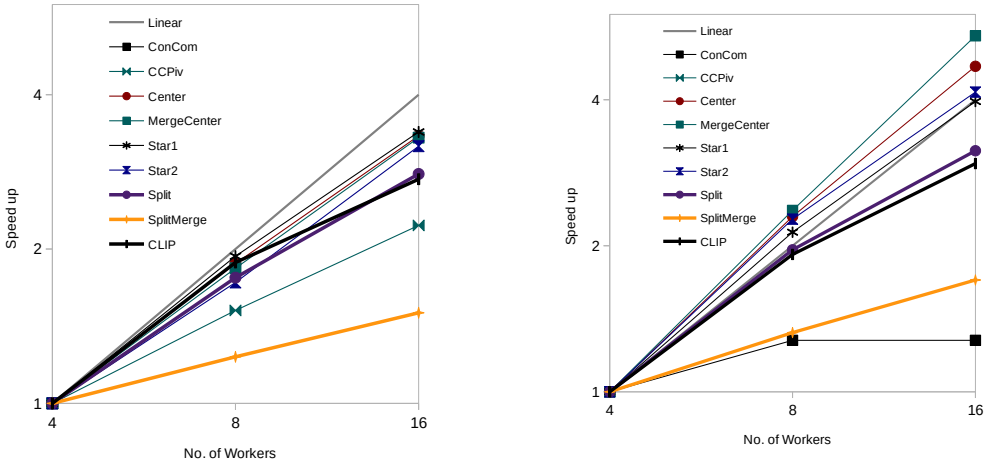
results for ConnectedComponent and MergeCenter are among the best overall. In fact, the repaired results for ConnectedComponent are essentially identical to the CLIP results. Overlap resolution is only applied to the results of Star-1 and Star-2 and helps to also achieve very good quality for their repaired clusters. The clusters determined with CCPivot and Center can be improved to a lesser degree since these algorithms remove already many links thus hurting recall. The only exception is DS-P for which all clustering schemes achieve a similarly high recall so that the repaired results after applying RLIP are also close together for all algorithms.

4.4.3 Runtimes and speedup

We determine the runtimes of the clustering algorithms on a shared nothing cluster with 16 worker nodes. Each worker consists of an E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM, two 4 TB SATA disks and runs openSUSE 13.2. The nodes are connected via 1 Gigabit Ethernet. Our evaluation is based on Hadoop 2.6.0 and Flink 1.1.2. We run Apache Flink standalone with 6 threads and 40 GB memory per worker. In our experiments, we vary the number of workers by setting the parallelism parameter to the respective number of threads (e.g., 4 workers correspond to 24 threads). The runtime of all algorithms is measured for the largest dataset DS-P with 5 and 10 parties applying the configuration from Table 4.2 with $\theta = 0.80$. The DS-P input data size is thus doubled for 10 parties compared to 5 parties. We only evaluate the runtimes for the clustering algorithms since the time to determine the similarity graphs is the same for all clustering approaches. Some clustering approaches could not be executed for 1 or 2 workers only due to high memory requirements. We thus evaluate the runtimes for configurations between 4 and 16 workers.

Table 4.3: Runtimes for clustering schemes (seconds)

dataset	DS-P1			DS-P2		
#workers	4	8	16	4	8	16
ConCom	51	57	55	101	79	79
CCPivot	1530	1008	688	-	-	1303
Center	390	208	117	1986	864	423
MergeCenter	640	349	194	3767	1592	695
Star-1	288	149	85	783	367	197
Star-2	214	124	67	720	317	173
Split	255	145	86	873	445	278
SplitMerge	1754	1423	1168	4792	3618	2819
CLIP	190	101	69	674	351	228



(a) DS-P1 **(b)** DS-P2

Figure 4.8: Speedup

Table 4.3 shows the measured runtimes for the two DS-P datasets. The increased dataset size for 10 parties leads to higher runtimes for all algorithms although to different degrees. As expected, the fastest runtimes are achieved by the simple connected components approach. By contrast, CCPivot and SplitMerge have the worst runtimes due to large memory requirements and a high message overhead for iterative processing. CCPivot even suffered from out-of-memory errors and could only be executed for 16 workers for the bigger dataset (10 parties).

The Split approach is much faster than SplitMerge and among the fastest of all algorithms. This shows that the final merge phase is the main performance bottleneck of SplitMerge since it requires the similarity computation for a large number of cluster pairs and an expensive iterative merge processing. CLIP is even faster than Split and thus among the fastest algorithms.

Except for connected components, all algorithms can reduce their runtimes by applying more workers, especially for the larger dataset with 10 parties. Figure 4.8 shows the resulting speedup values. For DS-P with 5 parties, most algorithms except the iterative CCPivot and SplitMerge approaches achieve an almost linear speedup. By contrast, the high-quality approaches Split and CLIP scale well for this dataset.

For the bigger dataset with 10 parties, speedup values are mostly even better and partly super-linear. The latter, however, is an artifact for the slower algorithms like Merge Center that perform poorly for 4 workers because of memory bottlenecks (its runtime for 4 workers is almost 6 times higher for 10 parties than for 5 parties). The substantially increased aggregate memory capacity for 8 and 16 workers thus enabled super-linear runtime improvements but without reaching the absolute runtimes of fast algorithms

like Star-2. Again, SplitMerge scales poorly due to the overly expensive merge phase while Split and CLIP achieve both low absolute runtimes and good speedup.

The high runtimes for SplitMerge (and CCPivot) are heavily influenced by the underlying Flink and Gelly systems and its approaches for iterative processing leading to high memory and communication overhead.

4.5 Case Study

We outline the use of the tool FAMER to address the schema and entity matching tasks for the DI2KG 2019 challenge[7]. We have shown that the FAMER tool could reasonably well solve the entity resolution task of the challenging 2019 DI2KG dataset. While there is still room for improvement, our approach determined matches that helped the conference organizers to enhance the golden truth (which thus may be more a silver truth). We could also provide a reasonable solution for (simplified) property matching, but more effort is necessary to achieve a full-fledged solution.

4.5.1 Task Definition

The challenge comprised of three main tasks:

- Entity Resolution
- Schema Alignment
- Knowledge Graph Augmentation

Each task required participants to build a knowledge graph consisting of a set of predefined entities and properties. Participants were provided with a set of selected HTML pages regarding products from a variety of sources, each page correlated with a JSON file containing the result of an automated process of specifications extraction. The JSON files consist of a series of key and value pairs extracted from the associated HTML page as depicted in Listing 4.1.

Participants were also provided with a set of records from the manually built ground truth in order to have the possibility of training models. The ground truth was partitioned in two parts. One part was available in the download for training or testing by the users. The second part was going to be used by the challenge organizers for evaluating submitted solutions and was not disclosed to participants.

[7] http://di2kg.inf.uniroma3.it/2019/#challenge

Listing 4.1: A json file example

```
{
    "<page title>": "Nikon Coolpix S3600 Digital Camera - Silver (VNA550AA) | Camerafarm
         Australia",
    "battery  batteries": "Rechargeable Li-ion Battery EN-EL19",
    "brand": "Nikon Web Site",
    "charging time": "2 hours and 50 minutes (when using Charging AC Adapter EH-70P and
        when no charge remains) (Approx.)",
    "date added": "28/4/2014",
    "exposure compensation": "2 EV in steps of 1/3",
    "exposure control": "Programmed auto exposure Exposure compensation (-2.0 to +2.0 EV
         in steps of 1/3 EV)",
    "exposure metering": "Matrix Center-weighted (digital zoom less than 2x) Spot (
        digital zoom 2X or more)",
    "exposure modes": "Auto Scene Scene Auto Selector Smart Portrait",
        "incamera image editing": "Copy Crop D-Lighting Filter Effects Glamour Retouch
            Print Order Protect Quick Retouch Red eye correction Rotate Image Slide Show
             Small Picture Voice Memo",
    "manuf no": "VNA550AA",
    "our price": "122.05",
    "part no": "316972",
    "rating": "Not Rated",
    "rrp": "$149.00",
    "ship weight": "2 KG\nDetails",
    "usually ships": "1-3 Days\nMore info",
    "you save": "$26.95",
    "power sources": "One Rechargeable Li-ion Battery EN-EL19 (supplied) AC Adapter EH-6
        2G (available separately)"
}
```

4.5.2 Preparation

In this section the preparation steps including the schema matching part done by my colleague Daniel Obraczka is explained.

To illustrate the data quality problems in the given dataset of the DI2KG challenge, we show in Table 4.4 two matching *Nikon* camera products from different sources. We observed significant differences in the set of properties and property values. For example, the first entity owns the property *features* while the second camera does neither contain this property nor the corresponding value (*Slimline*). This may happen even among entities of the same source. Moreover, the same property values are not represented similarly in different entities. For example, in the first camera the property *camera resolution* with the value *16 Megapixels* is represented as *"approx resolution": "16MP"* for the second camera. Altogether, the challenge includes 24 sources with vastly heterogenuous schemas. For example, the source "www.ebay.com" has over 2000 properties some of which are likely duplicate properties such as "maximum shutter speed" and "max shutter speed".

FAMER currently expects to be provided with already matched properties for entity resolution. For the DI2KG challenge, we therefore need to first align the properties before we can apply our entity resolution approach. To this end, prior to performing the

Table 4.4: Example raw data

property	value
”35mm equivalent”	”25-300mm”
”<page title>”	”Nikon Coolpix S6800 Digital Camera (Black) | UK Digital Cameras”
”brand”	”Nikon”
”camera resolution”	”16 Megapixels”
”colour”	”Black”
”features”	”Slimline”
”hd video”	”Full HD (1080P)”
”lcd size”	”3.0””
”lens tele mm”	”300”
”lens wide mm”	”25”
”mpn”	”VNA520E1”
”optical zoom”	”23”
”optical zoom range”	”18x and higher”
”<page title>”	”Nikon Coolpix S6800 Price in India with Offers, Reviews & Full Specifications | PriceDekho.com”
”color”	”Black”
”amazon”	”Infibeam Ebay Homeshop18 Snapdeal Flipkart”
”digital zoom”	”4x”
”bangalore”	”Hyderabad Chennai Mumbai Delhi Pune”
”approx resolution”	”16 MP”
”external memory”	”Yes”
”face detection”	”NA”
”gps”	”NA”
”hdmi”	”NA”
”maximum shutter speed”	”1/2000 sec”
”metering”	”NA”
”minimum shutter speed”	”1 sec”
”optical zoom”	”18x”
”screen size”	”3 Inches”
”usb”	”Yes”
”video display resolution”	”NA”
”wifi”	”Yes; Wi-Fi 802.11 b/g/n”

entity matching, an incremental schema matching was performed. The schema matching initially starts with *preprocessing* on the input dataset to derive some statistics and to perform data cleaning steps. In particular, both entity and schema matching are focused on the most frequent properties since infrequent properties are unlikely to be present for all matching pairs of entities so that their use is of limited value. For example, the property *energy consumption per year* only occurs in one entity in the entire dataset and will therefore most likely not have a corresponding property in other sources and is thus useless for entity resolution. Therefore, for each source the k (≤ 10) most frequent properties are determined.

Moreover data cleaning is performed to harmonize property values to make similarity computations more meaningful. For example, different units are used for *weight* in different sources. Comparing values in *ounces* with values in *grams* would lead to a poor similarity value and we therefore transform both into the same unit. Further data

cleaning procedures are performed, such as lowercasing strings and using canonical abbreviations.

The incremental property matching aims at calculating a combined similarity between properties of a new source and already considered properties of previous sources of the same category. The similarity between two properties is based on the similarity of property names and the aggregated similarity of all property values. The property values are derived from all relevant matches for the considered sources from the training data.

The calculated similarities are used to build and update a similarity graph consisting of the properties as vertices and the similarities as edges. This graph is given to FAMER's Clustering module to determine new property clusters. This is iteratively done until no more sources are left to integrate. The resulting property clusters can now be used in the entity resolution step by fusing all members of a cluster to a new property.

4.5.3 Entity Matching

FAMER assumes the knowledge of matching properties for both blocking and pair-wise linking. We therefore use the schema matching result and data cleaning for the most frequent properties to harmonize the entities before entity resolution. Table 4.5 indicates the improved data of Table 4.4 after preprocessing and property alignment. As illustrated we consider only a subset of the properties and both the property names and some property values have been harmonized.

FAMER provides many options to perform entity resolution for the prepared dataset and we aim at a comparative evaluation of several configurations. In particular, we can apply a batch-like (static) matching and clustering for all (24) sources at once or we can apply an incremental approach that iteratively adds and matches one source after the other. We decided to compare a batch-like approach, which we will denote as *1step*, and an alternative approach dubbed *2step*, in which we first deduplicate each source

Table 4.5: Example data after preprocessing and property alignment

property	value
page title	nikon coolpix s6800 digital black
manufacturer	nikon
resolution	16 mp
color	black
optical zoom	18x
screen size	3.0 inch
page title	nikon coolpix s6800
resolution	16 mp
color	black
optical zoom	18x
digital zoom	4x
screen size	3.0 inch

independently, fuse duplicate entities and then perform matching and clustering on the deduplicated sources.

In both cases blocking is done on the *manufacturer* property that is needed for a sufficiently low runtime. The camera products lacking the value of *manufacturer* form a special block and are matched with all other entities.

The most promising linking configuration used the following weighted similarity:

$$sim(e_1, e_2) = \omega_1 * productSim(e_1, e_2) + \omega_2 * JaroWinkler(e_1, e_2),$$

where ω_i are weights. The similarity $productSim$ is 0 or 1 depending on whether the product codes of the entities e_1 and e_2 match. The product codes are extracted from the *page title* attribute. Finally, $JaroWinkler$ is the JaroWinkler similarity performed on the concatenation of all respective properties of the entities except the page title.

The third approach we submitted utilized machine learning. We used the provided training data as input to Magellan's [92] XGBoost [24] implementation. As before we used the first 2 letters of manufacturers. Negative training examples were created by taking the most dissimilar entities in a block. Since Magellan is only able to perform pairwise matching we ran this approach for all possible data source pairs, where training data was available. The trained classifiers were then used to classify unseen entity pairs and the resulting classifier probabilities were used to create a similarity graph of all sources. Finally, FAMER's Clustering module was used on this similarity graph.

4.5.4 Results

In this section we will describe the performance of our approaches on the tasks *schema matching* and *entity matching* of the DI2KG challenge 2019. We present the evaluation of our results at the time of our submission, as well as the results obtained from the workshop organizers. Unfortunately, we could not directly use the ground truth for a comprehensive evaluation but had to rely on the results determined by the workshop organizers.

As described in Section 4.5.3, we submitted results of three different entity resolution approaches. While we initially also wished to employ word embeddings in these methods, in initial tests this technique did not prove as promising for the given dataset. The first two approaches consist of manually created configurations of our system, while the third utilized machine learning. For the weighted similarity, used in the first two approaches, the best weights were determined to be $\omega_1 = 0.6$ and $\omega_2 = 0.4$. The results are presented in Table 4.6. Before submission we created a test dataset to avoid only evalu-

Table 4.6: Performance of ER approaches on training data and ground truth

Measure	Training data				Ground truth		
	1step	2step	ML(train)	ML(test)	1step	2step	ML
F-Measure	0.91	0.88	0.59	0.60	0.64	0.56	0.002
Precision	0.99	0.98	0.77	0.77	0.78	0.59	0.06
Recall	0.84	0.79	0.48	0.50	0.54	0.54	0.001

ating the machine learning approach on the data we trained on. To obtain this test data, the entities with the most similar page titles in a block where regarded as true matches, and the most dissimilar entities where regarded as non-matches. At the time of submission we already observed that our manually created configurations were superior to the machine learning approach. We attribute this to the low number of training examples (especially per source-pair). The conference organizers informed us that our first two approaches enabled them to augment their ground truth with roughly 800 new entities that were previously not identified as matching indicating that the considered ground truth is not yet in a perfect state. We can see a huge difference between the performance on the training data set and the larger ground truth. This might indicate that the training data generally contains simpler examples, or our methods overfit to the training data. The bad performance of the machine learning approach on the whole ground truth is not explainable at this point and might be due to some error. Unfortunately, a more detailed analysis of this issue was not yet possible due to the unavailability of the ground truth for us.

Our *1step* method outperformed the *2step* method. We assume, deduplicating each source and fusing detected duplicate entities in one entity, may create false links in the 2nd step between the wrongly fused entities and other entities from other sources explaining the relatively low precision for the ground truth (Table 4.6). The more detailed comparison of 1-step vs. 2-step approaches is another topic for future study.

4.6 Related Works

Most previous ER algorithms try to find matches either in a single source or between two sources only. For a single source, matching entities are typically grouped within disjoint clusters such that any two entities in a cluster should match with each other and no entity should match with entities of other clusters. For two sources, the match result is mostly a binary mapping consisting of pairs of matching entities (also called match correspondences or links). Binary match mappings may be postprocessed to determine clusters of matching entities, e.g., by calculating the transitive closure of the

correspondences (connected components) in the simplest case. In FAMER, we extend this approach to more than two sources by first determining a similarity graph with binary match links between entities and then determining clusters of matching entities within the similarity graph. A similar use of similarity graphs has been considered in [144].

Hassanzadeh and colleagues [68] comparatively evaluated several clustering methods for single-source ER. The FAMER tool includes distributed versions for a subset of the best algorithms from [68] and supports their use for clustering entities from multiple sources. While the problem of cluster overlaps was already observed in [68], considering multiple sources also leads to the problem of source-inconsistent clusters that we addressed in this chapter. Clustering entities from multiple sources of course leads to increased difficulties to achieve high performance and effectiveness compared to considering only one or two data sources. We aimed at supporting scalability and efficiency by applying both blocking and parallel processing. Furthermore, we proposed advanced clustering methods that avoided the problems of previous clustering schemes and achieved a better match and cluster quality. We are not aware of other tools supporting a parallel entity clustering for multiple sources.

Repair was already studied in pairwise ontology matching, e.g., to ensure that mappings only contain 1:1 matches or to correct other mapping-induced inconsistencies [21, 145]. However, repair for multiple source mappings was not covered. For entity resolution, [173] and [127] already investigated repair techniques mainly by exploiting the transitive closure of matches to add or remove match links. The repair of entity clusters proposed in [173] depends on manual user feedback which is difficult to provide for large datasets. Ngonga et. al. are not concerned with entity clusters but focus on finding missing/wrong links and also try to repair entities replicated in different sources [127]. By contrast, CLIP and RLIP avoid/repair overlapping and source-inconsistent clusters in multi-source entity resolution utilizing different link features. Moreover, CLIP and RLIP are implemented as parallel algorithms on Apache Flink to allow for large-scale entity resolution.

4.7 Conclusion

This chapter proposed a new method called CLIP to cluster matching entities from multiple sources as well as a repair method called RLIP to improve entity clusters determined by other clustering schemes. The approaches avoided or resolved overlapping and source-inconsistent clusters and utilized several features of similarity links in a new

way, in particular the link strength. Our evaluation for three datasets showed that the new approaches achieve excellent cluster quality and outperform previous clustering schemes to a large degree. The RLIP repair approach could improve the quality for all considered clustering schemes and achieved comparable quality than applying CLIP alone. The parallel implementations for CLIP and RLIP achieved good speedup values thereby supporting scalability to larger datasets.

SplitMerge with numerous trial and error experiments for finding the best threshold could compete and even be superior than CLIP. The output quality of SplitMerge is highly dependent on the input linking configurations and the runtime and scalability are not convincing.

5

Multi-source Clean/Dirty Clustering

The subsequent chapter is based on [38, 101, 102, 156]. Previous approaches for matching and clustering entities between multiple (> 2) sources either treated the different sources as a single source or assumed that the individual sources are duplicate-free, so that only matches between sources have to be found. This chapter proposes and evaluates a general Multi-Source Clean Dirty (MSCD) scheme with an arbitrary combination of clean (duplicate-free) and dirty sources. For this purpose, we extend a constraint-based clustering algorithm called Affinity Propagation (AP) as well as the Hierarchical Cluster Analysis (HCA) for entity clustering with clean and dirty sources (MSCD-AP/MSCD-HAC). We also consider parallel solutions of them for improved scalability. Our evaluation considers a full range of datasets containing 0% to 100% of clean sources. We compare our proposed algorithms with other clustering schemes of FAMER in terms of both match quality and runtime. The proposed algorithms outperform previous methods and achieve an excellent precision in MSCD scenarios. The MSCD clustering concept was presented on the BTW conference in 2021 and published in the corresponding proceeding.

5.1 Motivation

As described in Chapter 4, most previous ER approaches focus on finding matches in either a single source or between two sources. Multi-source ER aims at finding matching entities in an arbitrary number of sources which is more challenging than dealing with 1-

2 sources since not only the degree of heterogeneity but also the variance in data quality generally increases with the number of sources.

In Chapter 4, the clustering approaches for clustering matches in a single source and clean multi-source are investigated. In this chapter, we investigate a so-called Multi-Source Clean Dirty (MSCD) entity clustering approach that is able to utilize clean sources but can also deal with dirty sources so that only a fraction (possibly 0%) of the sources have to be clean. The goal is to achieve better match quality than with a general clustering scheme when there are clean sources while avoiding the limitation of requiring that all sources have to be clean. While one could first deduplicate dirty sources and then applying a clustering for clean sources, the effort to determine these source-specific deduplication approaches is immense and perhaps not completely successful[1]. Consequently, it is much more flexible to support a mix of both dirty and clean sources. For this purpose, we propose extensions to the Affinity Propagation (AP) clustering approach [53] that converts the problem of clustering into a constraint optimization problem. Our extension MSCD-AP adds a new constraint to deal with clean sources. We also consider a hierarchical variation of MSCD-AP for improved scalability. In addition, we investigate the use of Hierarchical Agglomerative Clustering (HAC) for MSCD entity clustering. The provided parallel implementations of all methods are based on Apache Flink.

This chapter makes the following contributions:

- Consideration of a mix of clean and dirty sources for multi-source entity resolution.
- Proposal of an extended version of Affinity Propagation clustering, MSCD-AP, for clustering entities of a mix of clean and dirty sources.
- Proposal of hierarchical variation of MSCD-AP for improved scalability and providing parallel implementations for the clustering schemes based on Apache Flink.
- Proposal of MSCD-HAC algorithm for multi-source entity clustering with a combination of clean and dirty sources. The clusters to merge in the next iteration can be selected based on the maximal, minimal, or average similarity of their cluster members. The approach utilizes the clustering constraint for clean sources and can optionally ignore so-called weak links in the similarity graph for improved quality and runtime.
- Proposal of parallel MSCD-HAC to support scalability.

[1]We experimented with such an approach for a data integration challenge [131] but it performed worse than with matching dirty sources (see Section 4.5)

- Comprehensive evaluation of match quality, runtimes and scalability of the new approaches for different datasets and comparison with previous clustering schemes.

Section 5.2 initially gives a brief summary of the standard AP algorithm and presents the new methods in detail. Section 5.3 elaborates on hierarchical cluster analysis and its newly introduced usage for multi-source clustering. Finally, Section 5.4 shows the evaluation results. We discuss the related work in Section 5.5 and conclude in Section 5.6.

5.2 Affinity Propagation for Multi-source Clean/Dirty Datasets

In this section we briefly explain the standard Affinity Propagation algorithm [53] and then present the Multi-Source clean/Dirty Affinity Propagation (MSCD-AP) and its scalable version in detail.

5.2.1 Affinity Propagation

The Affinity Propagation clustering algorithm [53] groups entities by identifying so-called exemplars. An exemplar is the entity that best represents all the entities of a cluster. The non-exemplar entities are assigned to the most appropriate exemplar. The goal of AP is to find exemplars and cluster assignments in a way that the sum of similarities inside clusters are maximized. In [63], AP is solved by the iterative max-sum algorithm on a factor graph. The factor graph is a bipartite graph between the exemplar assignments (variable nodes) and so-called factor nodes representing two constraints, called the g- and h-constraints.

Figure 5.1[2] illustrates such a factor graph for AP. Variable nodes and factor nodes are represented as circles and rectangles respectively. For clustering n entities, the factor graph is represented by a n^2 binary matrix $\mathbb{B}$. The variable b_{ij} has the value 1 if the datapoint (entity) j is the exemplar of i. The factor nodes g_i and h_j assure a valid clustering by applying the constraints. The g-constraint enforces that a datapoint has to have exactly one exemplar. It means in each row of the binary matrix there must be exactly one variable with value 1. The h-constraint assures that a datapoint selects itself as its exemplar, if it is already chosen as exemplar by at least one other datapoint. It means, if there exists at least one 1 in a column of the binary matrix, then the diagonal element b_{jj} of that column must be set to 1 too. The cluster assignments are based on the similarity between

[2]The figure is from the paper "Extended Affinity Propagation: Global Discovery and Local Insights" (arxiv: 1803.04459)

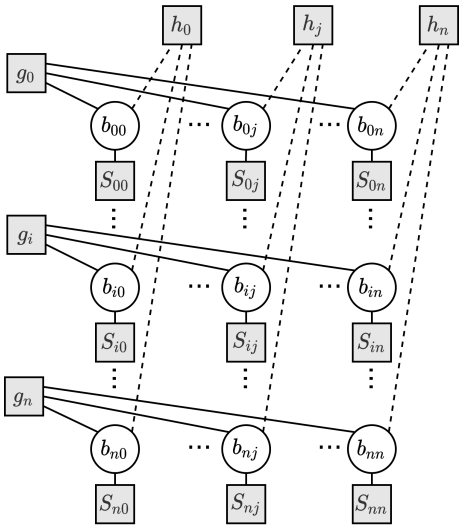
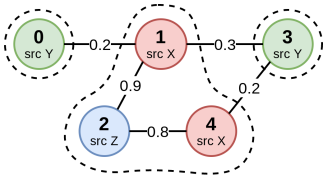


Figure 5.1: Factor graph of AP



(a) AP clustering example

	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	1	0	0	0
3	0	0	0	1	0
4	0	1	0	0	0

(b) Binary matrix

s_{ij}	0	1	2
0	0,8	**0,9**	0,1
1	**0,9**	0,8	0,1
2	0,1	0,1	0,8

1	0	0	0	1	0
1	0	0	0	1	0
0	0	1	0	0	1

(c) Oscillation

Figure 5.2: Affinity Propagation concepts

entities so that similarity values are also represented as factor nodes (factor node S_{ij} provides the similarity information between entities i and j). Figure 5.2a illustrates an example clustering of AP where five entities 0-4 from three (differently colored) sources X, Y and Z are grouped in three clusters. The corresponding output binary matrix in Figure 5.2b shows that entities 0, 1 and 3 are the exemplars of the three clusters. As described above, the rows of the binary matrix illustrate the exemplar (cluster) assignment while the columns depict the clusters. The group of 1 values in column j represents the entities of the cluster with exemplar j.

AP aims at finding a cluster assignment maximizing the sum of similarities within clusters. This optimization problem can be formulated with the energy function shown in Eq. (5.1). Maximizing Eq. (5.1) requires to find an optimal configuration of the variables in $\mathbb{B}$ so that the sum of the similarities between entities and their exemplars is maximized and the two constraints are met. An exact maximization of the energy func-

tion is computationally intractable because a special case of this maximization problem is the NP-hard k-median problem [53].

$$E(\mathbb{B}) = \sum_{ij} s_{ij} b_{ij} + \sum_{i} g_i(\mathbb{B}(i,:)) + \sum_{j} h_j(\mathbb{B}(:,j)) \tag{5.1}$$

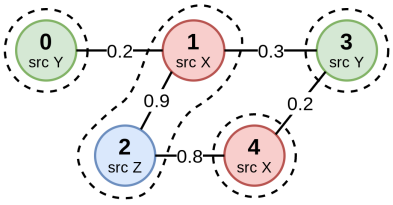
with

$$g_i(\mathbb{B}(i,:)) = \begin{cases} 0 & \text{if } \sum_j b_{ij} = 1 \\ -\infty & \text{otherwise} \end{cases} \qquad h_j(\mathbb{B}(:,j)) = \begin{cases} 0 & \text{if } b_{jj} = \max_i b_{ij} \\ -\infty & \text{otherwise} \end{cases}$$

The proposed iterative max-sum algorithm uses several parameters that affect the clustering result and that deal with the problem of non-convergence. The most important parameter is called *preference*. It defines the self-similarity S_{ii} of an entity i. The higher the preference value is chosen the more likely the entity becomes an exemplar. Parameters to deal with non-convergence are the noise level and the damping factor λ. AP suffers from oscillation between solutions that are similarly well suited for optimizing the energy function. For the similarity matrix in the top portion of Figure 5.2c, the symmetrical similarity values between entities 0 and 1 make both equally well suited as an exemplar. In such a situation, AP does not converge and oscillates between the two solutions with either entity 0 or 1 as the exemplar as shown in the bottom part of Figure 5.2c. Oscillation is avoided by adding a tiny amount of noise to the similarity values. The damping factor has a similar goal and is related to the used message passing implementation for the iterative computation and leads to an adaptation of values exchanged between iterations. If oscillations nevertheless occur, the noise or the damping factor must be adapted (see next section).

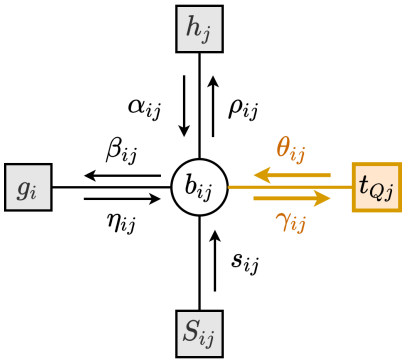
5.2.2 MSCD Affinity Propagation

For clustering mixed datasets of clean and dirty sources, we propose an extension to AP called MSCD-AP. Since clean sources have no duplicates every cluster should have at most one entity of a clean source. This is now controlled by an additional *clean-source consistency* constraint. It means that in each column of the binary assignment matrix $\mathbb{B}$, value 1 is only allowed for at most one (row) entity of a clean source. Figure 5.3a shows a possible clustering of MSCD-AP for the running example when sources X and Y are clean. There are four *source-consistent* clusters with at most one entity per clean source. In the corresponding binary matrix, each column has at most one entity with value 1 per



	0	1	2	3	4
0	1	0	0	0	0
1	0	1	0	0	0
2	0	1	0	0	0
3	0	0	0	1	0
4	0	0	0	0	1

(a) MSCD-AP clustering example



(b) Messages of the MSCD-AP factor graph

Figure 5.3: MSCD-AP concepts

clean source. For example, the column (cluster) for exemplar entity 1 has two associated entities (1 and 2) from different sources.

Our proposed clean-source consistency constraint is expressed in Eq. (5.2). It uses a function t to assign penalty 0 if the constraint is obeyed and a large penalty otherwise. The constraint requires that for a column j the value 1 is allowed for at most one datapoint from a clean source Q. The t function is used in an extended energy function as listed in Eq. (5.3).

$$t_{Qj}(\mathbb{B}(i \in Q, j) = \begin{cases} 0 & \text{if } \sum_{i \in Q} b_{ij} \leq 1 \\ -\infty & \text{otherwise} \end{cases} \tag{5.2}$$

$$E(\mathbb{B}) = \sum_{ij} s_{ij} b_{ij} + \sum_{i} g_i(\mathbb{B}(i,:)) + \sum_{j} h_j(\mathbb{B}(:,j)) + \sum_{Q} \sum_{i \in Q, j} t_{Qj}(\mathbb{B}(i,j)) \tag{5.3}$$

Figure 5.4 illustrates the extension of the AP factor graph to cluster our running example data. For clean sources X and Y, additional factor nodes t_x and t_y (marked in red and green) are added to each column of the binary matrix. The factor node t_{xj} assures the clean-source consistency constraint for source X and column j. It is connected to the variable node b_{ij} only if entity i is from data source X. The clean-source constraint may get in conflict with the h-constraint of AP. The h-constraint enforces a datapoint

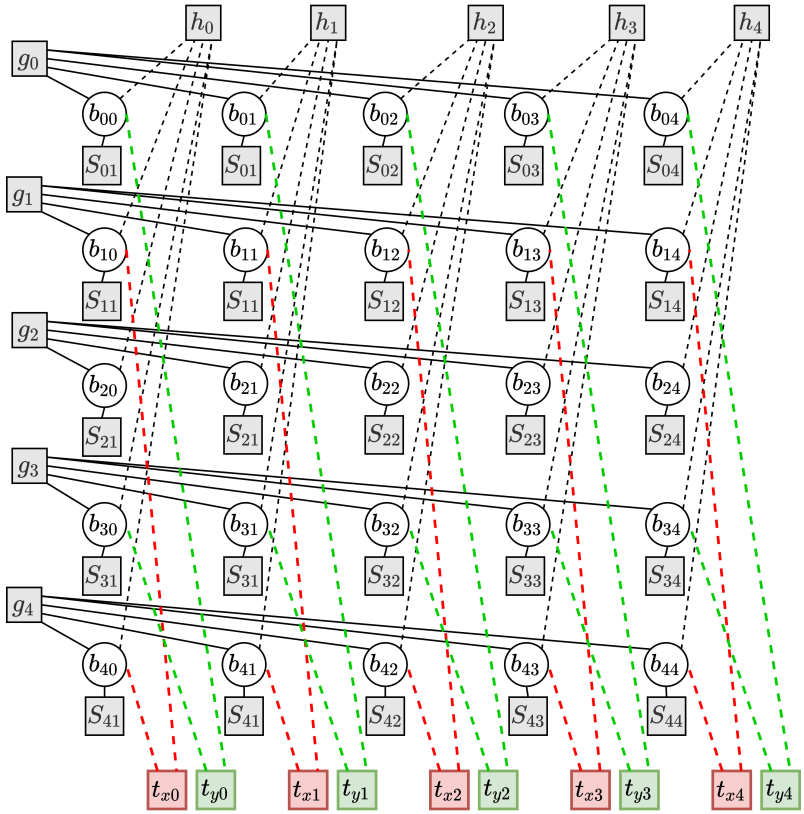


Figure 5.4: The factor graph of MSCD-AP for the running example

to choose itself as its own exemplar, if it is selected by at least one other datapoint. So the diagonal element b_{jj} of column j is enforced to be 1, if there is any other 1 in that column. On the other hand, the clean-source constraint enforces b_{jj} to be 0, if another datapoint of the same clean source selected it as its exemplar. So the two constraints enforce different values for b_{jj} and thus the algorithm may struggle to converge. This situation is simply avoided in our implementation by not having links between entities of the same clean source which is a default feature of the linking component of FAMER.

For the traditional AP clustering, the max-sum optimization has been implemented by a message passing algorithm [63]. The messages are exchanged between factor and variable nodes of the factor graph to reflect the mutual dependencies within an iterative process. The messages are computed differently depending on whether the recipient node is a variable node or a factor node. Figure 5.3b shows the messages exchanged between the nodes of the new factor graph of MSCD-AP. The grey-colored factor nodes enforce the g and h constraints while the new factor node t (marked in yellow) applies the clean-source consistency constraint via the θ and γ messages.

We build on the formulae from [64] to update messages for the original constraints and specify the new message formulas for our MSCD extension. In the max-sum algorithm, outgoing messages of a variable node summarize all incoming messages to that node, except of the node to which the new message will be sent. Due to the new constraint, all outgoing messages from variable nodes to factor nodes are now modified because the new factor nodes t_{Qj} are additional neighbours of b_{ij}. As sum of the incoming messages from the neighbouring nodes, except of the recipient, the modified messages β and ρ as well as the new message γ are easily deduced as listed in Eq. (5.4) - Eq. (5.6).

The message formulas from factor nodes to variable nodes do not change in AP when a new factor node is added. Therefore, the incoming messages of α (Eq. (5.7)) and η (Eq. (5.8)) remain unchanged compared to AP. The new incoming message θ from the new factor node t_{Qj} is expressed in Eq. (5.9). The variable assignments that maximize the energy function are calculated by Eq. (5.10).

$$\beta_{ij} = s_{ij} + \alpha_{ij} + \theta_{ij} \quad (5.4) \qquad \rho_{ij} = s_{ij} + \eta_{ij} + \theta_{ij} \quad (5.5) \qquad \gamma_{ij} = s_{ij} + \alpha_{ij} + \eta_{ij} \quad (5.6)$$

$$\alpha_{ij} = \begin{cases} \sum_{k \neq j} \max(0, \rho_{kj}) & i = j \\ \min[0, \rho_{jj} + \sum_{k \neq \{i,j\}} \max(0, \rho_{kj})] & i \neq j \end{cases} \quad (5.7)$$

$$\eta_{ij} = -\max_{k \neq j} \beta_{ik} \quad (5.8) \qquad \theta_{ij} = min(0, -\max_{k \neq i}[\gamma_{kj}]) \quad (5.9)$$

$$b_{ij} = \begin{cases} 1 & \alpha_{ij} + \rho_{ij} > 0 \\ 0 & \alpha_{ij} + \rho_{ij} \leq 0 \end{cases} \quad (5.10)$$

Eq. (5.11)-Eq. (5.23) explain the derivation of the message θ (Eq. (5.9)) from the max-sum algorithm. In MSCD-AP, θ is a message from a factor node to a variable node. Thus, θ is derived from Eq. (5.11) of the max-sum algorithm [64, 95].

$$\mu_{f \to x}(x) = \max_{n(f) \setminus \{x\}} \left(lnf(x, y_1, ..., y_m) + \sum_{y_i \in n(f) \setminus \{x\}} \mu_{y_i \to f}(y_i) \right) \quad (5.11)$$

The binary variable b_{ij} either obtains the value 1 or 0. Firstly, we investigate both cases by considering *all possible configurations* of all neighboring variable nodes $b_{kj}(k \neq i)$ of t_{Qj} and then according to Eq. (5.12) [64], we combine them in order to to get a scalar value for the θ message.

$$\mu_{ij} = \mu_{ij}(1) - \mu_{ij}(0) \quad (5.12)$$

For $b_{ij} = 1$: Eq. (5.13) shows θ for the case that i chooses j as its exemplar. All neighbors of t_{Qj} are from the same clean source Q. Let q be the number of entities in Q. All incoming messages $\mu_{b_{kj} \to t_{Qj}}(b_{kj})$ of t_{Qj} are defined as $\gamma_{kj}(b_{kj})$. In order to not violating the *clean source constraint*, no other datapoint in Q is allowed to choose j as its exemplar. Therefore, all other neighboring variable nodes $b_{kj}(k \neq i)$ of t_{Qj} are set to 0. This is the only configuration that satisfies the clean source constraint and thus the optimal one. According to Eq. (5.2), the t_{Qj} function evaluates its maximum value of 0.

$$\begin{aligned} \theta_{ij}(1) &= \max_{b_{kj}, k \neq i} [ln\, t_{Qj}(b_{1j} = 0, ..., b_{ij} = 1, ..., b_{qj} = 0) + \sum_{b_{kj}, k \neq i} \gamma_{kj}(b_{kj} = 0)] \\ &= \sum_{k \neq i} \gamma_{kj}(0) \end{aligned} \tag{5.13}$$

For $b_{ij} = 0$: There is more flexibility for finding the optimal solution if datapoint i does not choose j as its exemplar. In order to guarantee the clean source consistency, utmost one of the b_{kj} variables is allowed to be set to 1. There are q possible solutions that satisfy the clean source constraint: $q - 1$ for each b_{kj} being set to 1 and one for all b_{kj} variables being set to 0. Let the case when all b_{kj} are set to 0 be x (Eq. (5.14)) and the case when exactly one of the b_{kj} is set to 1 be y (Eq. (5.15)). The message for $b_{ij} = 0$ in Eq. (5.16) is the maximum of the two cases x and y.

$$x = 0 + \sum_{k \neq i} \gamma_{kj}(0) \tag{5.14}$$

$$y = \max_{k \neq i} [0 + \gamma_{kj}(1) + \sum_{p \notin \{k,i\}} \gamma_{pj}(0)] \tag{5.15}$$

$$\begin{aligned} \theta_{ij}(0) &= \max_{b_{kj}, k \neq i} [ln\, t_{Qj}(b_{1j}, ..., b_{ij} = 0, ..., b_{qj}) + \sum_{b_{kj}, k \neq i} \gamma_{kj}(b_{kj})] \\ &= \max(x, y) \end{aligned} \tag{5.16}$$

$\theta_{ij}(1)$ and $\theta_{ij}(0)$ combined: In Eq. (5.17) - 5.23, we bring both formulas for the cases $b_{ij} = 0$ and $b_{ij} = 1$ together. According to Eq. (5.12), the scalar message is the difference of the message values for the two settings of the binary variable.

Eq. (5.20) is transformed to Eq. (5.21) by the transformation $a - max(b_0, b_1, ..., b_n) = -max(b_0 - a, b_1 - a, ..., b_n - a)$. Subtracting the two sums in Eq. (5.20), only $-\gamma_{kj}(0)$ is left (Eq. (5.22)) and then Eq. (5.22) is transformed to Eq. (5.23), according to Eq. (5.12).

$$\theta_{ij} = \theta_{ij}(1) - \theta_{ij}(0) \tag{5.17}$$
$$= x - max(x, y) \tag{5.18}$$
$$= min(0, x - y) \tag{5.19}$$
$$= min(0, \sum_{k \neq i} \gamma_{kj}(0) - max_{k \neq i}[\gamma_{kj}(1) + \sum_{p \notin \{k,i\}} \gamma_{pj}(0)]) \tag{5.20}$$
$$= min(0, -max_{k \neq i}[\gamma_{kj}(1) + \sum_{p \notin \{k,i\}} \gamma_{pj}(0) - \sum_{k \neq i} \gamma_{kj}(0)]) \tag{5.21}$$
$$= min(0, -max_{k \neq i}[\gamma_{kj}(1) - \gamma_{kj}(0)]) \tag{5.22}$$
$$= min(0, -max_{k \neq i}[\gamma_{kj}]) \tag{5.23}$$

The pseudo code of the MSCD-AP is listed in Algorithm 6. The inputs of the algorithm are the similarity matrix S, the source information ($srcInfo$) specifying the clean sources, the damping factor (λ), two preference values for datapoints of dirty (p_{dirty}) and clean (p_{clean}) sources, the $noiseLevel$ specifying the decimal position of the similarity values where random Gaussian noise is added, and adaptation steps for preference values ($step_{pref}$) and damping factor ($step_{dmp}$). The adaptation steps are real values in (0,1] that are used to increase the original values towards the maximum 1 or decrease them towards 0. The output of the algorithm is the binary matrix $\mathbb{B}$ with the exemplar assignment of every entity.

After the initialization of the messages and output matrix (line 2 and line 3) the diagonal elements s_{jj} of the similarity matrix are set to the defined preference values and

Algorithm 6: MSCD-AP

Data: S, $srcInfo$, λ, p_{dirty}, p_{clean}, $noiseLevel$, $step_{pref}$, $step_{dmp}$
Result: $\mathbb{B}$ with exemplar assignments

```
1  repeat
2      initializeMessages();
3      initializeB();
4      modifyS(p_dirty, p_clean, noiseLevel, srcInfo);
5      for iteration = 0 : max do
6          updateMessages(λ);
7          updateB();
8          if isConverged() then break;
9      soultionFound ← isSolutionFound(B);
10     if ¬soultionFound then adaptParameters(step_pref, step_dmp);
11 until soultionFound;
```

the noise is added to all similarity values in line 4. The iterative message passing starts in line 5. In each iteration, the messages are updated in line 6 according to Eq. (5.4) to Eq. (5.8). Additionally, α and ρ messages are damped in order to prevent oscillations. Finally in line 7, the binary matrix values are updated according to Eq. (5.10). If no changes are observed in the binary matrix after a specific number of iterations, the algorithm converges and is ended (line 8). Otherwise it ends after a maximal number of iterations. If the algorithm stops but the solution is not found yet (line 9 and line 10), then it has to be restarted with adapted parameters. For this purpose, function `adaptParameters` initially decreases the preference values by *preference adaption step* ($step_{pref}$) until the minimum value 0. If convergence is still not reached, the preference values are then increased step by step until the maximum 1 is reached. In case of no success, the preference values are reset to their original values and the damping factor λ is now increased by *damping adaption step* ($step_{dmp}$). These process continues until the algorithm finds a valid solution.

5.2.3 Scalable MSCD Affinity Propagation

Clustering large datasets is a challenge for AP since its time and memory complexity grows quadratically with the number of entities and thus the data volume[3]. Liu et al. [106] proposed Hierarchical Affinity Propagation (HAP) to make AP suitable for clustering large-scale datasets. Following a divide and conquer strategy, HAP clusters the dataset by executing AP several times on different levels of data.

Figure 5.5 illustrates the hierarchical clustering for three levels. In the first (the lowest) hierarchy level, the dataset is randomly divided into equal-sized partitions of maximal size M. Then AP is executed on each partition, resulting into a set of so called *local*

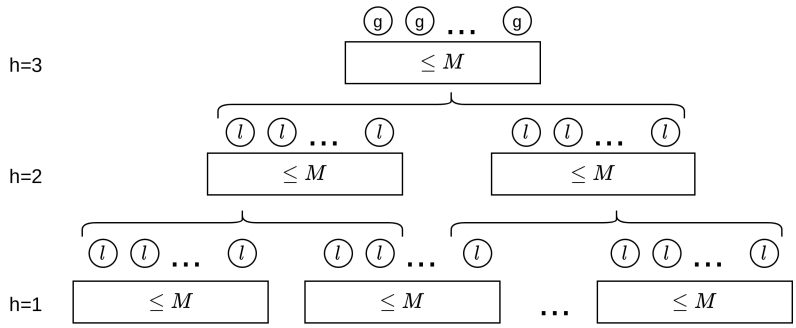


Figure 5.5: HAP for three hierarchy levels h (l: local exemplar, g: global exemplar)

[3]In the case of a sparse similarity matrix, the time complexity reduces to $Nklog(N)$ with k being the average connectivity of the similarity matrix [187].

exemplars for each partition. In the next hierarchy level, the exemplars of the previous level are merged and again partitioned. This process is repeated until the input size of a hierarchy level is lower or equal to M. The execution of AP on the top hierarchy level determines the *global exemplars* for the dataset. All non-exemplar entities are assigned to the global exemplar with the highest similarity. Thus, AP is executed once for each partition of each hierarchy level with a complexity of $O(M^2)$.

Unfortunately, applying the hierarchical algorithm for MSCD-AP does not guarantee the clean-source consistency. This is because, the clustering of local exemplars by MSCD-AP on intermediate hierarchy levels violates the clean-source constraint when two local exemplars from a previous level are clustered together although they have associated entities from the same clean source. A naive solution is to extend each local exemplar with the source information of the entities assigned to it in the previous hierarchy level. This could be used in subsequent cluster decisions to avoid that more than one entity of a clean source is assigned to an exemplar. This approach, however, can lead to poor clustering results. A bad decision in a lower level of the hierarchy, where an entity of a clean source with a low similarity is assigned to a local exemplar, can prevent that a much more similar entity from the respective source is merged at a higher level resulting in poor cluster decisions.

A more promising solution is to assign entities to global exemplars separately for clean and dirty sources. Initially, HAP is executed using MSCD-AP to determine local and global exemplars on the partitions. As in HAP, dirty source entities are then assigned to the exemplars with the highest similarity. By contrast, clean source entities are assigned using the Hungarian algorithm [96, 114]. Given the similarities between these entities and exemplars, the Hungarian algorithm finds a 1:1 assignment between entities of a clean source and exemplars (i.e., each exemplar is assigned to at most one entity of a clean source) so that the overall similarity of all assignments is maximized. If the number of entities from a clean source exceeds the number of exemplars, the excess points form singleton clusters. When a global exemplar is from a clean source, the clean-source consistency is also enforced since there is no similarity link between entities of the same clean source, i.e., only entities from dirty sources can be assigned to such an exemplar.

The Hungarian algorithm has a computational complexity of $\mathcal{O}(mk^2)$ for a $m \times k$ cost matrix [35] with k global exemplars and m entities from one clean source. The complexity is higher compared to AP, but the bipartite matching is executed on small subsets of the dataset ($m, k \ll n$). Thus, the combination of HAP with MSCD-AP and the Hungarian algorithm is still more suitable for large datasets than MSCD-AP. We call this combination MSCD-HAP and comparatively evaluate it in the next section.

5.3 Hierarchical Clustering for Multi-source Clean/Dirty Datasets

In this section we briefly explain the basic Hierarchical Cluster Analysis (HCA) [176] and then present the Multi-Source clean/Dirty Hierarchical Agglomerative Clustering (MSCD-HAC) and its scalable version in detail.

5.3.1 Hierarchical Cluster Analysis

Hierarchical Cluster Analysis (HCA) [176] comprises clustering algorithms that pursue building a hierarchy of clusters where a higher-level cluster combines two clusters of the level and this construction principle is recursively applied leading to a hierarchy of clusters. The hierarchies can be formed in a bottom-up or top-down manner. The bottom-up approach known as *agglomerative* merges the two most similar clusters as one cluster that is moved up the hierarchy. In contrast, the top-down approach is *divisive* and initially assumes all entities build a single cluster. Then, it performs splitting this cluster into two clusters in a recursive manner. Each splitted cluster moves one step down the hierarchy [152].

The results of hierarchical clustering form a binary tree that can be visualized as a dendrogram [128]. The decision on merging (in agglomerative approach) or splitting (in divisive approach) is based on a greedy strategy [116]. Due to the fact that there are 2^n possibilities for splitting a set of n entities, the divisive approach is not usually feasible for practical applications [85]. Therefore, we focus on Hierarchical Agglomerative Clustering (HAC).

The agglomerative approach is listed in Algorithm 7 [103]. The algorithm initially assumes each entity as a cluster (line 1) and it then selects and merges the two most similar clusters as one cluster (line 3-line 4). The process of selecting and merging continues in an iterative way until a stopping condition is satisfied (line 5). The hierarchical clustering scheme may lead into totally different clustering results depending on the approach

Algorithm 7: Hierarchical Agglomerative Clustering

```
1 Initialize each entity as a cluster
2 do
3 |   Select the best two clusters to merge
4 |   Merge selected clusters into one cluster
5 while stopping condition is satisfied;
```

to determine the similarity between clusters and depending on the stopping condition [85]. The rule that determines the most similar clusters is known as *linkage strategy*. There is a wide range of linkage strategies for computing the similarity of two clusters (inter-cluster similarity), each of them showing a different impact on the final clustering result [85, 117].

We implement and evaluate three commonly used approaches with low computation cost. Considering two clusters c_i and c_j, the similarity of them computed by different linkage strategies is defined as follows: [85]

S-LINK (single-linkage) is referred to as the nearest-neighbor strategy. It determines the cluster similarity based on the two closest entities from each cluster, i.e., considering the maximal similarity between members of the two clusters. The single linkage implies that $Sim_{c_i,c_j} = max\{sim(e_m, e_n)\}$ where $e_m \in c_i$ and $e_n \in c_j$. This is an optimistic approach that ignores that there may be dissimilar members in the two clusters which might help to improve recall at the expense of precision.

C-LINK (complete-linkage) is known as the furthest-neighbor strategy. The two most dissimilar entities of two cluster determine the inter-cluster similarity, i.e. based on the minimum similarity between members of the two clusters. The complete linkage implies that $Sim_{c_i,c_j} = min\{sim(e_m, e_n)\}$ where $e_m \in c_i$ and $e_n \in c_j$. This is a conservative or pessimistic approach that might help to improve precision at the expense of recall.

A-LINK (average-linkage) defines the cluster similarity as the average similarity of the entities of two clusters: $Sim_{c_i,c_j} = \frac{1}{|c_i| \cdot |c_j|} \sum_{e_m \in c_i, e_n \in c_j} sim(e_m, e_n)$.

The application of HAC results in a set of clusterings, one at each level of the cluster

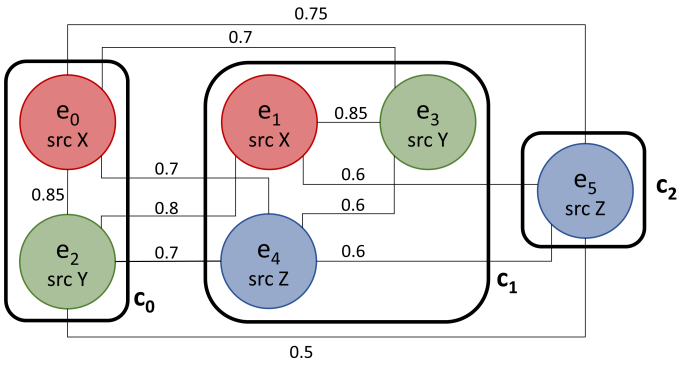


Figure 5.6: Hierarchical clustering example

Table 5.1: Linkage types

Cluster pair	S-LINK	C-LINK	A-LINK
c_0, c_1	**0.80**	0.00	0.48
c_0, c_2	0.75	0.50	**0.62**
c_1, c_2	0.60	**0.60**	0.40

hierarchy. Determining the optimal clustering from the hierarchy is not a trivial decision with large datasets. Therefore, metrics such as number of clusters or a minimum merge threshold are used as the stopping criteria. Due to the fact that the number of output clusters are not predefined in ER applications, we use the a *merge threshold* as stopping condition. Hence, the algorithms stops as soon as there is no further pair of cluster whose similarity is exceeding the merge threshold.

Figure 5.6 shows an example of three clusters along with the similarities between entities (from the similarity graph). Table 5.1 lists the inter-cluster similarity of all possible cluster pairs for our three linkages types. For S-LINK (first column), the most similar cluster pair is $\{c_0,\ c_1\}$ because the maximum link between these clusters has the highest similarity compared with the two other cluster pairs. For C-LINK (second column) we have cluster similarity 0 for $\{c_0,\ c_1\}$ due to the missing similarity links for cluster members. Thus, c_1 and c_2 with inter-cluster similarity 0.6 are the most similar clusters. For A-LINK, the cluster pair $\{c_0,\ c_2\}$ has the highest average similarity. Hence, we have different merge decisions for each of the three strategies.

5.3.2 MSCD Hierarchical Agglomerative Clustering

Performing ER for a mixed collection of clean and dirty data sources requires to determine source-consistent clusters as the final output. Therefore, the ER pipeline should take clean sources into account in both linking and clustering phases. Hence, the linking phase does not create similarity links between entities of the same clean source. However the indirect connections (transitive closures) can still lead to source-inconsistent clusters. To address this issue, we propose an extension to Hierarchical Agglomerative clustering called Multi-Source Clean/Dirty HAC (MSCD-HAC). The proposed algorithm aims at clustering datasets of combined clean and dirty sources. Our extension to HAC introduces the following contributions:

1. When picking the most similar cluster pair, the algorithm checks whether merging them would lead to a source inconsistent cluster. Such pairs are ignored to

ensure that only source-consistent clusters are determined. If the source consistency constraint is not satisfied, then the pair is removed from the candidate pairs set and thus the algorithm skips computing the inter-cluster similarity of them. For our running example (Figure 5.6), merging clusters c_0 and c_1 for all linking strategies would thus be forbidden under the assumption that X and Y are clean sources.

2. When there are several clean sources, the algorithm can remove weak inter-links of clean sources in order to improve output quality. When this option is chosen (by a parameter), a similarity graph with removed weak links is processed for clustering. For the example of Figure 5.6, ignoring the weak link between entities e_1 and e_2 would decrease the maximal similarity between clusters c_0 and c_1 from 0.8 to 0.7. Therefore, S-LINK does not decide on merging them.

The pseudo code of MSCD-HAC is shown in Algorithm 8. The input of the algorithm is a similarity graph $\mathcal{G}$ in which the vertices $\mathcal{V}$ represent entities and each edge in the edges $\mathcal{E}$ connects two similar entities and stores the similarity value of them. Further input parameters are the stopping merge threshold $\mathcal{T}$, linkage strategy, weak link strategy $weakFlag$, and the set of clean sources $\mathcal{S}$. The algorithm guarantees to create a set of source-consistent clusters $\mathcal{CS}$ as output. If the weak link strategy is selected, weak links are removed prior to performing the clustering process (line 1-line 2). As for the basic Hierarchical Agglomerative clustering, the algorithm first initializes the output cluster set $\mathcal{CS}$ by assuming each entity as a cluster (line 4). Then, it iterates over all cluster pairs in $\mathcal{CS}$ (line 8). If merging a cluster pair would lead to a source-consistent cluster (line 9), the inter-cluster similarity of the pair is computed using the linkage method in line 10. The pair with the maximum similarity is considered as candidate pair for merging (line 11-line 14) and if the similarity of candidate pair (sim_{max}) is higher than $\mathcal{T}$, then the clusters of the candidate pair are merged and the cluster set is updated (line 17-line 20). The iterative algorithm terminates when sim_{max} is lower than the minimum threshold $\mathcal{T}$ (line 21).

5.3.3 Scalable MSCD Affinity Propagation

We initially define the concept of *Reciprocal Nearest Neighbour (RNN)* in order to describe the parallel vaariation of MSCD-HAC.

Reciprocal Nearest Neighbour (RNN): If entity e_i is the nearest neighbour of entity e_j ($NN(e_j) = e_i$) and vice versa ($NN(e_i) = e_j$), then e_i and e_j are reciprocal nearest neighbours. In [160] such links have been called strong links.

Algorithm 8: MSCD-HAC

Input $\mathcal{G}(\mathcal{V}, \mathcal{E})$, $\mathcal{T}$, $linkage$, $weakFlag$, $\mathcal{S}$
⋮
Output: Cluster Set $\mathcal{CS}$

```
1  if weakFlag then
2  |  G(V, E') ← removeWeakLinks(G(V, E), S)
3  end
4  CS ← initializeClusters(V)
5  do
6  |  sim_max ← 0
7  |  candidatePair ← {}
8  |  foreach c_i, c_j ∈ CS do
9  |  |  if isSourceConsistent(c_i, c_j, S) then
10 |  |  |  sim ← computeSim(c_i, c_j, linkage)
11 |  |  |  if sim > sim_max then
12 |  |  |  |  sim_max ← sim
13 |  |  |  |  candidatePair ← c_i, c_j
14 |  |  |  end
15 |  |  end
16 |  end
17 |  if sim_max > T then
18 |  |  merge(candidatePair)
19 |  |  CS ← update(CS)
20 |  end
21 while sim_max > T;
```

For parallelizing our approaches, we follow the concept of Reciprocal Nearest Neighbour (RNN) which has been used for parallel graph clustering algorithms including HAC [116] and Center clustering [158]. If two clusters are both at the same time the nearest neighbor of each other, it means they are the two most similar clusters that can be merged with each other. This is utilized in our parallel MSCD-HAC algorithm shown in Algorithm 9. The input and output are the same as for the sequential MSCD-HAC (Algorithm 8). Similar to the sequential algorithm in line 1-line 3 weak links are optionally removed and cluster set initialization is done (line 4). Then, for each cluster c_i in the cluster set $\mathcal{CS}$ the nearest neighbour which satisfies the source consistency constraint is determined (line 7-line 8). If any source-consistent nearest neighbour c_j is found (line 9) and the nearest neighbour of c_j is c_i, then c_i and c_j are assumed as RNN (line 10) and thus will be merged and the $\mathcal{CS}$ is updated (lines line 11-line 12). Any occurring merge represents a change in the cluster set $\mathcal{CS}$ which sets the $isChanged$ flag as $true$ (line 13). The iterative algorithms terminates when no change is possible in the $\mathcal{CS}$ (line 17).

Algorithm 9: Parallel MSCD-HAC

Input $\mathcal{G}(\mathcal{V}, \mathcal{E}), \mathcal{T}, linkage, weakFlag, \mathcal{S}$
⋮
Output: Cluster Set $\mathcal{CS}$

```
1  if weakFlag then
2  |  G(V, E') ← removeWeakLinks(G(V, E), S)
3  end
4  CS ← initializeClusters(V)
5  do
6  |  isChanged ← false
7  |  foreach c_i ∈ CS in Parallel do
8  |  |  c_j ← findConsistentNN(c_i, T, S)
9  |  |  if c_j ≠ Null then
10 |  |  |  if findNN(c_j) = c_i then
11 |  |  |  |  merge(c_i, c_j)
12 |  |  |  |  CS ← update(CS)
13 |  |  |  |  isChanged ← true
14 |  |  |  end
15 |  |  end
16 |  end
17 while isChanged
```

The algorithm is implemented on top of Apache Flink framework using the Gelly library for parallel graph processing. Each clustered vertex stores the cluster-ID as a vertex property so that vertices with the same cluster-ID belong to the same cluster. In order to facilitate computing inter-cluster similarity and updating cluster information, each cluster is represented by a center vertex which maintains all cluster information. The center vertex is chosen randomly and stores the list of cluster members, the list of neighbour centers, the list of links to the neighbour centers, and the list of data sources of the members. We use the scatter-gather iteration processing of Gelly that provide sending messages from center vertex to any target vertex such as neighbor centers and cluster members. For merging two clusters, one cluster accepts the cluster-ID and center of the other cluster. Then, all lists of the center are updated. Each iteration of the algorithm consists of four supersteps explained as follows:

1. During the first scatter-gather step the source-consistent RNNs are found. In addition, the center status of one cluster center in each RNN is removed.

2. The old centers (vertices that lost their center status in the previous superstep) now produce the following messages to complete the cluster merge: one for each

cluster member informing it about the new cluster center and cluster-ID, one for the new center including the new cluster members, and one for each neighbor including the new centerID. In the gather step vertices that receive any message from the old center update their information. The neighbor centers can update their edges accordingly so that all edges in the edge list are only connecting center vertices.

3. Now that all edges are adjusted, the old center vertices produce messages for their new cluster centers including all their neighbors and corresponding link values.

4. In the last step the nearest neighbor vertices are recalculated. If the similarity to the nearest neighbour is less than the stopping threshold, the vertex will not produce any messages during the following phase. Thus, after each round of four iterations the number of active vertices as well as the number of clusters decreases.

5.4 Evaluation Results

This section presents the cluster effectiveness and efficiency of the proposed MSCD extensions of AP in comparison to standard AP and previous clustering schemes. Firstly, the used datasets from four domains is described and then the effectiveness of the proposed algorithm is comparatively analyzed. Finally, the evaluation of runtime performance and scalability are presented.

5.4.1 Datasets

We evaluate the new approaches with four datasets of clean sources that have also been used in previous studies [157, 158, 160]. Table 5.2 gives an overview of the datasets from four domains (geography, camera, music, persons) including available properties and number of entities. For the evaluation of mixed datasets of clean and dirty sources, we use the dataset of the ACM SIGMOD 2020 Programming Contest[4]. It contains approximately 30k product specifications from 24 dirty sources. For our purposes, we determine a subset called DS-C focusing on camera products and excluding source *www.alibaba.com*[5]. Table 5.3 lists the 23 remaining sources and their number of entities with and without duplicates. The matching result of the SIGMOD contest winner is considered as the

[4] http://www.inf.uniroma3.it/db/sigmod2020contest/index.html
[5] The source mostly contains non-camera entities.

Table 5.2: Overview of evaluation datasets

	General information				Perfect result	
	domain	entity properties	#entity	#src	#clusters	#links
DS-G	geography	label, longitude, latitude	3,054	4	820	4,391
DS-M	music	artist, title, album, year, length	19,375	5	10,000	16,250
DS-C	camera	heterogeneous key-value pairs	21,023	23	3,910	368,546
DS-P1	persons	name, surname, suburb, postcode	5,000,000	5	3,500,840	3,331,384
DS-P2			10,000,000	10	6,625,848	14,995,973

Table 5.3: Overview of camera dataset (DS-C)

Source name	ID	#entity	#entity dedup.
buy.net	1	358	244
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
	sum	21,023	8,123

Table 5.4: MSCD datasets

Name	%cln[1]	cln[2]	#cln[3]	#dirt[4]
DS-C0	0		0	21,023
DS-C26	26	1-6, 8-23	4,868	14,009
DS-C32	32	7	3,255	7,014
DS-C50	50	7, 18, 19,20, 22, 23	4,822	4,786
DS-C62A	62	1, 4, 6 7, 9, 11, 13, 15, 17, 19, 20	5,748	3,536
DS-C62B	62	2, 3, 5, 7, 8, 10, 12, 14, 16, 18, 21-23	5,630	3,478
DS-C80	80	1-12 14-18	6,894	1,719
DS-C100	100	1-23	8,123	0

[1] Percentage of entities from clean sources
[2] Clean source IDs
[3] Number of entities from clean sources
[4] Number of entities from dirty sources

golden truth. It achieved f-measure of 99% by extensive domain-specific preprocessing and matching camera entities against a prepared list of nearly all available cameras in the market. Our matching and clustering approaches are generic and applicable to different datasets. Our goal is not to achieve the best possible result but to enable a fair comparison of the clustering schemes based on reasonably good input similarity graphs for different datasets.

Using DS-C, we create eight datasets with different combinations of clean and dirty sources and thus different degrees of dirtiness. As shown in Table 5.4, we name the datasets according to the percentage of entities from clean sources, where DS-C0 and DS-C100 means that all entities are from dirty and clean sources, respectively. For the mixed cases, an important distinction is whether a clean or dirty version of source 7 (*www.ebay.com*) is considered because it is the largest source and contains many dupli-

Table 5.5: Linking configurations of clean multi-source datasets

	Blocking key	Similarity function
DS-G	prefixLength1 (label)	Jaro Winkler (label) & geographical distance
DS-M	prefixLength1 (album)	Trigram (title)
DS-P1/P2	prefixLength3 (surname) + prefixLength3 (name)	avg (Trigram (name) + Trigram (surname) + Trigram (postcode) + Trigram (suburb))

cates. In DS-C62A and DS-C62B, the clean form of source 7 is included, while all other sources that are clean in 62A are dirty in 62B and vice versa.

The blocking and matching configurations for the clean datasets are listed in Table 5.5 and correspond to the ones in previous studies [157, 160]. For the camera dataset, we extracted the manufacturer name, a list of model names, manufacturer part number (mpn), european article number (ean), digital and optical zoom, camera dimensions, weight, product code, sensor type, price and resolution from the heterogeneous product specifications. In order to reduce the number of comparisons, standard blocking with a combined key of manufacturer name and model number is applied. Within these blocks, all pairs with exactly the same model name, mpn or ean are classified as matches. We assign a similarity value to the matched pairs determined from a weighted average of the 3Gram similarity of string values and a numerical similarity of numerical values (within a maximal distance of 30%).

5.4.2 Quality Results

To evaluate the quality of the clustering results, we use the standard metrics precision, recall and their harmonic mean, f-measure w.r.t. the links of the perfect cluster results (last column of Table 5.2).

Quality of MSCD-AP: We compare the quality of the AP and the proposed MSCD-AP approaches with seven previous clustering schemes that are included in the FAMER system [157] including the CLIP approach tailored to clean sources and six general approaches for dirty sources (connected components, correlation clustering CCPivot, two variants of star clustering and two variants of center clustering). We also provide the quality of the input similarity graph (without clustering) in our figures. For AP and MSCD-AP we manually determined suitable parameter configurations. We use the interval [0.01, 0.7] for preference values and set a higher preference value for clean sources than for dirty sources to choose exemplars preferably from clean sources. The damping factor is set to 0.5 and noise is added to the similarity values from the third decimal place. For the smaller datasets DS-G, DS-M and DS-C, we used a partition size of 1000 while for the person datasets we apply MSCD-HAP with partition size 400 to reduce runtimes.

When the size of a connected component is smaller than the partition size, MSCD-AP is executed. Higher similarity thresholds (*sim th*) result in fewer links and smaller components that are mostly executed without partitioning. In DS-G and DS-C, partitioning is never used and thus AP and MSCD-AP are executed.

We first analyze cluster quality for the datasets with only clean sources. Figure 5.7 shows the precision, recall and f-measure results for the three datasets DS-G, DS-M and DS-P2 for different similarity thresholds to generate the input similarity graph. As expected, the f-measure results are the best for the CLIP approach tailored to ER for clean sources. However, the proposed MSCD-AP approach achieves about the same quality for two datasets (DS-G, DS-P2) and performs better than the six general clustering schemes for DS-M. It also outperforms AP in all cases. These surprisingly good results are mainly due to an excellent precision of MSCD-AP which can outweigh its comparatively low

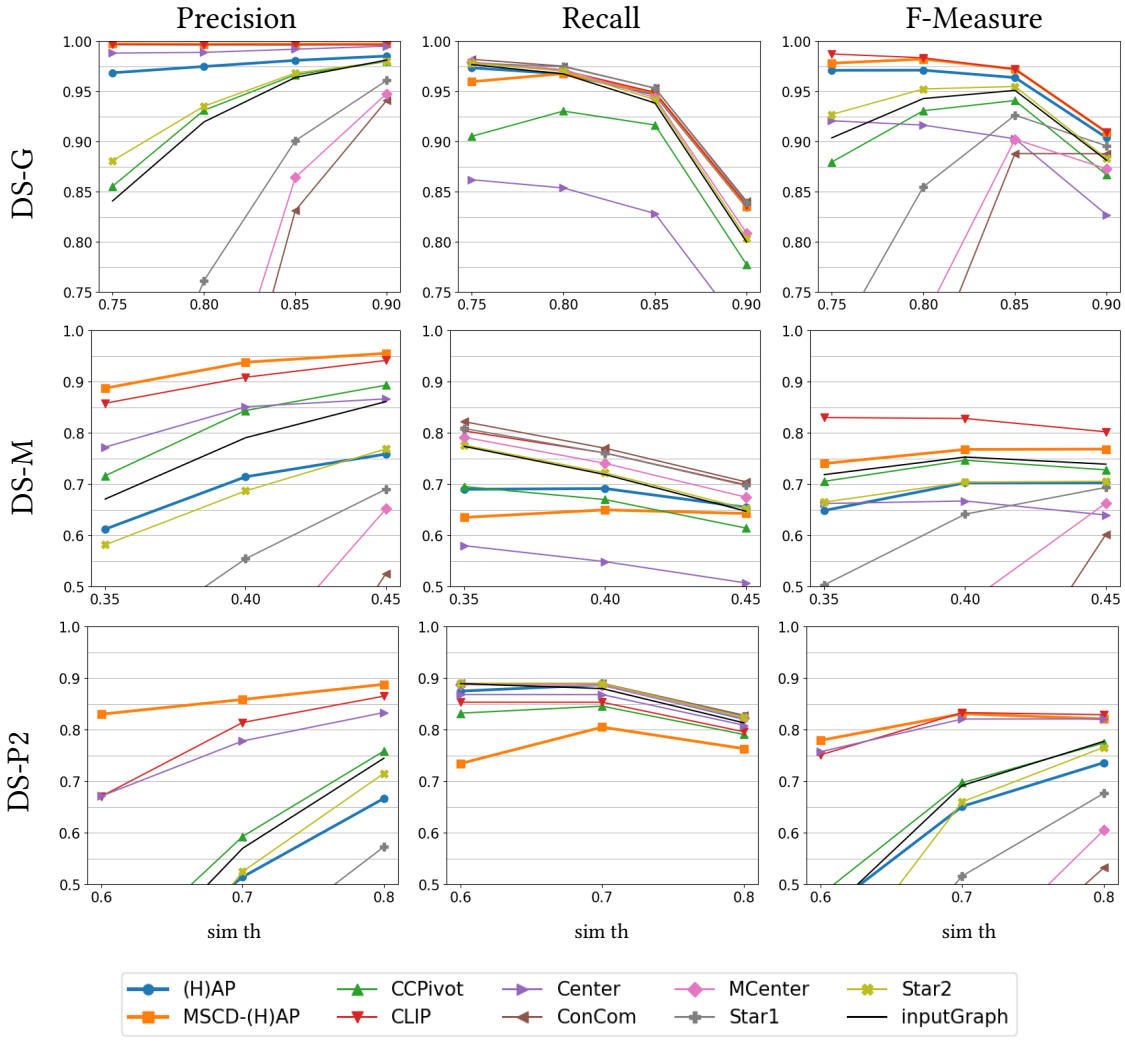


Figure 5.7: MSCD-AP evaluation for MSC datasets

recall. The recall is limited since AP and MSCD-AP strongly depend on the relative similarity values and can even consider a high similarity value such as 0.8 as low if it is below the average of the considered value range, e. g.[0.8, 1.0]. This leads to more small clusters and thus a lower recall compared to other algorithms. Due to the clean-source constraint, MSCD-AP creates more exemplars than AP and therefore obtains a lower recall compared to AP but a much better precision.

Figure 5.8 shows the quality of the clustering results for the camera datasets with different degrees of dirtiness. Due to space constraints we show results for five of the eight cases but the results for the remaining datasets confirm the overall outcome (see Appendix B). We observe that MSCD-AP achieves the best f-measure for all cases with a mix of dirty and clean sources. For the case of only clean sources (DS-C100) it is only outperformed by CLIP. For dirty sources only (DS-C0) MSCD-AP is identical to AP which is among the best approaches. As a result, MSCD-AP is the best or among the best approaches over all configurations while other schemes like CLIP are good in only one configuration. Another strong point of MSCD-AP is that its f-measure is nearly stable over all similarity values used to determine the input similarity graph while the general clustering schemes depend on finding a suitable threshold value. Like for the datasets of clean sources only, the good results of MSCD-AP are mainly due to its excellent precision in all cases that outweighs its lower recall results.

Quality of MSCD-HAC: Figure 5.9 shows the average precision and recall results for graphs with the lowest match threshold (*sim th*) for merge thresholds in [0,1). The top row shows the results for clean sources only while the lower row shows results for MSCD sources (mix of clean and dirty camera sources). We compare the basic hierarchical clustering schemes (S-Link, C-Link, etc.) with the ones applying the proposed MSCD extension. For all datasets except DC-C0 (with only dirty sources), MSCD approaches improve precision dramatically while keeping the same recall; for DS-C0, MSCD-HAC has the same results as the basic HAC. Hence, the new MSCD approaches can clearly outperform the basic HAC schemes. Ignoring weak link for the basic schemes can help to improve precision in several cases but to a much smaller degree that with MSCD. As expected, C-Link (S-LINK) achieves the highest (lowest) precision and the lowest (highest) recall for all datasets due to the use of the minimal (maximal) similarity between cluster members to determine merge candidates. A-LINK follows a more moderate strategy compared to the strict C-LINK and relaxed S-LINK strategies. In addition, applying the MSCD strategy or removing weak links improves S-LINK the most while C-LINK yields the same results as basic HAC. Due to the fact that entities of the same clean source are never directly linked to each other, C-LINK obtains source-consistent clusters as the MSCD approaches.

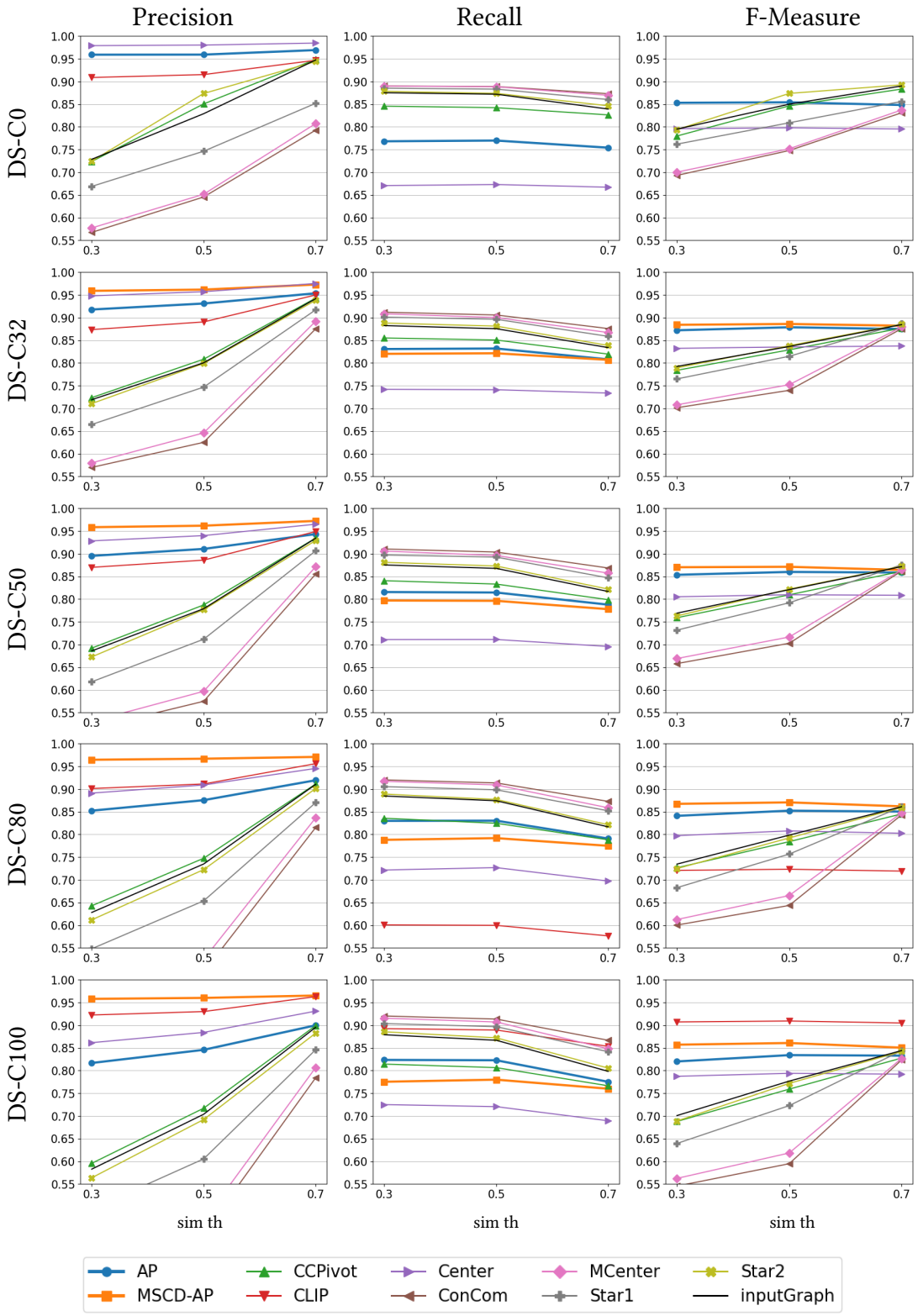


Figure 5.8: MSCD-AP evaluation for MSCD datasets

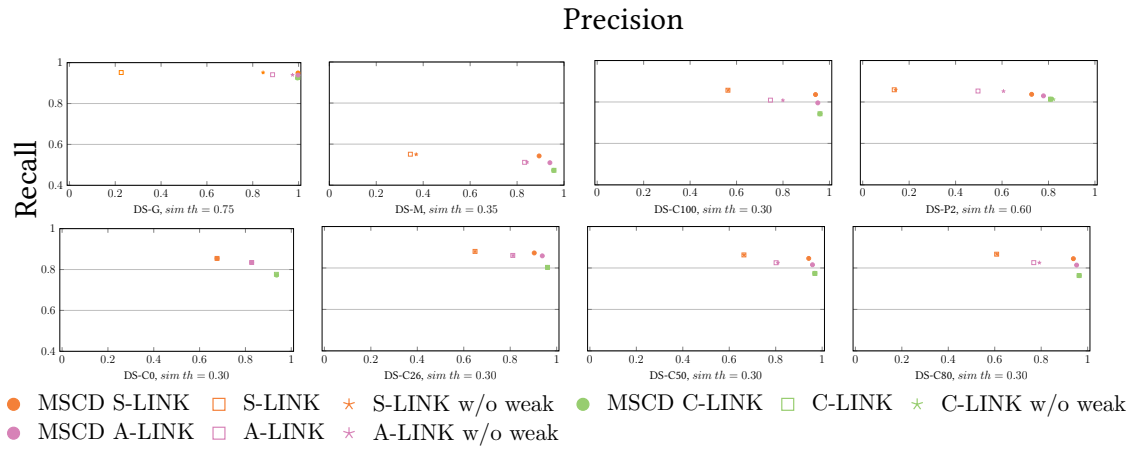


Figure 5.9: Precision/Recall for hierarchical clustering schemes.

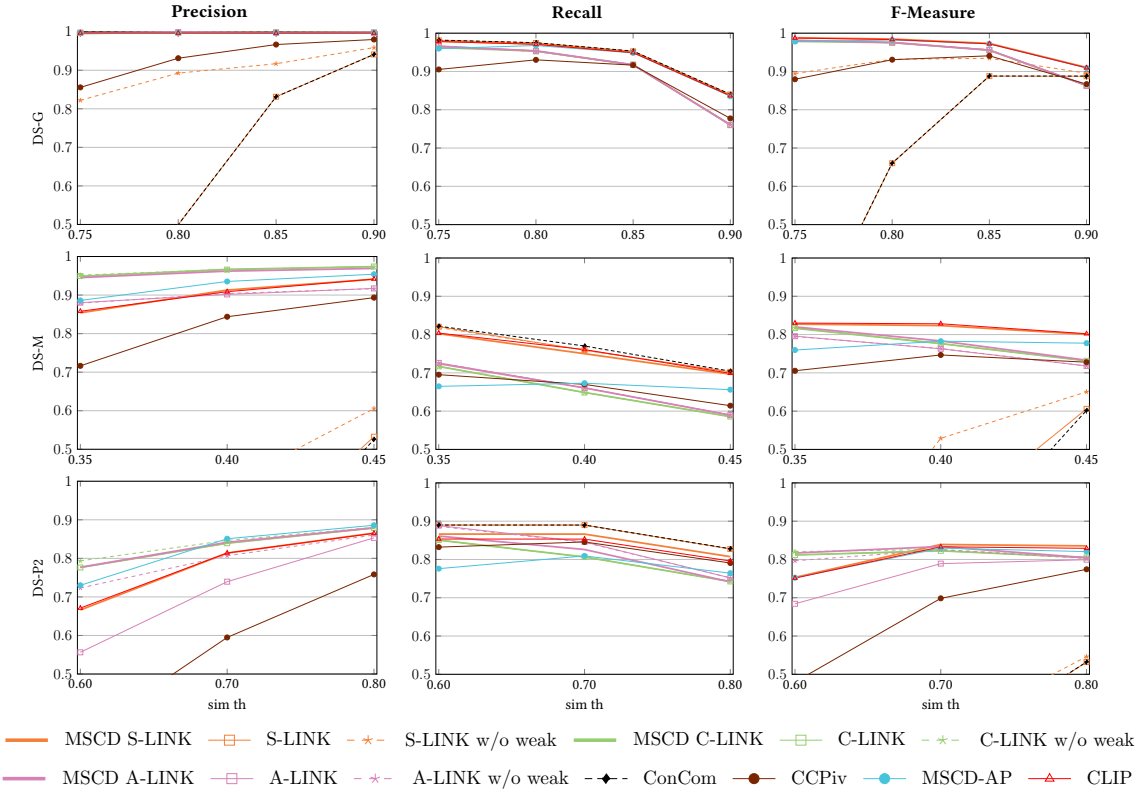


Figure 5.10: MSCD-HAC evaluation for MSC datasets

Figure 5.10 and Figure 5.11 show the results of our proposed approaches for different match (*sim th*) and merge thresholds (equal match and merge threshold) for clean (MSC) and mixed (MSCD) datasets in comparison with the baseline algorithms connected components, Correlation clustering (CCPivot variation) [26] as popular ER clustering schemes, the MSC algorithm named CLIP [160] and the MSCD-AP approach based on Affinity Propagation [102]. As expected for MSC datasets (Figure 5.10), connected

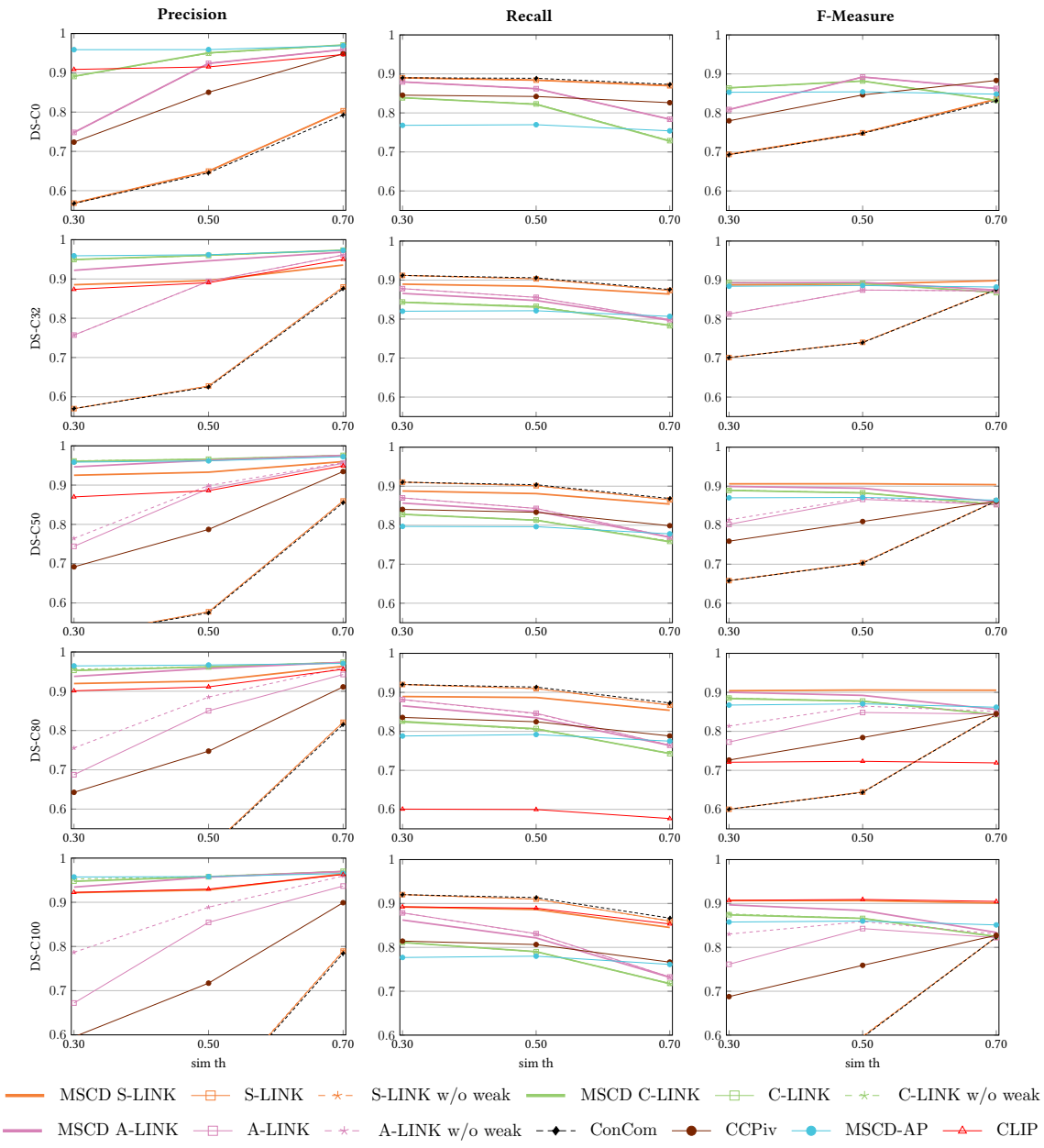


Figure 5.11: MSCD-HAC evaluation for MSCD datasets

components and S-LINK obtain the lowest precision. Removing weak links improves precision for S-LINK but it is still not sufficient to compete with the best algorithms. The C-LINK approaches and MSCD-AP achieve the best precision but at the cost of low recall. In contrast, CLIP and MSCD S-LINK obtain similarly high recall and precision. Therefore, for all datasets, MSCD S-LINK and CLIP are superior in terms of F-Measure and outperform the basic HAC approaches as well as the previous MSCD-AP approach for mixed datsets. For the bigger dataset DS-P2, CLIP and MSCD S-LINK obtain lower

precision compared to MSCD A-LINK, because they form clusters with the maximum possible size (10, one entity per source) which leads to obtaining false positives. Therefore, MSCD A-LINK surpasses MSCD S-LINK and CLIP for the low threshold 0.6.

For MSCD datasets (Figure 5.11), MSCD-HAC and HAC give the same results for the dataset with all dirty sources (DS-C0). Therefore, for DS-C0, MSCD S-LINK along with connected components obtains the lowest precision and the highest recall. As the ratio of clean sources increases MSCD S-LINK obtains better precision while keeping the recall high. Therefore, for all MSCD datasets, MSCD S-LINK obtains the best F-Measure. The algorithm CLIP yields very low F-Measure, because it is designed for clustering clean datasets. The algorithm MSCD-AP can not compete with MSCD-HAC approaches due to its lower recall (about 10% less than MSCD S-LINK). When the dataset comprises a large portion of or only dirty sources, the strict method MSCD C-LINK obtains the best results for lower thresholds. In all datasets except for DS-C0, CCPiv can not compete with the best algorithms in both terms of precision and recall. With DS-C0, CCPiv is slightly better than A-LINK due to the higher recall it achieves (see Appendix C for the evaluation of DS-P1 and the remaining camera datasets).

5.4.3 Runtimes and Speedup

We evaluate runtimes and speedup behavior for the larger datasets from the person domain. The speedup of MSCD approaches are determined for the parallel execution with different numbers of workers. The experiments are performed on a shared nothing cluster with 16 worker nodes. Each worker consists of an E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM, two 4 TB SATA disks and runs openSUSE 13.2. The nodes are connected via 1 Gigabit Ethernet. Our evaluation is based on Hadoop 2.6.0 and Flink 1.9.0. We run Apache Flink standalone with 6 threads and 40 GB memory per worker.

MSCD-HAP: Table 5.6 lists the runtime of each clustering approach for a parallel execution on 16 workers. As expected, the larger dataset DS-P2 leads to higher runtimes

Table 5.6: Runtimes for clustering schemes (seconds)

	DS-P1			DS-P2		
threshold	0.6	0.7	0.8	0.6	0.7	0.8
HAP	116	70	63	393	162	130
MSCD-HAP	564	134	93	1434	330	205
CCPivot	558	532	426	1395	1131	964
CLIP	119	90	81	362	236	195
Center	270	179	156	1089	726	603
ConCom	51	40	37	107	66	57
MCenter	417	255	210	1619	991	730
Star1	224	130	124	626	330	273
Star2	162	130	124	460	267	233

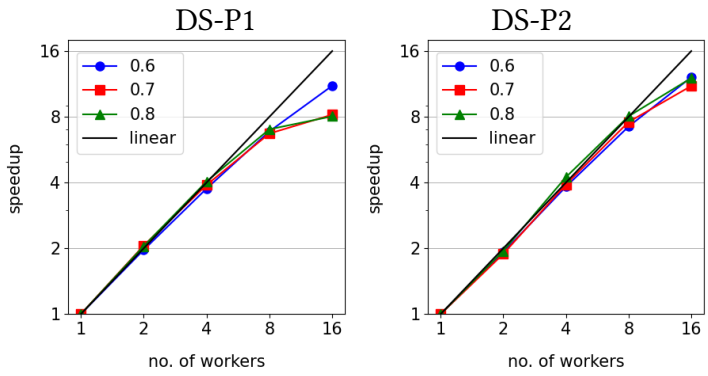


Figure 5.12: Speedup of MSCD-HAP for different similarity thresholds

than for DS-P1 while a higher similarity thresholds reduce runtimes due to the lower number of edges in the similarity graph. MSCD-HAP is slower than HAP because the calculations for the clean-source constraint and the exemplar assignment by the Hungarian algorithm need additional runtime. The clean-source constraint of MSCD-AP also leads to more exemplars and potentially more entities of the same source that are equally well suited to be an exemplar. Thus, oscillations occur more frequently for MSCD-AP compared to AP leading to more parameter adaptations to find a converging solution.

For the lowest threshold, MSCD-HAP along with CCPivot and MergeCenter are among the slowest algorithms. Yet with higher similarity thresholds the runtime of MSCD-HAP improves significantly making it one of the fastest algorithms. This is because a high minimum threshold avoids that a large number of entities are connected in the similarity graphs resulting in mostly small clusters and reduced work for the the Hungarian algorithm. Moreover, oscillations occur less in such cases.

Figure 5.12 depicts the speedup of MSCD-HAP with partition size 100 for different similarity thresholds and for 1 to 16 worker machines. We observe that close to perfect speedup is achieved for the larger dataset DS-P2 and for a lower similarity threshold (bigger similarity graph) for the smaller DS-P1 dataset. For the higher thresholds the needed computations for DS-P1 cannot utilize 16 machines so that a good speedup is only achieved until 8 workers.

Figure 5.13 investigates the effect of partition size on both runtime and clustering quality. We observe that larger partition sizes lead to much higher runtimes but also to improved clustering quality. These effects are most pronounced for smaller similarity threshold such as 0.6 that lead to bigger similarity graphs and thus to more computations. With larger partition sizes there are more entities and more similarity values in each partition. Therefore, the probability of finding good local and global exemplars rises and consequently the precision is improved. Yet recall drops slightly, because on bigger

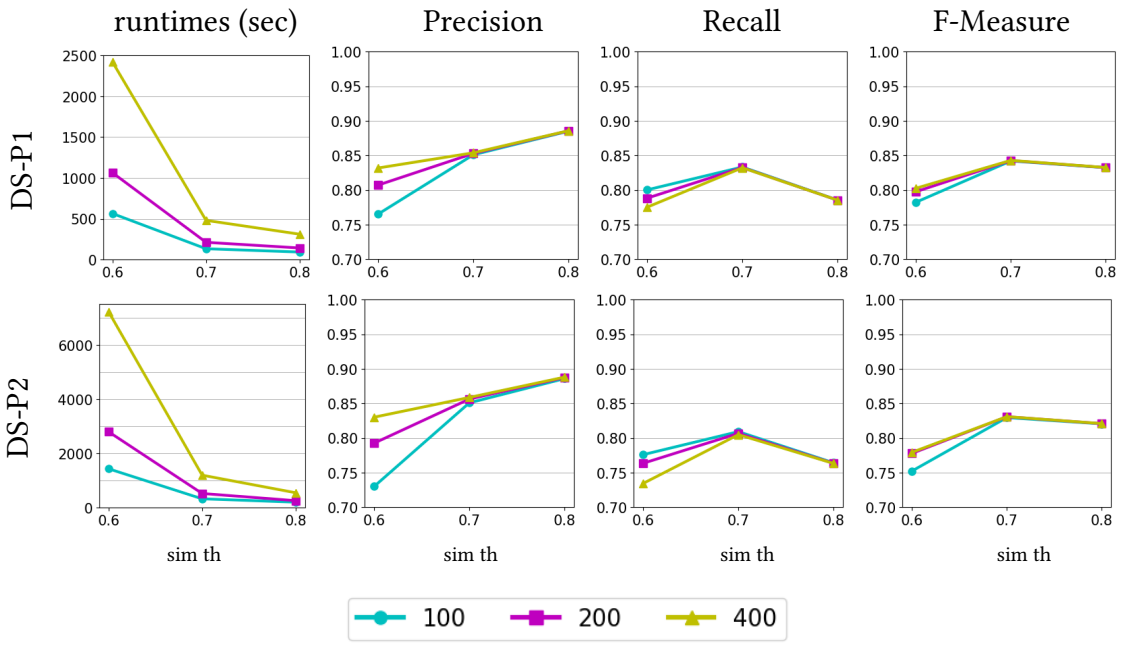


Figure 5.13: Clustering quality and runtime for different partitions sizes of MSCD-HAP

partitions more exemplars can be found and AP generally tends to form many small clusters. While the runtime is up to seven times higher for partition size 400 compared to 100 (for DS-P2) for threshold 0.6, these differences largely go away for higher thresholds and much smaller similarity graphs. This is also the case for clustering quality, where similarity value 0.7 or higher leads to about the same f-measure for all partition sizes.

MSCD-HAC: We evaluate runtimes and speedup behavior for the larger datasets from the person domain for the graph with match and merge threshold 0.8. Table 5.7 lists runtimes of all introduced approaches evaluated on 16 machines. The first row shows that S-LINK is the slowest algorithm but MSCD S-LINK improves the runtime of S-LINK dramatically. Moreover, removing weak links decreases runtime slightly. Neither applying MSCD strategy nor removing weak links improves the runtime of C-LINK and A-LINK,

Table 5.7: Runtimes (seconds)

	DS-P1			DS-P2		
	-	MSCD	NW	-	MSCD	NW
S-LINK	2256	130	2149	6818	506	6789
C-LINK	128	128	130	422	417	401
A-LINK	127	129	128	430	417	411
ConCom		37			59	
CCPiv		463			1030	
MSCD-AP		93			207	
CLIP		80			200	

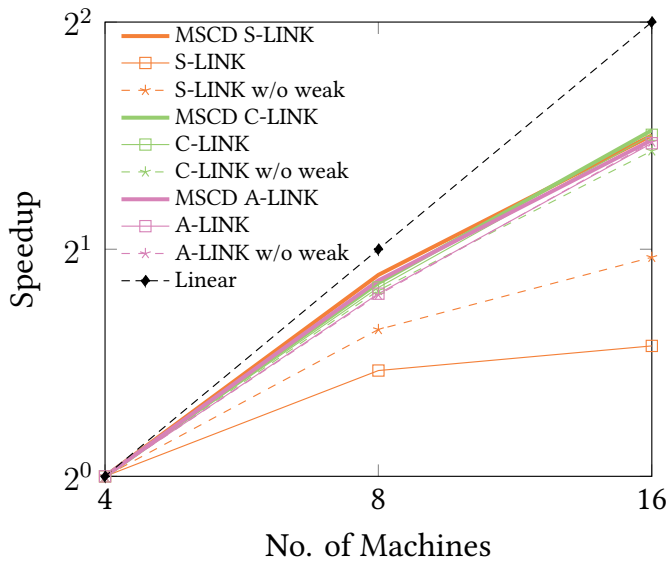


Figure 5.14: HAC speedup

because the approaches are strict enough in merging clusters. The last four rows of Table 5.7 list the runtime of approaches that are compared with HAC-based schemes. Among them connected components is the fastest approach while CLIP and MSCD-AP are 1.4x-2x faster and CCPiv is 2x-3.5x slower than HAC-based approaches. Figure 5.14 shows the speedup for DS-P1 with 5M entities. All approaches have speedup close to linear expect S-LINK. Removing weak links improves the speedup of S-LINK but it is still far from linear speedup. The approaches can not utilize 16 machines so that a good speedup is achieved until 8 workers.

5.5 Related Works

Most previous entity clustering approaches focus on finding matches in a single (dirty) source. Example approaches include Connected components, Center and Merge-Center clustering [69], Affinity Propagation [53], Ricochet clustering [179], Markov clustering [169] and Correlation clustering [7]. Hassanzadeh at al. [68] comparatively evaluated many of these algorithms for a single source. In Chapter 4 we have shown that these approaches can be adapted for multi-source entity clustering and we comparatively evaluated several approaches for such a setting. We further presented new multi-source entity clustering approaches such as CLIP [160] that work for clean (duplicate-free) data sources and can outperform the more general approaches for dirty sources. In this chapter, we compared the new MSCD entity clustering approaches based on Affinity Propa-

gation and Hierarchical Agglomerative clustering with these previous methods for dirty and clean sources.

Affinity Propagation is a constraint-based clustering algorithm that was firstly introduced by Frey et al. [53]. Hierarchical Affinity Propagation [62] gives the possibility of AP clustering in multiple layers which provides scalable solution to standard AP. Hierarchical clustering is a popular clustering approach that has also been employed for ER [108]. Moreover, collective [14] and progressive [178] entity resolution utilize hierarchical clustering for performing ER task effectively and efficiently. Collective ER determines matches based on their common neighbours while progressive ER follows the pay-as-you-go logic in order to perform ER in real-time. Recently, Yan et al. [184] propose a modified hierarchical clustering that aims at avoiding so-called hard conflicts introduced by systematically missing information in different sources. The conflicts may occur due to systematically missing information from different sources. Hierarchical clustering moreover supports results with multiple confidence levels that is utilized for determining final matches with preferable precision/recall trade-off. Many approaches try to make hierarchical clustering faster which is inherently iterative and thus sequential. Some approaches reduce the hierarchical clustering to the problem of creating the Minimum Spanning Tree (MST) [36] while others approximate the results by utilizing Locality-Sensitive Hashing (LSH) [87]. Another option considered is to partition data evenly on processing nodes before performing clustering [37, 72, 78]. Furthermore, a method based on the concept of Reciprocal Nearest Neighbors (RNN) that fits graph clustering can be applied [115, 116].

In this chapter we extend the usage of hierarchical clustering for efficient and effective clustering of entities from a combination of arbitrary portion of clean and dirty sources. We further enable the algorithm to improve the final results by removing potential false links (weak links) in a preprocessing step. To improve scalability, the parallel variant is implemented based on the RNN concept using scatter-gather iterations [81].

5.6 Conclusion

In this chapter, we studied how to support multi-source entity clustering for a mix of clean (duplicate-free) and dirty data sources. The proposed extension of Affinity Propagation and Hierarchical clustering, showed to be highly effective and perform better than previous methods for mixed configuration where a subset of the sources is duplicate-free. To improve runtimes we proposed the use of parallel versions and provided parallel im-

plementations of the algorithms. The parallel implementations achieved good speedup values thereby supporting scalability to larger datasets.

Comparing the proposed MSCD clustering algorithms, MSCD-HAC approaches always exceed MSCD-AP algorithm. On the other hand MSCD-AP results in the best precision with better speedup behavior. However, the scalable variation of MSCD-AP (MSCD-HAP), is an approximation of the original algorithm and may result in poor results compared to the original MSCD-AP. The evaluation of MSCD-HAC with different linkage types showed that MSCD S-LINK obtains superior cluster results compared to previous clustering schemes specifically for MSCD datasets with dirty sources. For the case of clean sources the same or better quality than the best methods such as CLIP is achieved. In some cases such as for larger clusters (many sources), MSCD S-LINK is outperformed by other linkage strategies. We will therefore investigate how to automatically select the best linkage strategy for MSCD clustering.

6

Incremental Entity Resolution

This chapter is based on [159]. Incremental entity resolution is needed for the completion of knowledge graphs integrating data from multiple sources. Compared to previous approaches we aim at reducing the dependency on the order in which new sources and entities are added. For this purpose, we consider sets of new entities for an optimized assignment of them to entity clusters. We also propose the use of a light-weight approach to repair entity clusters in order to correct wrong clusters. The new approaches are integrated within the FAMER framework for parallel and scalable entity clustering. A detailed evaluation of the new approaches for real-world workloads shows their high effectiveness. In particular, the repair approach outperforms other incremental approaches and achieves the same quality than with batch-like entity resolution showing that its results are independent from the order in which new entities are added. Our approaches were presented at ESWC 2020.

6.1 Motivation

Knowledge graphs (KG) physically integrate numerous entities with their properties and relationships as well as associated metadata about entity types and relationship types in a graph-like structure [146]. The KG entities are typically integrated from numerous sources, such as other knowledge graphs or web pages. The initial KG may be created from a single source (e.g., a pre-existing knowledge graph such as DBpedia) or a static integration of multiple sources. KG completion (or extension) refers to the incremental addition of new entities and entire sources. The addition of new entities requires solving

several challenging tasks, in particular an incremental entity resolution to match and cluster new entities with already known entities in the KG [131].

Most previous work on entity resolution (ER) deals with static ER to match entities from one or several static data sources. Such static approaches are not sufficient to add entities to an in-use KG where the majority of already integrated entities is largely unaffected by new entities and should not have to be re-integrated for every update. ER for entities of multiple sources typically groups or clusters matching entities and these clusters can then be used to fuse (merge) the properties of the matching entities to obtain an enriched entity description for the KG. Incremental ER thus requires to update these entity clusters for new entities. A naive approach is to simply add a new entity either to the most similar existing cluster or to create a new cluster if there is no similar one [122, 177]. However, this approach typically suffers from a strong dependency on the order in which new entities are added. In particular, wrong cluster decisions, e.g., due to data quality problems, will not be corrected and can lead to further errors when new entities are added. The overall ER quality can thus be much worse than for batch ER where all entities are simultaneously integrated.

We therefore propose and evaluate new approaches for incremental entity clustering that reduce the dependency on the order in which new entities and sources are added. The approaches have been developed for the framework FAMER that supports a parallel ER for entities from multiple sources [160]. As described in Chapter 3, for batch ER, FAMER first applies pairwise linking among entities and derives a so-called similarity graph. This graph is input for entity clustering that determines a set of clusters where each cluster groups the matching entities from several sources. These linking and clustering steps now need to become incremental while preserving a similarly high quality than for batch ER.

Specifically, the following contributions are made in this chapter:

- A proposal of several approaches for incremental linking and clustering. For an optimized cluster assignment, we consider the addition of sets of entities and so-called *max-both* assignments that add an entity to the most similar cluster only when there is no more similar new entity from the respective data source. Furthermore, we optionally can link new entities with themselves before updating entity clusters. We also support the fusion of cluster members to a single entity which simplifies and speed-ups incremental clustering as new entities need no longer be compared to several entities of a cluster.

- A proposal of a new method called *n-depth reclustering* for incremental ER that is able to repair existing clusters for improved quality and a reduced dependency on the insert order of new entities.
- The evaluation of the incremental approaches for datasets of three domains in terms of cluster quality and runtime efficiency. We also provide a comparison to a previous approach for incremental cluster repair [66] and with batch ER.
- All methods are implemented on top of Apache Flink framework for improved runtimes and high scalability to large datasets.

Section 6.2 presents the new methods in detail and Section 6.3 is the evaluation. A discussion of related work is presented in Section 6.4. Finally, we conclude in Section 6.5.

6.2 Incremental Approaches

As explained in Chapter 3 the input of the workflow is a stream of new entities from existing sources or from a new source plus the already determined clustered similarity graph from previous iterations. The *linking* part now focuses on the new entities and does not re-link among previous entities. We also support the linking among new entities to provide additional links in the similarity graph that may lead to better cluster results. The output of the linking is a *grouped similarity graph* composed of existing clusters and the group of new entities and the newly created links (the light-blue colored group in the middle of Figure 6.1).

The *Incremental Clustering/Repairing* part supports two methods for integrating the group of new entities into clusters. In the base (non-repairing) approach called *Max-Both Merge* (MBM) the new entities are either added to a similar existing cluster or they form a new cluster. A more sophisticated approach is able to repair existing clusters to achieve

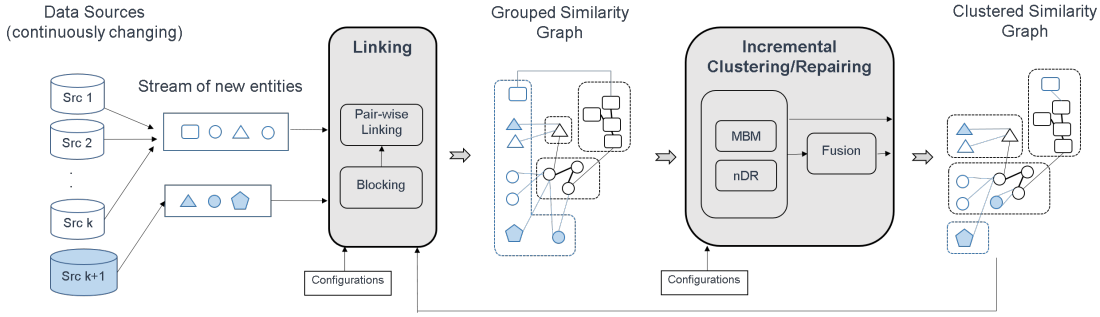


Figure 6.1: FAMER workflow for incremental entity resolution

a better cluster assignment for new entities by reclustering a portion of the existing clustered graph. The method is named *n-depth reclustering* (nDR) where n is a parameter to control the portion of the similarity graph that is considered for reclustering.

The output of incremental clustering is a fully clustered graph. The clusters can optionally be fused in the *Fusion* component so that all entities are represented by a single entity called cluster representative. Fusion can improve linking efficiency since new entities only have to be compared with the cluster representatives instead of all cluster members. On the other hand, we loose the possibility to recluster if we retain only a single fused entity per cluster.

In the rest of this chapter, we first define the main concepts in Section 6.2.1. We then describe the general incremental ER process in Section 6.2.2 and the base approach MB in Section 6.2.3. Finally, the repairing method is described in Section 6.2.4.

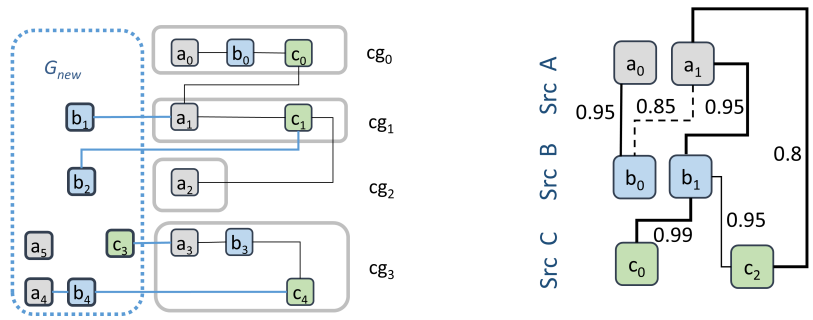
6.2.1 Concepts

Grouped similarity graph: A grouped similarity graph $\mathcal{GG}$ is a similarity graph where each entity can be associated to a group or cluster. Clustered entities have a cluster-id of the cluster they belong to. The grouped similarity graph allows us to maintain already determined clusters together with the underlying similarity graph as input for incremental changes such as adding new entities. A grouped similarity graph may also include new entities with their similarity links to other entities. Figure 6.2a shows a grouped similarity graph with four groups cg_0, cg_1, cg_2, cg_3 and group g_{new} with new entities. There are links between entities of the same group, so-called *intra-links*, as well as links between entities of different groups (*inter-links*) resulting in group neighborhoods.

Fused similarity graph: A fused similarity graph is a clustered similarity graph in that each cluster is only represented with a cluster representative. The cluster representative combines the property values of the original cluster members and also records the ids of the originating data sources as provenance information (see sample cluster representatives in Figure 6.5a).

Max-Both link: An entity from a source A may have several links to entities of a source B. From these links, the one with the highest similarity value is called maximum link. If a link is a maximum link from both sides, it is a max-both or strong link. In Figure 6.2b, for entity a_1 the maximum link to source B is the one to entity b_1 (similarity 0.95). This link is also maximum for b_1 so that it is a max-both link. By contrast, the link between c_2 and b_1 is only the maximum link for one side (c_2) and the link between a_1 to b_0 for none of the sides.

n-depth neighbor graph: If a group in a grouped similarity graph is linked to the



(a) Sample grouped similarity graph

(b) Max-Both concept

Figure 6.2: Incremental clustering concepts

other groups via inter-links, the graphs directly linked to it are called 1-depth neighbor graphs. Recursively, the 1-depth neighbors of the n-depth neighbors are the (n+1)-depth neighbors. For example in Figure 6.2a, G_{new} is the 1-depth neighbor of cg_1 and cg_3 and also 2-depth neighbor of cg_0 and cg_2.

6.2.2 Incremental Entity Resolution

Incremental ER limits linking and clustering to the new entities rather than processing all entities as for batch ER. At the same time the resulting linkage and cluster quality should be similar to batch ER which means that the order in which entities are added should ideally have no impact on quality. The latter requirement is a main reason for re-clustering as otherwise wrong cluster decisions can impact further cluster decisions and thus lead to increasing quality problems.

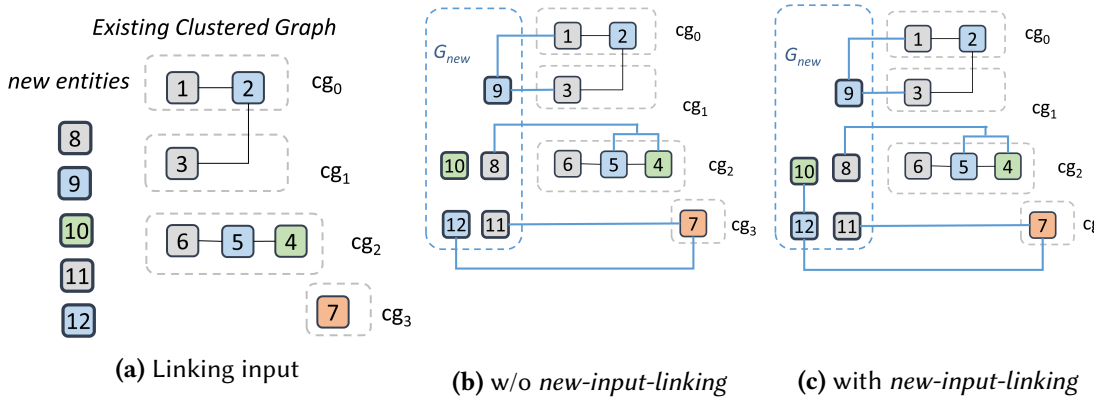
Incremental ER entails the two main steps of *Linking* and *Clustering*. The input of linking is an existing clustered graph $\mathcal{CG}_{exist}$ and a set of new entities $\mathcal{E}_{new}$ from already known sources or from a new source. For illustration, we consider a running example with existing entities from four sources (shown in part (a) of Figure 6.3) and new entities

id	name	surname	src	key
1	George	Walker	A	wa
2	george	Waker	B	
3	George	Wabel	A	
4	Frankie	Pollock	C	po
5	Franklin	Pollock	B	
6	Franklin	Pollim	A	
7	Berja	Summeahville	D	su

id	name	surname	src
8	Franklin	Pollock	A
9	George	Wabel	B
10	Bertha	Summercille	C
11	Berta	Summeahville	A
12	Bertha	Summeahville	B

key	wa				po				su			
id	1	2	3	9	4	5	6	8	7	10	11	12

Figure 6.3: Running example: existing entities, new entities and blocking



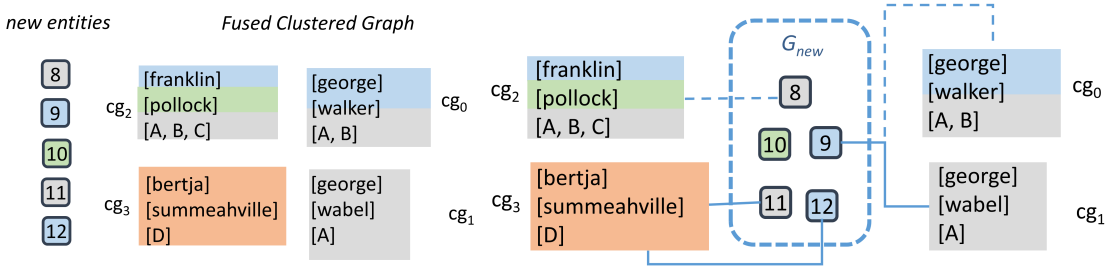
(a) Linking input **(b)** w/o *new-input-linking* **(c)** with *new-input-linking*

Figure 6.4: Incremental linking

to be integrated (shown in part (b) of Figure 6.3). As typical for real-world data, the entity properties are partly erroneous. Figure 6.4a shows the clustered similarity graph indicating that the previous entities form four clusters named cg_0 to cg_3. Note, that the colors indicate the originating source and that every cluster contains at most one entity per source.

For the linking of new entities we optionally support a linking among new entities. While this introduces additional computations, the additionally found links may lead to better clusters. Note that this *new-input-linking* is not applicable if all new entities are from the same source due to the assumption of duplicate-free sources. To limit the number of comparisons we apply blocking and only compare new entities with other entities of the same block. For the running example we assume that the two initial letters of the surname are used as blocking key (specified in the configuration) as shown in part (c) of Figure 6.3. Without new-input-linking, we only compare new entities (marked in blue) with previous entities of the same block. With new-input-linking, we additionally link new entities among each other, e.g., for blocking key *su*. All links between new entities with a similarity above a threshold (specified in the configuration) are added to the similarity graph. Figure 6.4b and Figure 6.4c illustrate the resulting grouped similarity graphs without and with new-input-linking, respectively. The only difference occurs for the new entity 10 which is not linked with any previous entity but a link with the new entity 12 is generated by new-input-linking so that entity 10 may be added to the same cluster.

The clustering part (second step of incremental ER) uses the determined grouped similarity graph $\mathcal{GG}$ and the clustering configuration as input. The clustering configuration specifies either one of the base methods or the repair method with their parameters (to be explained in Section 6.2.3 and Section 6.2.4). The output is an updated clustered graph $\mathcal{CG}_{updated}$ that includes the new entities within updated clusters.



(a) Linking input

(b) Linking output with fused clustered graph

Figure 6.5: Fusion example

The sketched process is similar when we choose to fuse all entities of a cluster to build cluster representatives and when we use a fused similarity graph instead of a clustered similarity graph. The reduced number of entities in this graph reduces the number of comparisons and can thus lead to a more efficient linking. Figure 6.5a shows the fused similarity graph of the running example to which the new entities have to be compared. The cluster representatives (fused entities) may contain per property multiple values from the original entities. When linking a new entity we can choose to only link to cluster representatives that do not yet include an entity from the same source. For example, in Figure 6.5b, the link between entity 9 and cluster cg_0 does not need to be created (indicated as dashed line) since this cluster already contains an entity of the same source.

6.2.3 Max-Both Merge

The max-both merge approach integrates new entities into already existing clusters or creates new clusters for them. The decision is based on the max-both (strong) links between new entities and already clustered entities. In case of new-input-linking, we first apply a pre-clustering among the linked new entities to create source-consistent clusters which may then be merged with the existing clusters. The case without new-input-linking can be viewed as a special case where each new entity forms a singleton cluster.

If $\mathcal{GG}$ is a grouped similarity graph consisting of $\mathcal{G}_{new}$, $\mathcal{CG}_{exist}$ and $\mathcal{L}_{exist\_new}$, the max-both approach merges a new cluster $n \in \mathcal{G}_{new}$ with an existing cluster $c \in \mathcal{CG}_{exist}$ if there is a max-both link $l(e_i, e_j) \in \mathcal{L}_{exist\_new}$ between a new entity $e_i \in n$ and an entity $e_j \in c$ and the two clusters n and c have only entities from different sources. Hence, max-both merge assigns a new cluster to the maximally similar existing cluster and merges them only if this does not violate source consistency. For the example in Figure 6.6, we would assign entity 9 neither to cluster cg_0 nor to cg_1 if the link between entity 9 and entity 1 of cg_0 has a higher similarity than the link with entity 3 of cg_1.

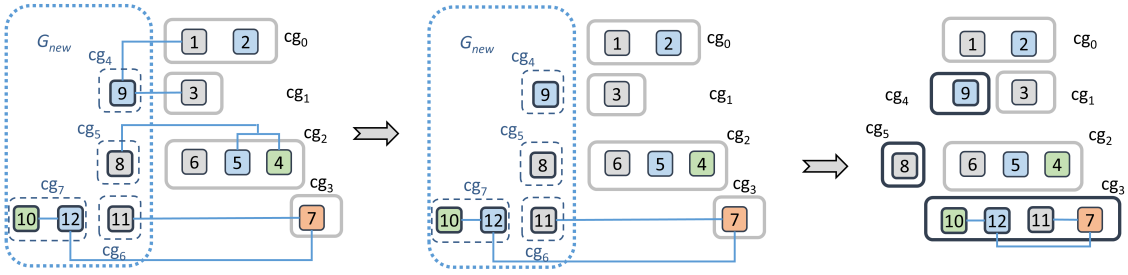


Figure 6.6: Max-Both merge

The further processing of the selected max-both links has to consider that max-both links ensure the maximal entity similarity only w.r.t. a fixed pair of sources. Hence, it is possible that clusters can have several max-both links referring to entities of different sources. As a result, it may be possible to merge more than two clusters as long as source consistency is ensured. For the example in Figure 6.6, we would merge three clusters including cg_6, cg_7 and cg_3, because the links from the new entities 11 and 12 to the existing entity 7 are max-both links and merging all of the associating clusters (cg_6, cg_7 and cg_3) as one cluster still keeps the source consistency constraint. When merging more than two clusters is not possible due to the source consistency constraint, we determine for each existing cluster cg_i, the linked new clusters as candidates. These candidate clusters are sorted and processed according to the link similarity and the cluster size giving preference for merging to higher similarity values and bigger candidate clusters.

Figure 6.6 illustrates the max-both merge algorithm for the grouped similarity graph of Figure 6.4c. The left part of the Figure 6.6 shows the result after pre-clustering the new entities resulting in clusters cg_4 to cg_7. Then, the links are selected that are max-both and that connect mergeable clusters as shown in the middle part of Figure 6.6 (the links from the new clusters cg_4 and cg_5 to clusters cg_0 and cg_2 would lead to source inconsistency and are thus removed). The right part of Figure 6.6 indicates the final merge result with six instead of eight clusters. The existing cluster cg_3 is linked to two new clusters cg_6 and cg_7. Assuming that both links have the same similarity value, the sort order would first consider the bigger cluster cg_7 and merge it. Then, cluster cg_6 is considered and also merged with cg_3 since source consistency is preserved.

For fused clusters, we use the provenance information in the cluster representatives to avoid linking new entities to clusters containing already an entity from the same source (Figure 6.5b). This leads to an incremental clustering result corresponding to the one for the max-both approach.

6.2.4 n-Depth Reclustering

The approaches described so far cannot add a new entity to an existing cluster if there is already another entity of the respective source. This can lead to wrong cluster decisions, e.g., if the previously added entity is less similar to the other cluster members than the new entity. Our n-depth reclustering scheme addresses this problem to obtain better clusters and to bcome largely independent from the order in which new entities are added. At the same time, we want to limit the amount of reclustering in order to maintain good efficiency.

The approach reclusters the new entities in $\mathcal{G}_{new}$ with their neighbors in the existing clustered graph $\mathcal{CG}_{exist}$. The parameter n controls the depth up to which the neighboring clusters and their entities are reconsidered thereby allowing us to control the scope of processing and associated overhead. For $n = 1$, the algorithm only re-evaluates entities of the existing clusters directly connected to the new entities. For $n = 2$, the neighbors of 1-depth neighbors are also selected. The selected portion of the grouped similarity graph $\mathcal{GG}$, $\mathcal{G}_{new}$ and the neighbors, are reclustered using a static clustering scheme.

Algorithm 10 outlines this process. In line 1, the neighbors up to depth n are determined. The union of the found neighbor clusters (including their intra- and inter-links) with the subgraph of new entities $\mathcal{G}_{new}$ forms the portion ($\mathcal{G}_{reclustering}$) of the grouped similarity graph to be re-clustered (line 2). In line 3, the static clustering scheme is applied leading to an updated set of clusters. Any clustering algorithm can be used for the `batchClustering`. In our experiments in Section 6.3 we used the CLIP algorithm that was shown in [160] to achieve better quality than other ER clustering approaches.

Figure 6.7 illustrates the algorithm for $n = 1$. The portion of the input to be reclustered consists of the new graph $\mathcal{G}_{new}$ and its 1-depth neighbor clusters (cg_0 to cg_3). The output (right part of the Figure 6.7) shows that the previous cluster cg_2 is changed so that the new entity 8 is included instead of the previous member 6 from the same source.

Algorithm 10: n-Depth Reclustering

Input: grouped similarity graph $\mathcal{GG}$ ($\mathcal{G}_{new}$, $\mathcal{CG}_{exist}$, $\mathcal{L}_{exist\_new}$), configuration $config$

Output: updated Clustered Graph $\mathcal{CG}_{updated}$

1 $\mathcal{CG}_{neighbors} \leftarrow$ `getNeighbors`$(\mathcal{GG}, n)$
2 $\mathcal{G}_{reclustering} \leftarrow \mathcal{CG}_{neighbors} \cup \mathcal{G}_{new} \cup \mathcal{L}_{exist\_new}$
3 $\mathcal{CG}_{new} \leftarrow$ `batchClustering`$(\mathcal{G}_{reclustering}, conf.getClustering())$
4 $\mathcal{CG}_{updated} \leftarrow \mathcal{GG}$
5 `updateGraph`$(\mathcal{GG}, \mathcal{CG}_{new})$
6 **return** $\mathcal{CG}_{updated}$

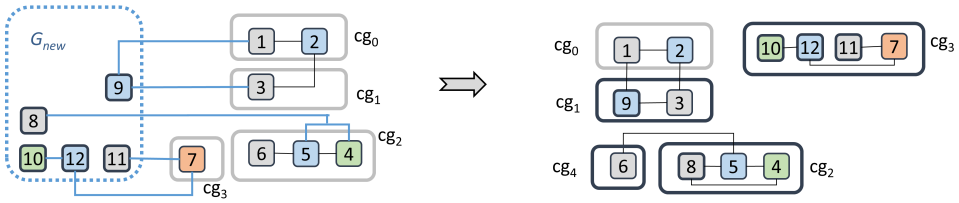
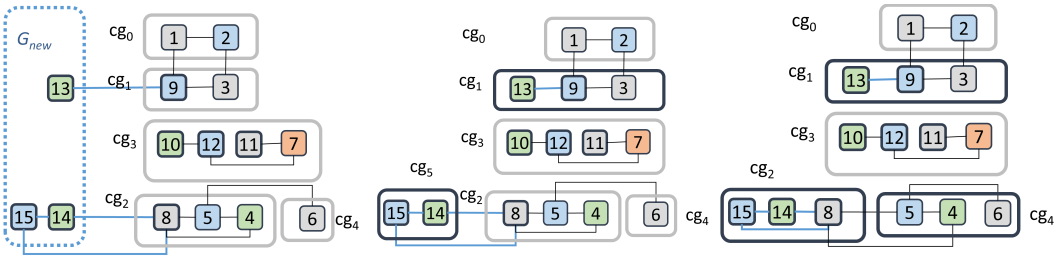


Figure 6.7: 1-depth reclustering (1DR)



(a) 2nd increment input **(b)** 1DR output **(c)** 2DR output

Figure 6.8: nDR example

Figure 6.8a shows the output of Figure 6.7 as existing clustered graph and the next increment of new entities (13, 14 and 15). By performing 1-depth reclustering (1DR), a small portion of the graph including clusters cg_1 and cg_2 plus the new entities are reclustered. As illustrated in Figure 6.8b only cluster cg_1 is modified and the entities 14 and 15 create a new cluster. For the same input choosing $n = 2$ would end to reclustering a bigger portion of the existing clustered graph compared with 1-depth reclustering. As illustrated in Figure 6.8c, the 2-depth neighbour cluster cg_4 and the 1-depth neighbor clusters, cg_1 and cg_2 are modified by the reclustering.

The introduced reclustering of existing clusters depends on the intra-cluster links. Therefore, the repairing method is not applicable for fused clusters.

6.3 Evaluation

We now evaluate the effectiveness and efficiency of the proposed incremental clustering/repairing algorithms in comparison to the batch ER approach of FAMER and the Greedy incremental cluster repair of [66]. We first describe the used datasets from three domains. We then analyze comparatively the match quality of the proposed algorithms. Finally, we evaluate runtime performance.

6.3.1 Datasets

We use datasets from four domains with different numbers of duplicate-free sources. The datasets are publicly available and have been used in prior ER studies[1]. Table 6.1 shows the main characteristics of the datasets in particular the number of clusters and match pairs of the perfect ER result. The datasets are explained thoroughly in Chapter 4 and Chapter 5. As explained in Section 5.4, we use the dataset of ACM SIGMOD 2020 Programming Contest[2] for the camera products. It contains camera specifications from 24 sources. The DS-C100 dataset is the result of removing the source *www.alibaba.com* [3] and deduplicating each source individually.

We evaluate our proposed methods with two scenarios of incremental ER. In the first scenario, called *sources-wise*, a complete new source is added to the existing clustered graph in each increment. In the second scenario, called *entity-wise*, specific portions of new entities from already existing sources are added to the clustered graph. For this case, we consider the four configurations listed in Table 6.2. Each configuration specifies the percentage of entities from each source that is added to the knowledge base in each increment. For example, in configuration $conf1$, the initial KG only contains 20% of the entities from each source. In each of the following four increments 20% of the entities from each source are added.

For linking, we apply different configurations for each dataset (listed in Table 6.3). All configurations use standard blocking with different blocking keys. The match rules rely on different attribute similarities using either string similarity functions (Jaro Winkler, Trigram) or geographical distance. For the camera dataset, we extracted the manufacturer name, a list of model names, manufacturer part number (mpn), european article number (ean), digital and optical zoom, camera dimensions, weight, product code, sensor type, price and resolution from the heterogeneous product specifications. In order to reduce the number of comparisons, standard blocking with a combined key of manufac-

Table 6.1: Evaluation datasets

	general information				perfect result	
	domain	entity properties	#entity	#src	#clusters	#links
DS-G	geography	label, longtitude, latitude	3,054	4	820	4,391
DS-C100	cameras	heterogeneous key-value pairs	8,123	23	3,910	16,014
DS-M	music	artist, title, album, year, length	19,375	5	10,000	16,250
DS-P	persons	name, surname, suburb, postcode	10,000,000	10	6,625,848	14,995,973

[1] https://dbs.uni-leipzig.de/research/projects/object_matching/benchmark_datasets_for_entity_resolution

[2] http://www.inf.uniroma3.it/db/sigmod2020contest/index.html

[3] This source mostly contains non-camera entities.

Table 6.2: Increment configurations

conf	1	2	3	4
base	20%	33%	50%	80%
inc 1	20%	33%	10%	10%
inc 2	20%	33%	10%	10%
inc 3	20%	-	10%	-
inc 4	20%	-	10%	-
inc 5	-	-	10%	-

Table 6.3: Linking configurations

	blocking key	similarity function
DS-G	prefixLength1 (label)	Jaro Winkler (label) geographical distance
DS-M	prefixLength1 (artist+title+album)	Trigram (artist+title+album)
DS-P	prefixLength4 (surname) + prefixLength4 (name)	avg (Trigram (name) + Trigram (surname) + Trigram (postcode) + Trigram (suburb))

turer name and model number is applied. Within these blocks, all pairs with exactly the same model name, mpn or ean are classified as matches. We assign a similarity value to the matched pairs determined from a weighted average of the 3Gram similarity of string values and a numerical similarity of numerical values (within a maximal distance of 30%).

6.3.2 Evaluation Results

Initially we evaluate the quality and robustness of our proposed methods for source-wise incremental ER. As described in Section 6.2.2, we do not need to perform new-input-linking and pre-clustering in this scenario since sources are duplicate free.

To analyze the impact of the order in which we add sources, we start with the results for the real-world datasets DS-G and DS-C100 where the sources differ strongly in size and quality. We compare our proposed incremental methods against the batch clustering approach of FAMER as well as the re-implemented Greedy algorithm from [66]. In Figure 6.9 and Figure 6.10, we show the obtained cluster quality results in terms of precision, recall and F-Measure for different similarity threshold of the linking phase which influences the number and the quality of generated links that are input to clustering.

Lower thresholds produce more links (good recall) at a higher chance of wrong links (lower precision) while higher thresholds lead to the opposite behavior.

For DS-G, twelve different orders of adding sources are possible. We examined all of them and report results for the best order ”ny, fb, geo, dp” (conf1) and the worst order ”dp, geo, ny, fb” (conf2) in Figure 6.9. For DS-C100 (Figure 6.10) similarly there are 253 different orders. We considered the best and the worst orders by adding sources in the descending (conf1) and ascending (conf2) oder of their sizes. With a good insert order, the quality of all approaches including MB (max-both merge) are close together and as good as batch ER. However, for the worst order MB achieves substantially lower recall and F-measure values indicating its strong dependency on the insert order. By contrast, our proposed re-clustering approach nDR (n=1) strongly reduces the dependency on the insert order and achieves the same quality as batch ER. The weakest results are observed for the Greedy approach [66]. Greedy initially tries to merge new entities to a randomly chosen neighboring cluster without considering the actual similarity value of the link. Then, if merging is not possible, it tries to maximize the objective function of the clus-

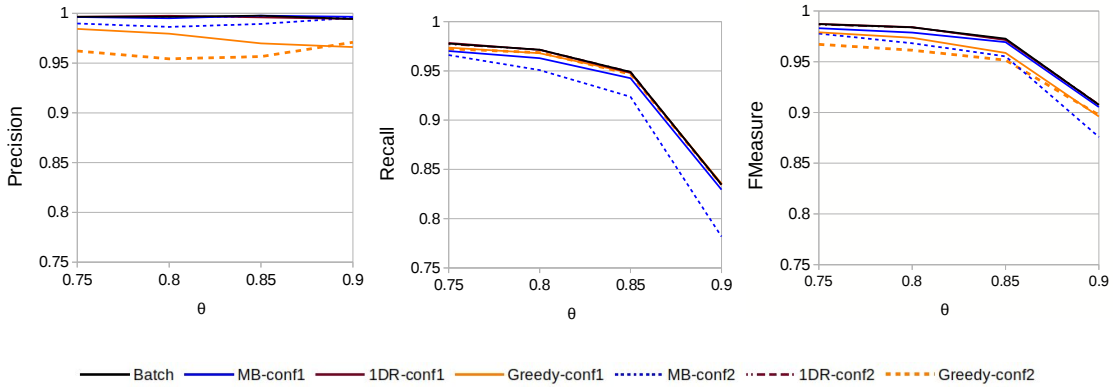


Figure 6.9: Source-wise cluster quality for dataset DS-G

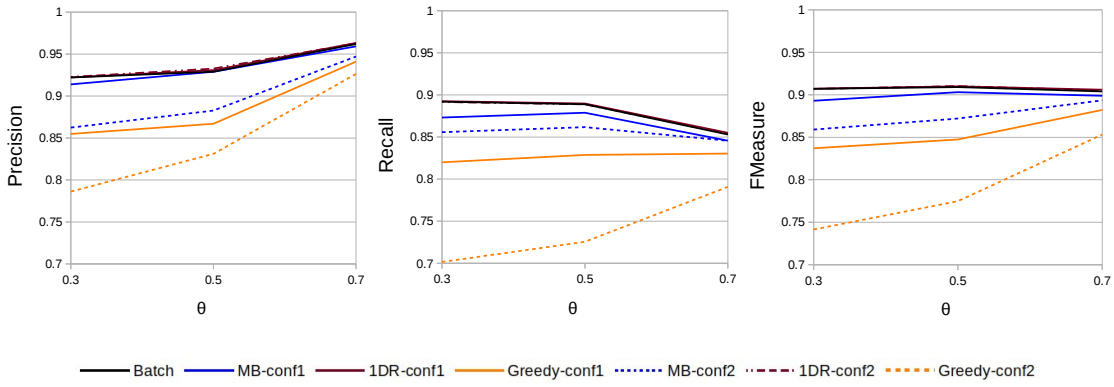


Figure 6.10: Source-wise cluster quality for dataset DS-C100

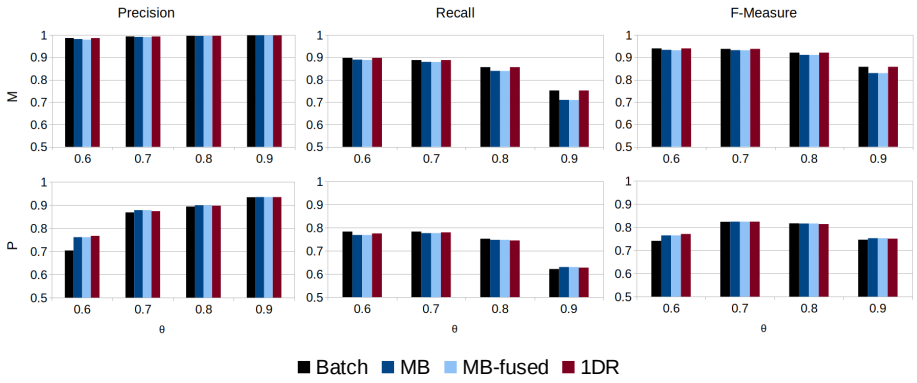


Figure 6.11: Source-wise incremental ER for DS-M (1st row) and DS-P (2nd row)

tering algorithm by iteratively splitting existing clusters and moving entities in between clusters until no the objective function is not improved further. However, the random assignment is problematic when a new entity has multiple neighboring clusters. We observed that even after many iterations of merge, split and move, some entities do not end up in the optimal cluster. Moreover, Greedy suffers from very long execution times due to its iterative nature and some experiments for larger datasets could not even finish. Therefore, the quality (particularly precision) results as well as the run-times are significantly lower than with our proposed approaches.

In Figure 6.11 we compare the cluster quality of our proposed methods against the non-incremental batch clustering approach of FAMER for datasets M and P and different similarity thresholds for linking. In all experiments, our incremental methods are able to compete with batch clustering. For dataset M (first row in Figure 6.11) all methods achieve high values for precision but lower recall values. The recall of the max-both approaches is consistently lower than nDR (n=1) which is like for dataset DS-G as effective as the batch approach. For the largest dataset P, the results are slightly different. Surprisingly, here all incremental methods could achieve better precision than batch clustering. This can be explained by the maximum possible cluster size of 10 while the average cluster size is only about 1.5 for this dataset. In batch clustering 10 entities from 10 different sources can be linked and considered as one cluster. Incremental methods do only touch the direct neighboring entities of the linked new entities. Hence, it is less likely for them to create clusters of non-matching entities.

In Figure 6.12 we report the F-Measure results for entity-wise incremental ER with the different increment configurations from Table 6.2. We evaluate all methods with and without new-input-linking (we use subscript IL to indicate new-input-linking). MB_{IL} achieves higher F-Measure than MB due to better recall. The positive effect of new-input-linking is also visible in the results for 1DR so that 1DR_{IL} mostly achieves higher

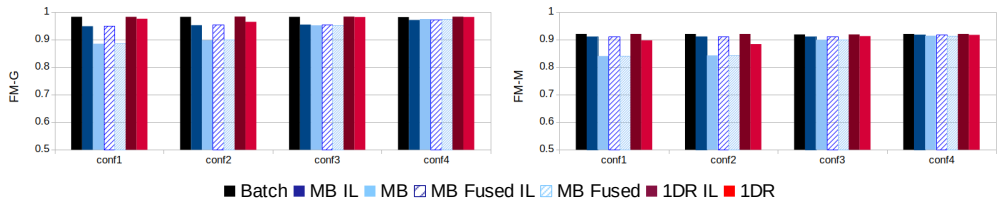


Figure 6.12: F-Measure results for entity-wise incremental ER

F-Measure than 1DR. The difference of methods with new-input-linking compared with their counterparts without new-input-linking in $conf3$ and $conf4$ is lower because a big portion of the dataset is already contained in the initial knowledge base and the data increments only contain 10% of the dataset. Therefore, when the volume of data in a new increment is much smaller than the volume of the existing knowledge graph, we may save the overhead of new-input-linking and pre-clustering. The approach $1DR_{IL}$ with new-input-linking consistently achieves the best results in all scenarios and newer achieves lower F-Measure than batch ER for our configurations.

6.3.3 Efficiency Evaluation

The run-times of all approaches are evaluated for the large dataset P and using a Hadoop cluster with 16 worker nodes, each consisting of an E5-2430 6(12) 2.5 Ghz CPU, 48 GB RAM and two 4 TB SATA disks. The nodes are connected via 1 Gigabit Ethernet. The used software versions are Flink 1.6.0 and Hadoop 2.6.0. We run Apache Flink with 6 threads and 40 GB memory per worker.

Table 6.4 shows the accumulated runtimes when executing the methods on clusters with 4, 8 and 16 workers for the large dataset P with a linking threshold of 0.7. As expected, all incremental approaches are faster than Batch. Moreover, the MB approaches are faster than our 1DR method. The reason is, that MB methods just process newly computed links while 1DR relies on intra-links of already existing clusters and the newly computed links. All methods achieve their best runtime with 16 workers. Batch shows to

Table 6.4: Accumulated runtimes in seconds for source-wise ER

#W	P $_{t_{min}0.7}$			
	Batch	MB	MB-fused	1DR
4	117 852	5 648	2220	21 179
8	33 791	2 178	1 562	4 283
16	8 542	1 778	1 184	2 513

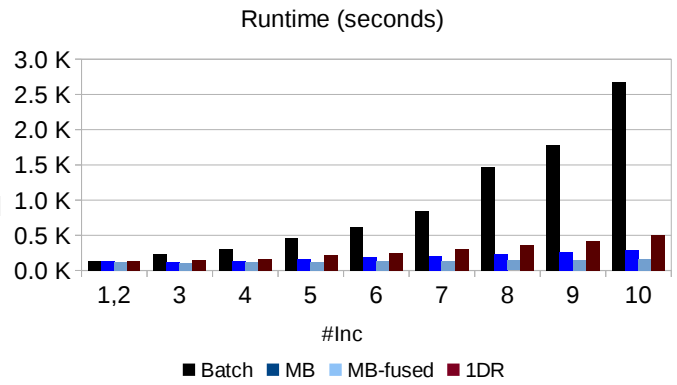


Figure 6.13: Incremental runtimes

have a better speedup, but starts at a much slower run-time. It is important to note that with less resources (less number of workers), the Batch runtime is significantly higher than the others. As expected, MB-fused performs slightly faster than MB.

In a further experiment we evaluated the runtimes of adding sources incrementally for dataset P. Figure 6.13 shows results of all 10 increments (adding 1 source per increment) for 16 workers. In every increment the incremental approaches are faster than the Batch method and MB-fused is faster than MB and both of them are faster than 1DR. In later increments the differences become higher. For example in the 10th increment the runtime of Batch is 5 times higher than 1DR. The reason is, that Batch clustering needs to process all vertices and links in each increment, whereas MB and 1DR only need to process a small fraction of links.

6.4 Related Works

Relatively little work has been done on incremental ER to deal with new entities which should be fast and not have to repeat the linkage of already linked entities. Most of these approaches [13][34][177] focus on a single data source only. In these approaches, new entities are either added to the most similar cluster (group) of entities or are considered as new entities. These approaches do neither aim at an optimized cluster assignment for sets of new entities nor do they repair previous match and cluster decisions.

Only little work coped with repairing previous cluster decisions for incremental ER and the previous approaches focus on a single source. Gruenheid et al. [66] maintain the clusters within a similarity graph and propose several approaches to update this graph based on different portions of the graph. Furthermore, a greedy method is introduced to use the updated graph to correct clusters by merging and splitting them or by moving entities among clusters. Nascimento et al. [118] extend the approach of [66] by defining

six filters to limit the number of cluster updates. The filters improve runtime but also reduce the quality. The evaluations in both [66] and [118] are limited to small single-source datasets. In our evaluation we will also consider the greedy approach of [66].

To our knowledge, there is no previous method for multi-source incremental entity clustering except the initial approach introduced in [122]. This method assumes duplicate-free sources and provides an optimized addition for sets of new entities or entire new sources which was shown to achieve better cluster quality than the isolated addition of one new entity at a time. The most effective approach was a so-called *max-both* assignment where an entity e from a set S of new entities is only assigned to the cluster c with the highest similarity to e (above a minimal similarity threshold) if there is no other entity in S from the same source than e with a higher similarity.

Here, we substantially extend this simple approach by considering more options for incremental linkage, in particular the optional linkage among new entities and the use of cluster fusion. Moreover, we propose and evaluate a new repair method for incremental multi-source entity clustering. We also provide distributed implementations of the approaches for improved performance.

6.5 Conclusion

In this chapter we proposed several new incremental methods for multi-source ER including a new method that can repair previous linking and cluster decisions. Our evaluation with datasets from different domains shows that the incremental approaches are much faster and similarly effective than batch ER. In particular, the introduced repair and re-clustering approach nDR achieves the same quality than batch ER while being still much faster. Its high effectiveness also shows that the quality does not depend on the order in which new entities are added in contrast to the non-repairing approaches such as max-both merge and previous repair schemes.

7 Conclusion and Outlook

7.1 Conclusion

This dissertation initiated with a comprehensive discussion about entity resolution and the necessity of clustering in multi-source scenarios as the last step of the ER pipeline. Furthermore, the existing clustering algorithms that have been already used in ER applications were overviewed. Finally, the framework FAMER (FAst Multi-source Entity Resolution system) was introduced and a comprehensive elaboration on its *Linking* and *Clustering* components has been given.

The focus of this thesis is on designing and developing exclusive clustering schemes for different ER scenarios. To this end, existing clustering approaches initially were implemented and incorporated for multiple clean data sources. They were comparatively compared in terms of both efficiency and efficacy. By investigating the output of the implemented clustering algorithms with the tool SIMG-VIZ (our developed visualization tool for FAMER), new schemes for clustering and repairing were proposed in order to overcome the deficiencies of the previously implemented approaches in multi-source clean scenarios. Thereafter, particular clustering schemes were adapted to tackle the more complicated multi-source scenarios such as having data from a dataset comprised of various clean and dirty heterogeneous sources. Moreover, several approaches were proposed to handle the incremental addition of new sources as well as new entities from existing sources. Following the Big Data needs and requirements, all approaches can be executed in parallel on different number of machines utilizing the Apache Flink framework.

7.1.1 CLEAN CLUSTERING

Previous research put little focus on clustering for the purpose of entity resolution specifically when data is from multiple (> 2) data sources. This thesis initially starts with investigating clustering algorithms for performing entity resolution in a single source scenario and applies them in multi clean source datasets. Then, the existing clustering methods are comparatively evaluated in both terms of efficiency and efficacy. With investigating the clustering results and considering ER-specific requirements for a clustering algorithm (e.g. the clustering scheme requires no the predefined number of clusters or other domain specific parameters as input) a new clustering algorithm called CLIP (Clustering based on LInk Priority) and a repairing method RLIP (Repairing based on LInk Priority) are proposed.

One lesson learned by inspecting output of existing clustering schemes is that the method should not be confined to the abstract value of similarities between entities (edge weights). Therefore, CLIP classifies links into three groups by defining the concept of link strength. In a multi clean source scenario, the link from an entity of source A to an entity of source B with the highest similarity value from both sides is classified as a *Strong link*. If the link has the highest similarity just from one side is a *Normal link* and if it does not have the highest similarity from both sides the link is classified as a *Weak link*. CLIP initially removes all weak links and then in the first phase forms clusters by considering only strong links and finalizes a cluster if it is a source-consistent complete cluster (i.e. the cluster contains entities from all sources), then in the second phase the remaining unclustered entities are grouped into clusters by prioritizing strong and normal links based on similarity value and strength. The impact rate of each criterion for prioritizing links can be adjusted by input parameters. In the early variation of CLIP links are prioritized dynamically i.e. after forming each cluster the remaining links were again prioritized but for efficiency reasons we decided to perform the prioritization task only once because the experiments also indicate the little impact of dynamic prioritization on the output quality. CLIP yields the best results in both quality and runtime compared with generic purpose clustering algorithms.

Probing the output of clustering algorithms reveals that some clustering schemes result in overlapping clusters which is improper in ER applications. Therefore, the repairing method RLIP resolves the overlapping as well as source-inconstant clusters by the link strength beside similarity values. RLIP improves the quality of output results of other algorithms at the cost of the additional post-processing step.

The pivotal concept of *link strength* has been further investigated in other researches and revealed positive effect on both quality and runtime. Considering max-both links

(strong links) as a post-processing method resulted in superior results compared to *Stable Marriage (SM)* and *Maximum Weight Matchings (MWM)* methods for privacy-preserving record linkage [51]. Moreover, removing weak links as a preprocessing step made execution of the novel clustering method *Sort and Keep Best (SKB)* [163] feasible.

7.1.2 CLEAN/DIRTY CLUSTERING

In real world scenarios specially when data is from numerous sources, only a portion of sources are clean. Thus, we were initially supposed to deduplicate each source individually in one round and then applying the CLIP algorithm in the second round in order to produce the final resulting clusters. Following the two-round strategy may result in low quality particularly when one or a few of sources has a very low quality which is prevalent case in multi-source ER. Therefore, we investigated the problem with heterogeneous datasets comprised of data sources with various number of entities and dissimilar quality. We created eight datasets each comprised of a combination of clean and dirty sources. Totally, we considered a full range of datasets containing 0% to 100% of clean sources. In order to devise a clustering scheme for multi-source clean/dirty (MSCD) scenarios, we adapted Affinity Propagation (AP) which is a constrained-based clustering algorithm. We further extended AP by adding the source-consistency constraint to it. The extensive evaluation showed superior results (up to 12% improvements in precision) compared to general-purpose clustering algorithms and standard AP especially in precision. Moreover, we adapted Hierarchical Agglomerative Clustering (HAC) and proposed the MSCD-HAC. The basic HAC initially considers each entity as a cluster and then merges the two most similar clusters iteratively until a condition is satisfied. In order to assess the two most similar clusters we applied three different linkage types based on the maximum link (single linkage), the minimum link (complete linkage), and the average similarity (average linkage) between two clusters. An extensive evaluation of comparing the linkage types as well as comparison with MACD-AP and generic-purpose algorithms revealed that MSCD SLINK (MSCD-HAC with single linkage strategy) results in superior clusters compared to other linkage types (up to 8% improvement in F-Measure), generic-purpose schemes and MSCD-AP algorithm.

7.1.3 INCREMENTAL CLUSTERING

Most previous work on entity resolution deals with static ER to match entities from one or several static data sources. Such static approaches are not sufficient to handle evolving data and ever increasing number of sources of Big Data. Due to the fact that the

majority of already integrated entities are largely unaffected by new entities, they do not need to be re-integrated for every update. Therefore, novel approaches are needed to integrate new data without requiring re-computations of previous results. The incremental approaches are further expected to yield results with the overall quality the same as batch ER where all entities are simultaneously integrated.

To this end, we developed approaches for integrating new data to the already existing integrated data. The incremental pipeline of the FAMER supports incremental addition of new sources as well as addition of new entities to the existing sources. The *Linking* part of FAMER is capable of maintaining previously computed results including links and clusters and creating new links between either existing and new entities or among new entities. The output of linking is a *grouped similarity graph* which comprised of already clustered entities that are clustered in previous iterations plus the newly linked entities. The incremental clustering/repairing of FAMER encompasses two main approaches for incorporating new entities and sources. One approach considers the so-called max-both assignments that adds an entity to the most similar cluster only when there is no more similar new entity from the respective data source. It furthermore supports the fusion of cluster members to a single entity which simplifies and speed-ups incremental clustering as new entities need no longer be compared to several entities of a cluster. The output quality of the basic method is dependent to the order of entering new entities or sources. Thus, we developed a sophisticated method called *n-depth reclustering (nDR)* (where n is a parameter to control the portion of the similarity graph that is considered for reclustering) which is able to repair existing clusters for improved quality and a reduced dependency on the insert order of new entities. The output of incremental clustering is a fully clustered graph.

The incremental approaches are evaluated for datasets of four domains in terms of cluster quality and runtime efficiency. We also provide a comparison to a previous approach for incremental cluster repair [66] and with batch ER. A detailed evaluation of the new approaches for real-world workloads shows their high effectiveness. In particular, the repair approach outperforms other incremental approaches and achieves the same quality than with batch-like entity resolution showing that its results are independent from the order in which new entities are added.

7.1.4 PARALLELIZATION OF ALL STRATEGIES

In order to serve the voluminous data, scalability has been one of the most important aspects that is considered and evaluated in this thesis. Therefore, the proposed framework FAMER is implemented on top of Apache Flink with the aim of developing scalable

solutions for data integration. To this end all approaches are in fact designed as parallel algorithms that are implemented using the Flink APIs and are executable on different number of machines in a distributed fashion by Flink engine.

In *Linking part*, we implemented blocking approaches in order to reduce the number of comparisons. Both standard blocking and sorted neighborhood methods can be simply implemented using grouping transformation of FAMER, but the naive implementation leads to load balancing problem due to the block skews. Thus, the proposed blocking methods for Map/Reduce were adapted according to Flink APIs and were implemented for FAMER. The method named as *Pair Range* [89] implements standard blocking by distributing comparisons evenly on each machine using the partitioner transformation of Flink. Prior to partitioning the statistics of each block is computed by a chain of grouping, join and map transformations. Due to the fact that the big blocks are spitted into multiple blocks, a number of pairs are replicated for each partition. Similarly, the load balanced sorted neighborhood [91] distributes entities evenly on each task manager by a map transformation. Then, the last *windowSize-1* entities of each task manager are replicated for the subsequent task manager in order to produce correct blocking results. Finally, pairs of candidate entities are generated inside each task manager by sliding down a window implemented by grouping transformation.

Due to the fact that clustering is inherently an iterative task, the vertex-centric model is used to implement most of the approaches in *Clustering part*. Gelly library of Flink includes various variations that implement the vertex-centric programming paradigm. In scatter-gather variation, the vertices are able to send messages to any other vertices accessible by vertex ID and update their stored information (status) by the received messages. If a vertex does not receive any message during one superstep, then it is out of sending and receiving game. Therefore, when there is no need to update a vertex status, the iterative execution is terminated. For implementing each iteration of the clustering algorithms, one or multiple supersteps were designed in a specific order. In each iteration, a group of vertices store their final cluster-ID and thus are removed from the graph. Vertices with the same cluster-ID belong to the same cluster. Nevertheless, CLIP is not implemented on top of Gelly. It initially computes the strength of edges by grouping neighboring entities (vertices) by the source of the other end. Then, in phase one, the components formed by strong links that comprise of entities from all sources are considered as clusters. In the next phase, the remaining vertices are clustered by prioritizing normal and strong edges. Again a grouping transformation assists on processing the remaining components so that each component is processed on a task manager in parallel with other components.

The scalable variation of Affinity Propagation (AP) is named Hierarchical Affinity Propagation (HAP) that follows the divide and conquer strategy. HAP performs AP on multiple hierarchy levels by initially dividing the dataset into equal-sized partitions randomly. Then, a set of local exemplars are produced for each partition by performing AP locally on each partition. By merging the exemplars of each level and repeating the process, the number of entities on top hierarchy level is small enough (less or equal to a predefined parameter M) to stop the partitioning process. The exemplars determined on the top level are considered as the global exemplars for the dataset. Finally, all non-exemplar entities are assigned to a global exemplar with the highest similarity.

The evaluations showed that FAMER scales very well with increasing number of data sources. On the other hand, utilizing the whole capacity of our configured cluster (16 machines) did not result in the maximum expected speed-up in most cases. This is explained by the incompatibility of the Flink underlying system with graph-structure data which leads to large communication overhead [154]. Therefore, improving data distribution techniques tailored to Apache Flink optimizer and scheduler may result in achieving a better speedup behavior.

7.2 Outlook

This thesis is dedicated to developing an ER framework that adapts with big data requirements. The focus of the research is mostly on clustering and repairing methods when data is from multiple sources. Considering the big data needs and requirements such as variety, velocity and volume, there are still room for improvements in both efficacy and efficiency aspects. Moreover, implementing the state of the art techniques such deep learning based methods sounds promising and opens doors for upgrading FAMER to a knowledge graph. Appraising comprehensive findings of this thesis, various important directions for future research are elaborated in the following sections.

7.2.1 Reinforcing variety support

Considering numerous sources as input of the ER framework implies dealing with unstructured, semi-structured or structured but highly heterogeneous data in different formats (XML, JSON, data from SPARQLE endpoints or relational databases). Therefore, ER frameworks must be capable of performing ER task in presence of schema heterogeneity by adapting their ER pipeline according to the new requirements. To this end, one solution is augmenting the ER process with a *schema matching* or *schema refinement*

preprocessing step and the other solution is developing schema-agnostic blocking and matching methods.

In order to augment FAMER with schema matching, the linking and specifically clustering modules can be utilized to define a schema matching operator. For example, by reusing existing links of Linked Data, the entity attributes as well as the corresponding values of linked entities can be matched together in order to produce clusters of matching attributes. Moreover, supplementing FAMER with schema refinement operators [70] facilitates finding syntactically similar attributes which increases the performance of the next workflow steps [138].

Applying traditional blocking methods for complicated data implies the essential need for background knowledge as well as many trials and errors to find the optimal blocking configuration. Therefore, developing learning-based approaches to automate this process is an essential. Furthermore, schema agnostic blocking methods such as token blocking [136] and its successor refinements [107, 137] make the FAMER capable of processing loosely structured data. Token blocking considers entities with at least a common token in their attribute values in the same block. Finally, in the matching part, the supervised and unsupervised methods that learn matching rules [126] manage matching noisy and unstructured data.

7.2.2 Reinforcing velocity support

Velocity refers to the challenge of continuous increase of data volume over time. In order to cover this specification of Big Data we developed incremental methods (see Chapter 6) which covers adding new entities from existing sources or even adding a new source to the knowledge graph. However, adding other operations such change and delete would be also interesting. Moreover, adapting an incremental blocking scheme such as the ones introduced in [30, 148] for FAMER would boost the performance of incremental methods enormously, because the experiments show that runtime is already mostly wasted in Linking part. Due to the fact that variety is still existing augmenting FAMER with incremental schema matching method [52, 162] seems essential .

Another interesting topic is considering entities with their associating temporal information. Performing ER then allows the system to keep track of one entity over time [29]. For example, grouping all publication records according to their authors allows us to view the publication history of each author [25]. FAMER is capacitated to augment temporal entity resolution by embedded temporal graph analysis of Gradoop [153, 154].

7.2.3 Storing and utilizing provenance information

In this dissertation we utilized the cleanliness status of data sources to cluster entities of multiple sources. Expanding the use of provenance information may result in designing sophisticated approaches for improving the quality of matching and merging [105]. Provenance information regarding the provenance of existing results focuses on describing both the origins and the processing of data [75]. In Entity Resolution applications beside the information related to data sources such as validity periods and security requirement, the details of matching and classification processes can further be stored and utilized [174]. For example, keeping the matching rule for a pair of matched entities beside the pure similarity value facilitates understanding and debugging of the whole ER pipeline [134]. Therefore, resolving inconsistent matched entities and grouping them into clusters as well as repairing process in incremental ER would be accomplished in a more accurate way.

7.2.4 Utilizing versatile types of entity attributes and leveraging external knowledge

Matching entities from multiple heterogeneous data sources implies that each source may carry a piece of information that can not be obtained from other sources. Catching all specifications of an entity leads into forming an accurate description that facilitates data integration accuracy. To this end, different data types such as factual, textual, or image-based attributes should be considered in the resolving task. This topic is covered as *multi-modal entity resolution*. A recent work [180] shows that extending the attribute-based matching system to incorporate image data for product matching task results in improved recall and overall quality.

Another interesting topic is to augment FAMER with word embeddings in both blocking [45, 186] and pair comparison steps [45, 113] as well as schema matching [5]. Word embeddings are vectorial representations of words that are resulted by pretraining words (phrases or even characters) over large and generic corpora. Therefore, these methods provide a representation of words which is independent of a specific domain or a particular ER task [138]. The examples of such pretrained models are word2vec [109], GloVe [143], and fastText [18].

7.2.5 Rehabilitating FAMER to a semantic knowledge graph

FAMER is already a graph-based framework which aims at discovering identical links between entities. Considering other types of relations as well as different types of enti-

ties will reinforce the framework enormously. Consequently FAMER would not be restricted to a single domain i.e. it will be able to cover many domains that exist in the real world. Possessing all mentioned criteria [142] makes the framework a knowledge graph which augmenting it by completion and correction operators and pipelines facilitates the data integration task. Furthermore, utilizing methods such as graph embeddings beside similarity of entity attributes assists integrating multiple knowledge graphs with high accuracy [132]. Graph embeddings for ER facilitate determining the similarity of entities based on the similarity of their graph neighborhood.

A

FAMER Configurations

A.1 Preprocessing

Listing A.1, Listing A.2, and Listing A.3 depict the configuration of different preprocessing tasks that are defined in Chapter 3.

Listing A.1: Read logical graph configuration

```
{
  "task":"READ",
  "return":"LOGICAL_GRAPH",
  "path":"path/to/graphDataFolder"
}
```

Listing A.2: Read data from a given benchmark dataset

```
{
  "task":"BENCHMARK",
  "name":"ABT_BUY",
  "return":"LOGICAL_GRAPH",
  "path":"path/to/benchmarkDataFolder"
}
```

Listing A.3: Read and combine any number of Gradoop logical graphs to a Gradoop graph collection

```
{
  "task":"COMBINE",
  "return":"GRAPH_COLLECTION",
  "graphs": "[...]"
}
```

A.2 Linking

A.2.1 Blocking

Listing A.4 depicts a configuration example that specifies the blocking details for FAMER. The selected blocking method is Sorted Neighborhood with *window size* of 20. The key is generated from the five initial letters of the *category* property. The *parallelism degree* specifies the number of task managers that are employed by Apache Flink. The blocking method uses this number in order to splitting comparisons evenly on the task managers.

Listing A.4: FAMER blocking configuration

```
"blockingMethod":"SortedNeighborhoodComponent",
 "windowSize":"20",
 "parallelismDegree":"96",
 "keyGenerationComponent":{
        "keyGenerationMethod":"PrefixLengthComponent",
        "attribute":"category",
    "prefixLength":"5"
 }
```

A.2.2 Pair-wise Comparison

Listing A.5 depicts a configuration example that specifies the similarity computation details of FAMER for a dataset containing cameras. The list of similarity components constitute of two similarity components. The first one computes the *Truncate Begin* similarity degree of the *brands* of two cameras and the second one computes the *Edit distance* similarity between the *models*. Each similarity component has the *weight* of 0.5 that specifies the effect of the computed similarity on the final aggregated similarity value.

Listing A.6 depicts a similarity aggregation configuration. It implies computing the aggregated similarity value by the mathematical expression $\frac{brand\_similarity+model\_similarity}{3}$.

Listing A.5: Similarity components configuration

```
"similarityComponents":[
  {
     "id":"camera_brand",
     "sourceGraph":"*",
     "targetGraph":"*",
     "sourceLabel":"camera",
     "targetLabel":"camera",
     "sourceAttribute":"brand",
     "targetAttribute":"brand",
     "similarityMethod":"TruncateBeginComponent",
     "length":"4",
     "weight":"0.5"
  },
  {
     "id":"camera_model",
     "sourceGraph":"*",
     "targetGraph":"*",
     "sourceLabel":"camera",
     "targetLabel":"camera",
     "sourceAttribute":"model",
     "targetAttribute":"model",
     "similarityMethod":"EditDistanceComponent",
     "weight":"0.5"
  }
]
```

If the computed value is bigger than 0.3, then an edge with the computed similarity value is generated. If the ”rule components” part is empty (Listing A.7), then by default the weighted average of all similarity degrees are computed as the aggregated similarity value.

A.2.3 Classification

FAMER offers both threshold-based and rule-based classification methods as well as the combination of them. The set of basic *rule components* are listed as ”(”, ”)”, ”similarity id” and ”selection operations”. The user-defined selection rule can be defined as shown in Listing A.8. It classifies the pairs with *camera_model ≥ 0.7 AND camera_priceUSD ≥ 0.8* as true matches only if their aggregated similarity value is bigger than or equal to *0.5*. More sophisticated selection rules such as *(camera_model ≥ 0.7 AND camera_priceUSD ≥ 0.8) OR (camera_model > 0.5 AND camera_weightKg > 0.6)* can be defined by combining

Listing A.6: Similarity aggregation sample configuration

```
"similarityComponents":[
"aggregationRule":{
    "aggregationThreshold":"0.3",
    "ruleComponents":[
      {
         "componentType":"OPEN_PARENTHESIS"
      },
      {
         "componentType":"SIMILARITY_FIELD_ID",
         "value":"camera_brand"
      },
      {
         "componentType":"ARITHMETIC_OPERATOR",
         "value":"PLUS"
      },
      {
         "componentType":"SIMILARITY_FIELD_ID",
         "value":"camera_model"
      },
      {
         "componentType":"CLOSE_PARENTHESIS"
      },
      {
         "componentType":"ARITHMETIC_OPERATOR",
         "value":"DIVISION"
      },
      {
         "componentType":"CONSTANT",
         "value":"3"
      }
    ]
  }
]
```

Listing A.7: Similarity aggregation sample configuration

```
"similarityComponents":[
        "aggregationRule":{
                "aggregationThreshold":"0.2",
        }
]
```

Listing A.8: Selction rule configuration (camera_model ≥ 0.7 AND camera_priceUSD ≥ 0.8)

```
"selectionComponent":{
  "selectionMethod":"MANUAL",
  "aggregationRule":{
    "aggregationThreshold":"0.5",
  }
  "selectionRule":{
    "ruleComponents":[
      {
        "componentType":"CONDITION",
        "conditionId":"con1",
        "similarityFieldId":"camera_model",
        "operator":"GREATER_EQUAL",
        "threshold":"0.7",
        "defaultOnAttributeNull":"true"
      },
      {
        "componentType":"SELECTION_OPERATOR",
        "value":"AND"
      },
      {
        "componentType":"CONDITION",
        "conditionId":"con2",
        "similarityFieldId":"camera_priceUSD",
        "operator":"GREATER_EQUAL",
        "threshold":"0.8"
      }
    ]
}
```

the available rule components. It may happen that a similarity field is not available for a pair due to the absence of corresponding properties in the paired entity. In this case the default value can be defined via *defaultOnAttributeNull*.

A.3 Clustering

Listing A.9 depicts the FAMER configuration for clustering. The configuration specifies ”Center” algorithm as ”clusteringMethod”. All algorithms are using ”maxIteration”, which can mostly be set to ”MAX_VALUE” or a specific integer value in the json configuration. It specifies the maximum number of iterations for the clustering algorithm. In case ”maxIteration” is set to ”MAX_VALUE”, the clustering algorithm terminates when

Listing A.9: Clustering configuration

```
"clustering":{
    "clusteringMethod":"CENTER",
    "prioritySelection":"MIN",
    "isEdgesBiDirected":false,
    "clusteringOutputType":"GRAPH",
    "maxIteration":"MAX_VALUE"
  }
```

all vertices are clustered. The rest of lines in the configuration file identifies the input parameters of the clustering algorithm.

The list of clustering algorithms supported by FAMER (listed in Table 3.1) as well as their configuration are explained in FAMER wiki [1].

A.4 Postprocessing

Listing A.10 depicts the configuration for writing a logical graph to the disk.

Listing A.10: Write logical graph configuration

```
{
  "task":"WRITE_GRAPH",
  "path":"path/to/graphDataFolder",
  "overwrite":"false"
}
```

A.5 Incremental Configurations

Listing A.11 shows that FAMER linking is capable of keeping already existing edges of the grouped similarity graph (through ”keepCurrentEdges”) and not recompute the similarity of them (through ”recomputeSimilarityForCurrentEdges”). The rest of linking configuration is the same as batch linking configuration. Each task configuration is explained in Section 3.3.

Listing A.12 specifies the configuration for the repairing method *MBM* as an example. Beside the required parameters of the repairing method, the ”clusterIdPrefix” prevents

[1] https://git.informatik.uni-leipzig.de/dbs/FAMER/-/wikis/Home/Configuration/Clustering-Configuration-(JSON)

Listing A.11: FAMER incremental linking configuration

```
{
   "keepCurrentEdges":"true",
   "recomputeSimilarityForCurrentEdges":"false",
   "blockingComponents":[
        ...
   ],
   "similarityComponents":[
        ...
   ],
   "selectionComponent":{
        ...
   }
}
```

Listing A.12: FAMER incremental repairing configuration

```
{
    "repairingMethod":"MBM",
    "delta":"0.0",
    "clusterIdPrefix":"mbm",
    "isSCRemoving":"false",
    "clustering": {
        "clusteringMethod":"CONNECTED_COMPONENTS",
        "clusteringOutputType":"GRAPH",
        "maxIteration":"MAX_VALUE"
    }
}
```

creating repetitive cluster ids that has been generated in the previous incremental repairing tasks.

B

MSCD-AP Quality Results

Figure B.1 shows the precision, recall and f-measure results for DS-P1. Similar to the results depicted in Chapter 5 (Figure 5.7), the f-measure of the proposed approach MSCD-AP competes with the CLIP approach due to the high precision.

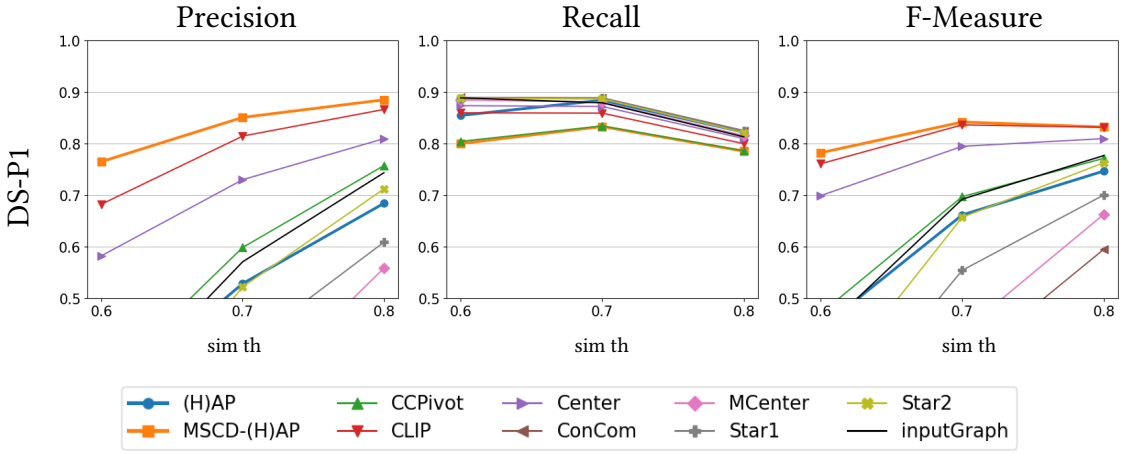


Figure B.1: MSCD-AP evaluation for DS-P1 dataset

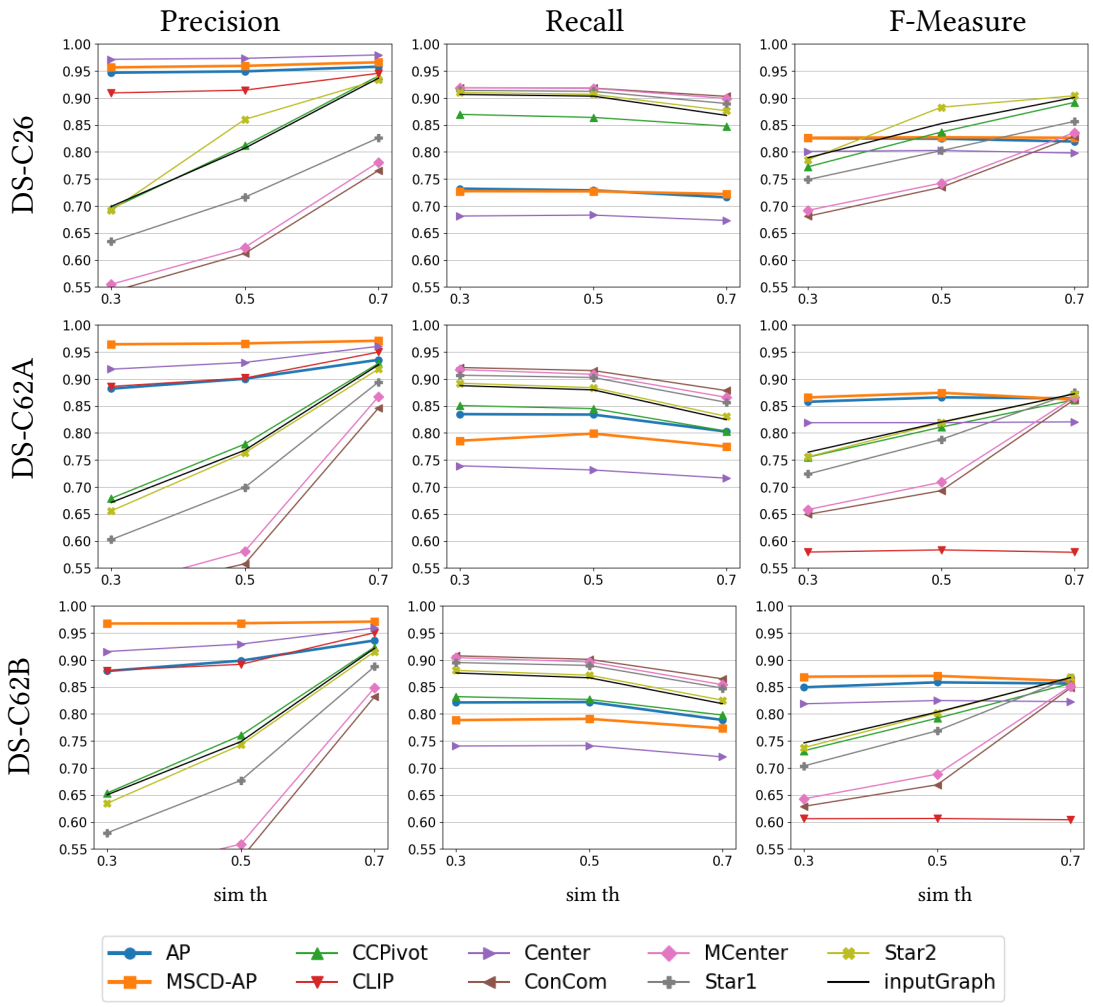


Figure B.2: MSCD-AP evaluation for camera datasets

Figure B.2 shows the precision, recall and f-measure results for DS-C26, DS-C62A and DS-C62B. Similar to the results depicted in Chapter 5 (Figure 5.8), for DS-C26 that the ratio of clean sources is too little, MSCD-AP is identical to AP while for DS-62A and DS-C62B it achieves the best F-Measure.

C

MSCD-HAC Quality Results

Figure C.2 confirms the conclusions of Chapter 5 (Figure 5.11). For the clean dataset DS-P1, MSCD S-LINK competes with CLIP approach which is tailored to clustering clean sources.

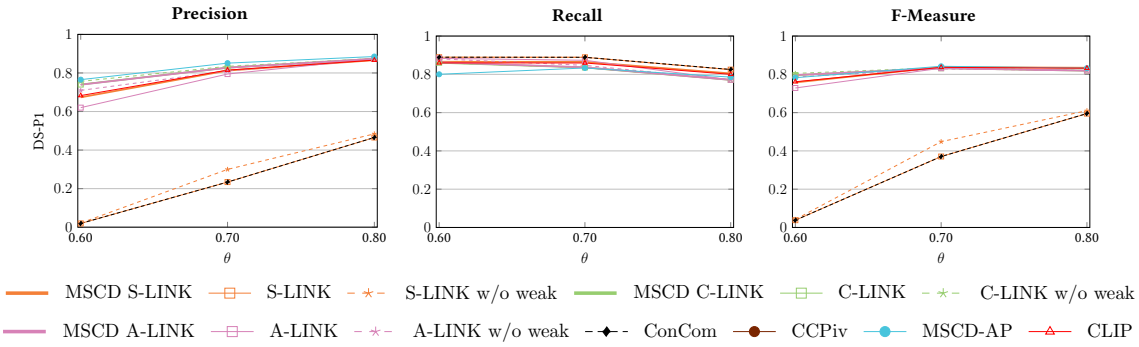


Figure C.1: MSCD-HAC evaluation for DS-P1 dataset

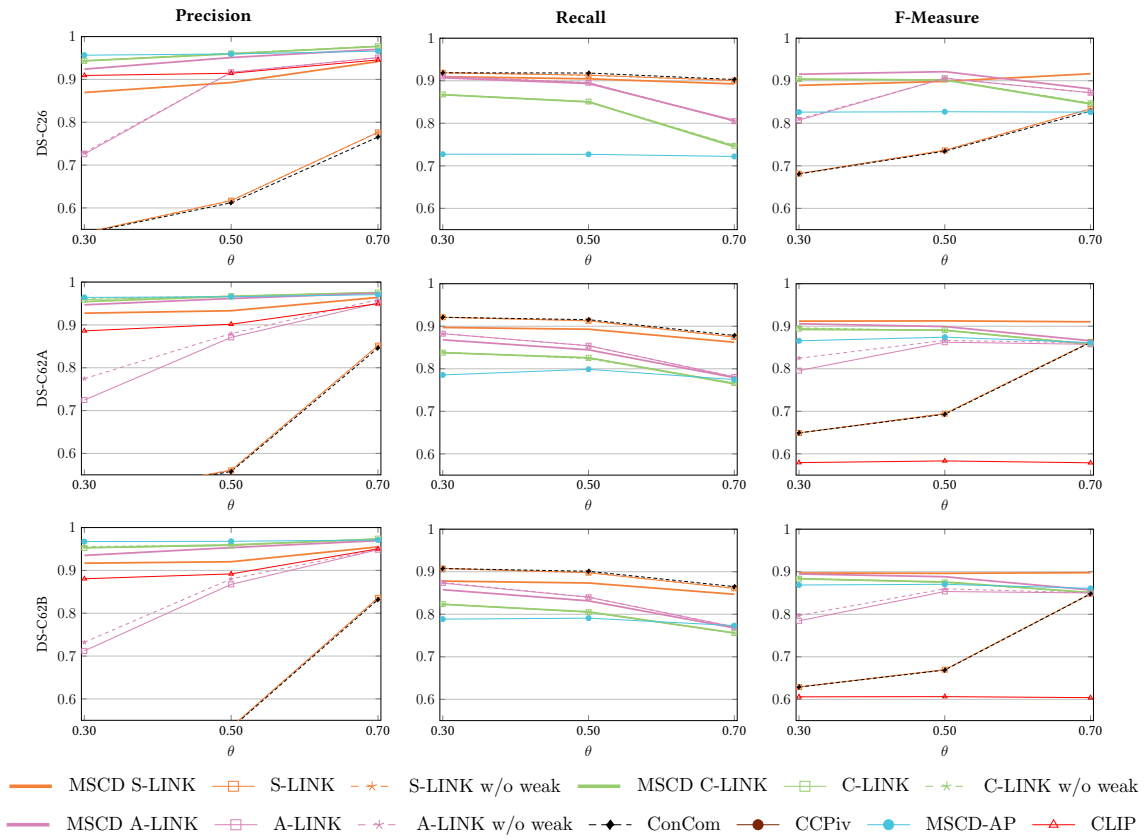


Figure C.2: MSCD-HAC evaluation for camera datasets

Figure C.2 shows the quality results for DS-C26, DS-C62A, and DS-C62B. As concluded in Chapter 5 (Figure 5.11), MSCD S-LINK obtains better precision while keeping the same recall as the ratio of clean sources increases.

Bibliography

[1] URL: https://github.com/scify/JedAIToolkit (visited on 02/07/2021).

[2] URL: https://hadoop.apache.org/docs/r1.2.1/mapred_tutorial.html (visited on 04/03/2021).

[3] Nir Ailon, Moses Charikar, and Alantha Newman. "Aggregating Inconsistent Information: Ranking and Clustering". In: *J. ACM* 55.5 (2008), 23:1–23:27. DOI: 10.1145/1411509.1411513.

[4] Javed A. Aslam, Ekaterina Pelekhov, and Daniela Rus. "The Star Clustering Algorithm for Static and Dynamic Information Organization". In: *J. Graph Algorithms Appl.* 8 (2004), pp. 95–129. DOI: 10.7155/jgaa.00084.

[5] Daniel Ayala, Inma Hernández, David Ruiz, and Erhard Rahm. "LEAPME: Learning-based Property Matching with Embeddings". In: *CoRR* abs/2010.01951 (2020).

[6] Eric Bair. "Semi-supervised Clustering Methods". In: *CoRR* abs/1307.0252 (2013). arXiv: 1307.0252.

[7] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. "Correlation Clustering". In: *43rd Symposium on Foundations of Computer Science (FOCS 2002), 16-19 November 2002, Vancouver, BC, Canada, Proceedings*. IEEE Computer Society, 2002, p. 238. DOI: 10.1109/SFCS.2002.1181947.

[8] Nikhil Bansal, Avrim Blum, and Shuchi Chawla. "Correlation Clustering". In: *Mach. Learn.* 56.1-3 (2004), pp. 89–113. DOI: 10.1023/B:MACH.0000033116.57574.95.

[9] Nilesh Bansal, Fei Chiang, Nick Koudas, and Frank Wm. Tompa. "Seeking Stable Clusters in the Blogosphere". In: *Proceedings of the 33rd International Conference on Very Large Data Bases, University of Vienna, Austria, September 23-27, 2007*, ed. by Christoph Koch et al. ACM, 2007, pp. 806–817. URL: http://www.vldb.org/conf/2007/papers/research/p806-bansal.pdf.

[10] Nils Barlaug and Jon Atle Gulla. "Neural Networks for Entity Matching: A Survey". In: *ACM Trans. Knowl. Discov. Data* 15.3 (2021), 52:1–52:37. DOI: 10.1145/3442200.

[11] Giuseppe Di Battista, Peter Eades, Roberto Tamassia, and Ioannis G. Tollis. *Graph Drawing: Algorithms for the Visualization of Graphs*. Prentice-Hall, 1999. ISBN: 0-13-301615-3.

[12] Kedar Bellare, Carlo Curino, Ashwin Machanavajihala, Peter Mika, Mandar Rahurkar, and Aamod Sane. "WOO: A Scalable and Multi-tenant Platform for Continuous Knowledge Base Synthesis". In: *Proc. VLDB Endow.* 6.11 (2013), 1114–1125. DOI: 10.14778/2536222.2536236.

[13] Omar Benjelloun, Hector Garcia-Molina, David Menestrina, Qi Su, Steven Euijong Whang, and Jennifer Widom. "Swoosh: A Generic Approach to Entity Resolution". In: *VLDB J.* 18.1 (2009), pp. 255–276. DOI: 10.1007/s00778-008-0098-x.

[14] Indrajit Bhattacharya and Lise Getoor. "Collective Entity Resolution In Relational Data". In: *ACM Trans. Knowl. Discov. Data* 1.1 (2007), p. 5. DOI: 10.1145/1217299.1217304.

[15] Indrajit Bhattacharya and Lise Getoor. "Entity Resolution in Graphs". In: *Mining graph data* 311 (2006). URL: https://drum.lib.umd.edu/bitstream/handle/1903/4021/4758.pdf?sequence=1.

[16] Mikhail Bilenko, Beena Kamath, and Raymond J. Mooney. "Adaptive Blocking: Learning to Scale Up Record Linkage". In: *Proceedings of the 6th IEEE International Conference on Data Mining (ICDM 2006), 18-22 December 2006, Hong Kong, China*. IEEE Computer Society, 2006, pp. 87–96. DOI: 10.1109/ICDM.2006.13.

[17] Jens Bleiholder and Felix Naumann. "Data Fusion". In: *ACM Comput. Surv.* 41.1 (2008), 1:1–1:41. DOI: 10.1145/1456650.1456651.

[18] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. "Enriching Word Vectors with Subword Information". In: *Trans. Assoc. Comput. Linguistics* 5 (2017), pp. 135–146. URL: https://transacl.org/ojs/index.php/tacl/article/view/999.

[19] Dhruba Borthakur. "The Hadoop Distributed File System: Architecture and Design". In: *Hadoop Project Website* 11.2007 (2007), p. 21.

[20] David Guy Brizan and Abdullah Uz Tansel. "A. Survey of Entity Resolution and Record Linkage Methodologies". In: *Communications of the IIMA* 6.3 (2006), p. 5. URL: https://scholarworks.lib.csusb.edu/ciima/vol6/iss3/5.

[21] Andrea Calì, Thomas Lukasiewicz, Livia Predoiu, and Heiner Stuckenschmidt. "A Framework for Representing Ontology Mappings Under Probabilities and Inconsistency". In: *Proc. URSW*. URSW'07 (2007). DOI: http://dl.acm.org/citation.cfm?id=2889833.2889835.

[22] Lucian Carata et al. "A Primer on Provenance". In: *Commun. ACM* 57.5 (2014), pp. 52–60. DOI: 10.1145/2596628.

[23] Paris Carbone, Asterios Katsifodimos, Stephan Ewen, Volker Markl, Seif Haridi, and Kostas Tzoumas. "Apache Flink™: Stream and Batch Processing in a Single Engine". In: *IEEE Data Eng. Bull.* 38.4 (2015), pp. 28–38. URL: http://sites.computer.org/debull/A15dec/p28.pdf.

[24] Tianqi Chen and Carlos Guestrin. "XGBoost: A Scalable Tree Boosting System". In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, ed. by Balaji Krishnapuram et al. ACM, 2016, pp. 785–794. DOI: 10.1145/2939672.2939785.

[25] Yueh-Hsuan Chiang. "Towards Large-scale Temporal Entity Matching". PhD thesis. The University of Wisconsin-Madison, 2013.

[26] Flavio Chierichetti, Nilesh N. Dalvi, and Ravi Kumar. "Correlation Clustering in MapReduce". In: *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, ed. by Sofus A. Macskassy et al. ACM, 2014, pp. 641–650. DOI: 10.1145/2623330.2623743.

[27] Peter Christen. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Data-Centric Systems and Applications. Springer, 2012. ISBN: 978-3-642-31163-5. DOI: 10.1007/978-3-642-31164-2.

[28] Peter Christen. "Febrl -: An Open Source Data Cleaning, Deduplication and Record Linkage System with a Graphical User Interface". In: *Proceedings of the 14th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Las Vegas, Nevada, USA, August 24-27, 2008*, ed. by Ying Li et al. ACM, 2008, pp. 1065–1068. DOI: 10.1145/1401890.1402020.

[29] Peter Christen and Ross W. Gayler. "Adaptive Temporal Entity Resolution on Dynamic Databases". In: *Advances in Knowledge Discovery and Data Mining, 17th Pacific-Asia Conference, PAKDD 2013, Gold Coast, Australia, April 14-17, 2013, Proceedings, Part II*, ed. by Jian Pei et al. Vol. 7819. Lecture Notes in Computer Science. Springer, 2013, pp. 558–569. DOI: 10.1007/978-3-642-37456-2_47.

[30] Peter Christen, Ross W. Gayler, and David Hawking. "Similarity-aware Indexing for Real-time Entity Resolution". In: *Proceedings of the 18th ACM Conference on Information and Knowledge Management, CIKM 2009, Hong Kong, China, November 2-6, 2009*, ed. by David Wai-Lok Cheung et al. ACM, 2009, pp. 1565–1568. DOI: 10.1145/1645953.1646173.

[31] Peter Christen and Dinusha Vatsalan. "Flexible and Extensible Generation and Corruption of Personal Data". In: *22nd ACM International Conference on Information and Knowledge Management, CIKM'13, San Francisco, CA, USA, October 27 - November 1, 2013*, ed. by Qi He et al. ACM, 2013, pp. 1165–1168. DOI: 10.1145/2505515.2507815.

[32] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. "An Overview of End-to-End Entity Resolution for Big Data". In: *ACM Comput. Surv.* 53.6 (2021), 127:1–127:42. DOI: 10.1145/3418896.

[33] Xu Chu, Ihab F. Ilyas, and Paraschos Koutris. "Distributed Data Deduplication". In: *Proc. VLDB Endow.* 9.11 (2016), pp. 864–875. DOI: 10.14778/2983200.2983203.

[34] Gianni Costa, Giuseppe Manco, and Riccardo Ortale. "An Incremental Clustering Scheme for Data De-duplication". In: *Data Min. Knowl. Discov.* 20.1 (2010), pp. 152–187. DOI: 10.1007/s10618-009-0155-0.

[35] Hong Cui, Jingjing Zhang, Chunfeng Cui, and Qinyu Chen. "Solving Large-scale Assignment Problems by Kuhn-Munkres Algorithm". In: *Proceedings of the 2nd International Conference on Advances in Mechanical Engineering and Industrial Informatics (AMEII 2016)*. Atlantis Press, 2016, pp. 822–827. ISBN: 978-94-6252-188-9. DOI: https://doi.org/10.2991/ameii-16.2016.160.

[36] Elias Dahlhaus. "Parallel Algorithms for Hierarchical Clustering and Applications to Split Decomposition and Parity Graph Recognition". In: *J. Algorithms* 36.2 (2000), pp. 205–240. DOI: 10.1006/jagm.2000.1090.

[37] Manoranjan Dash, Simona Petrutiu, and Peter Scheuermann. “pPOP: Fast yet accurate parallel hierarchical clustering using partitioning”. In: *Data Knowl. Eng.* 61.3 (2007), pp. 563–578. DOI: 10.1016/j.datak.2006.07.004.

[38] Lucie David. “A Distributed Hierarchical Clustering Algorithm for Multi-source Entity Resolution”. Bachelor’s Thesis. University of Leipzig, 2021. URL: https://dbs.uni-leipzig.de/file/Bachelorarbeit_LucieDavid3752995_ElektronischeFassung.pdf.

[39] Jeffrey Dean and Sanjay Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters”. In: *6th Symposium on Operating System Design and Implementation (OSDI 2004), San Francisco, California, USA, December 6-8, 2004*. 2004, pp. 137–150. URL: http://www.usenix.org/events/osdi04/tech/dean.html.

[40] AnHai Doan, Alon Y. Halevy, and Zachary G. Ives. *Principles of Data Integration*. Morgan Kaufmann, 2012. ISBN: 978-0-12-416044-6. URL: http://research.cs.wisc.edu/dibook/.

[41] AnHai Doan et al. “Magellan: Toward Building Ecosystems of Entity Matching Solutions”. In: *Commun. ACM* 63.8 (2020), pp. 83–91. DOI: 10.1145/3405476.

[42] Ugur Dogrusöz, Erhan Giral, Ahmet Cetintas, Ali Civril, and Emek Demir. “A Layout Algorithm for Undirected Compound Graphs”. In: *Inf. Sci.* 179.7 (2009), pp. 980–994. DOI: 10.1016/j.ins.2008.11.017.

[43] Xin Luna Dong and Divesh Srivastava. “Big Data Integration”. In: *Proc. VLDB Endow.* 6.11 (2013), pp. 1188–1189. DOI: 10.14778/2536222.2536253.

[44] Uwe Draisbach, Peter Christen, and Felix Naumann. “Transforming Pairwise Duplicates to Entity Clusters for High-quality Duplicate Detection”. In: *ACM J. Data Inf. Qual.* 12.1 (2020), 3:1–3:30. DOI: 10.1145/3352591.

[45] Muhammad Ebraheem, Saravanan Thirumuruganathan, Shafiq R. Joty, Mourad Ouzzani, and Nan Tang. “Distributed Representations of Tuples for Entity Resolution”. In: *Proc. VLDB Endow.* 11.11 (2018), pp. 1454–1467. DOI: 10.14778/3236187.3236198.

[46] Adil Fahad et al. “A Survey of Clustering Algorithms for Big Data: Taxonomy and Empirical Analysis”. In: *IEEE Trans. Emerg. Top. Comput.* 2.3 (2014), pp. 267–279. DOI: 10.1109/TETC.2014.2330519.

[47] Ivan P Fellegi and Alan B Sunter. "A Theory for Record Linkage". In: *Journal of the American Statistical Association* 64.328 (1969), pp. 1183–1210. DOI: 10.1080/01621459.1969.10501049.

[48] Jeffrey Fisher, Peter Christen, Qing Wang, and Erhard Rahm. "A Clustering-Based Framework to Control Block Sizes for Entity Resolution". In: *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Sydney, NSW, Australia, August 10-13, 2015*, ed. by Longbing Cao et al. ACM, 2015, pp. 279–288. DOI: 10.1145/2783258.2783396.

[49] Gary William Flake, Robert Endre Tarjan, and Kostas Tsioutsiouliklis. "Graph Clustering and Minimum Cut Trees". In: *Internet Mathematics* 1.4 (2003), pp. 385–408. DOI: 10.1080/15427951.2004.10129093.

[50] Lester Randolph Ford and Delbert R Fulkerson. "Maximal Flow Through a Network". In: *Canadian journal of Mathematics* 8 (1956), pp. 399–404.

[51] Martin Franke, Ziad Sehili, Marcel Gladbach, and Erhard Rahm. "Post-processing Methods for High Quality Privacy-Preserving Record Linkage". In: *Data Privacy Management, Cryptocurrencies and Blockchain Technology - ESORICS 2018 International Workshops, DPM 2018 and CBT 2018, Barcelona, Spain, September 6-7, 2018, Proceedings*, ed. by Joaquín García-Alfaro et al. Vol. 11025. Lecture Notes in Computer Science. Springer, 2018, pp. 263–278. DOI: 10.1007/978-3-030-00305-0_19.

[52] Michael J. Franklin, Alon Y. Halevy, and David Maier. "From Databases to Dataspaces: A New Abstraction for Information Management". In: *SIGMOD Rec.* 34.4 (2005), pp. 27–33. DOI: 10.1145/1107499.1107502.

[53] Brendan J. Frey and Delbert Dueck. "Clustering by Passing Messages Between Data Points". In: *science* 315.5814 (2007), pp. 972–976.

[54] Carol Friedman and Robert Sideli. "Tolerating Spelling Errors During Patient Validation". In: *Computers and Biomedical Research* 25.5 (1992), pp. 486–509. DOI: https://doi.org/10.1016/0010-4809(92)90005-U.

[55] Thomas M. J. Fruchterman and Edward M. Reingold. "Graph Drawing by Force-directed Placement". In: *Softw. Pract. Exper.* 21.11 (Nov. 1991), pp. 1129–1164. ISSN: 0038-0644. DOI: 10.1002/spe.4380211102.

[56] D. Gale and L. S. Shapley. "College Admissions and the Stability of Marriage". In: *Am. Math. Mon.* 120.5 (2013), pp. 386–391. DOI: 10.4169/amer.math.monthly.120.05.386.

[57] Diego García-Gil, Sergio Ramírez-Gallego, Salvador García, and Francisco Herrera. "A Comparison on Scalability for Batch Big Data Processing on Apache Spark and Apache Flink". In: *Big Data Analytics* 2.1 (Mar. 2017). DOI: 10.1186/s41044-016-0020-2.

[58] Lise Getoor and Ashwin Machanavajjhala. "Entity Resolution for Big Data". In: *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, ed. by Inderjit S. Dhillon et al. ACM, 2013, p. 1527. DOI: 10.1145/2487575.2506179.

[59] Lise Getoor and Ashwin Machanavajjhala. "Entity Resolution: Theory, Practice & Open Challenges". In: *Proc. VLDB Endow.* 5.12 (2012), pp. 2018–2019. DOI: 10.14778/2367502.2367564.

[60] Phan H Giang. "A Machine Learning Approach to Create Blocking Criteria for Record Linkage". In: *Health care management science* 18.1 (2015), pp. 93–105. DOI: 10.1007/s10729-014-9276-0.

[61] Aristides Gionis, Heikki Mannila, and Panayiotis Tsaparas. "Clustering Aggregation". In: *ACM ACM Trans. on Knowledge Discovery from Data (TKDD)* 1.1 (2007), p. 4. DOI: 10.1145/1217299.1217303.

[62] Inmar E. Givoni, Clement Chung, and Brendan J. Frey. "Hierarchical Affinity Propagation". In: *UAI 2011, Proceedings of the Twenty-Seventh Conference on Uncertainty in Artificial Intelligence, Barcelona, Spain, July 14-17, 2011*, ed. by Fábio Gagliardi Cozman et al. AUAI Press, 2011, pp. 238–246. arXiv: 1202.3722.

[63] Inmar E. Givoni and Brendan J. Frey. "A Binary Variable Model for Affinity Propagation". In: *Neural Comput.* 21.6 (2009), pp. 1589–1600. DOI: 10.1162/neco.2009.05-08-785.

[64] Inmar-Ella Givoni. "Beyond Affinity Propagation: Message Passing Algorithms for Clustering". PhD thesis. University of Toronto (Canada), 2012. URL: https://central.bac-lac.gc.ca/.item?id=NR97319&op=pdf&app=Library&oclc_number=1019492051.

[65] Nizar Grira, Michel Crucianu, and Nozha Boujemaa. "Unsupervised and Semi-supervised Clustering: A Brief Survey". In: *A review of machine learning techniques for processing multimedia content* 1 (2004), pp. 9–16.

[66] Anja Gruenheid, Xin Luna Dong, and Divesh Srivastava. "Incremental Record Linkage". In: *PVLDB* 7.9 (2014), pp. 697–708. DOI: 10.14778/2732939.2732943.

[67] Huadong Guo, Lizhe Wang, Fang Chen, and Dong Liang. "Scientific Big Data and Digital Earth". In: *Chinese science bulletin* 59.35 (2014), pp. 5066–5073.

[68] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. "Framework for Evaluating Clustering Algorithms in Duplicate Detection". In: *Proc. VLDB Endow.* 2.1 (2009), pp. 1282–1293. DOI: 10.14778/1687627.1687771.

[69] Oktie Hassanzadeh and Renée J. Miller. "Creating Probabilistic Databases from Duplicated Data". In: *VLDB J.* 18.5 (2009), pp. 1141–1166. DOI: 10.1007/s00778-009-0161-2.

[70] Oktie Hassanzadeh et al. "Discovering Linkage Points over Web Data". In: *Proc. VLDB Endow.* 6.6 (2013), pp. 444–456. DOI: 10.14778/2536336.2536345.

[71] Taher H. Haveliwala, Aristides Gionis, and Piotr Indyk. "Scalable Techniques for Clustering the Web". In: *Proceedings of the Third International Workshop on the Web and Databases, WebDB 2000, Adam's Mark Hotel, Dallas, Texas, USA, May 18-19, 2000, in conjunction with ACM PODS/SIGMOD 2000. Informal proceedings*, ed. by Dan Suciu et al. 2000, pp. 129–134. URL: http://www.research.att.com/conf/webdb2000/PAPERS/8c.ps.

[72] William Hendrix, Md. Mostofa Ali Patwary, Ankit Agrawal, Wei-keng Liao, and Alok N. Choudhary. "Parallel Hierarchical Clustering on Shared Memory Platforms". In: *19th International Conference on High Performance Computing, HiPC 2012, Pune, India, December 18-22, 2012*. IEEE Computer Society, 2012, pp. 1–9. DOI: 10.1109/HiPC.2012.6507511.

[73] Mauricio A. Hernández and Salvatore J. Stolfo. "Real-world Data is Dirty: Data Cleansing and The Merge/Purge Problem". In: *Data Min. Knowl. Discov.* 2.1 (1998), pp. 9–37. DOI: 10.1023/A:1009761603038.

[74] Mauricio A. Hernández and Salvatore J. Stolfo. "The Merge/Purge Problem for Large Databases". In: *Proceedings of the 1995 ACM SIGMOD International Conference on Management of Data, San Jose, California, USA, May 22-25, 1995*, ed. by Michael J. Carey et al. ACM Press, 1995, pp. 127–138. DOI: 10.1145/223784.223807.

[75] Melanie Herschel, Ralf Diestelkämper, and Houssem Ben Lahmar. "A Survey on Provenance: What for? What Form? What from?" In: *VLDB J.* 26.6 (2017), pp. 881–906. DOI: 10.1007/s00778-017-0486-1.

[76] Kai Hildebrandt, Fabian Panse, Niklas Wilcke, and Norbert Ritter. "Large-Scale Data Pollution with Apache Spark". In: *IEEE Trans. Big Data* 6.2 (2020), pp. 396–411. DOI: 10.1109/TBDATA.2016.2637378.

[77] Anil K. Jain. "Data Clustering: 50 Years Beyond K-means". In: *Pattern Recognit. Lett.* 31.8 (2010), pp. 651–666. DOI: 10.1016/j.patrec.2009.09.011.

[78] Chen Jin, Ruoqian Liu, Zhengzhang Chen, William Hendrix, Ankit Agrawal, and Alok N. Choudhary. "A Scalable Hierarchical Clustering Algorithm Using Spark". In: *First IEEE International Conference on Big Data Computing Service and Applications, BigDataService 2015, Redwood City, CA, USA, March 30 - April 2, 2015*. IEEE Computer Society, 2015, pp. 418–426. DOI: 10.1109/BigDataService.2015.67.

[79] Stephen C Johnson. "Hierarchical Clustering Schemes". In: *Psychometrika* 32.3 (1967), pp. 241–254. DOI: 10.1007/BF02289588.

[80] Martin Junghanns, Max Kießling, Niklas Teichmann, Kevin Gómez, André Petermann, and Erhard Rahm. "Declarative and Distributed Graph Analytics with GRADOOP". In: *Proc. VLDB Endow.* 11.12 (2018), pp. 2006–2009. DOI: 10.14778/3229863.3236246.

[81] Martin Junghanns, André Petermann, Martin Neumann, and Erhard Rahm. "Management and Analysis of Big Graph Data: Current Systems and Open Challenges". In: *Handbook of Big Data Technologies*, ed. by Albert Y. Zomaya et al. Springer, 2017, pp. 457–505. DOI: 10.1007/978-3-319-49340-4_14.

[82] 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, San Francisco, California, USA, July 1, 2016*, ed. by Akhil Arora et al. ACM, 2016, 3:1–3:8. DOI: 10.1145/2980523.2980527.

[83] Martin Junghanns, André Petermann, Niklas Teichmann, and Erhard Rahm. "The Big Picture: Understanding Large-scale Graphs Using Graph Grouping with Gradoop". In: *Datenbanksysteme für Business, Technologie und Web (BTW 2017), 17. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 6.-10. März 2017, Stuttgart, Germany, Proceedings*, ed. by Bernhard Mitschang et al. Vol. P-265. LNI. GI, 2017, pp. 629–632. URL: https://dl.gi.de/20.500.12116/674.

[84] Vasiliki Kalavri, Vladimir Vlassov, and Seif Haridi. "High-Level Programming Abstractions for Distributed Graph Processing". In: *IEEE Trans. Knowl. Data Eng.* 30.2 (2018), pp. 305–324. DOI: 10.1109/TKDE.2017.2762294.

[85] Leonard Kaufman and Peter J. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. John Wiley, 1990. ISBN: 978-0-47187876-6. DOI: 10.1002/9780470316801.

[86] Mayank Kejriwal and Daniel P. Miranker. "An Unsupervised Algorithm for Learning Blocking Schemes". In: *2013 IEEE 13th International Conference on Data Mining, Dallas, TX, USA, December 7-10, 2013*, ed. by Hui Xiong et al. IEEE Computer Society, 2013, pp. 340–349. DOI: 10.1109/ICDM.2013.60.

[87] Hisashi Koga, Tetsuo Ishibashi, and Toshinori Watanabe. "Fast Agglomerative Hierarchical Clustering Algorithm Using Locality-Sensitive Hashing". In: *Knowl. Inf. Syst.* 12.1 (2007), pp. 25–53. DOI: 10.1007/s10115-006-0027-5.

[88] Lars Kolb, Andreas Thor, and Erhard Rahm. "Dedoop: Efficient Deduplication with Hadoop". In: *Proc. VLDB Endow.* 5.12 (2012), pp. 1878–1881. DOI: 10.14778/2367502.2367527.

[89] Lars Kolb, Andreas Thor, and Erhard Rahm. "Load Balancing for MapReduce-based Entity Resolution". In: *IEEE 28th International Conference on Data Engineering (ICDE 2012), Washington, DC, USA (Arlington, Virginia), 1-5 April, 2012*, ed. by Anastasios Kementsietsidis et al. IEEE Computer Society, 2012, pp. 618–629. DOI: 10.1109/ICDE.2012.22.

[90] Lars Kolb, Andreas Thor, and Erhard Rahm. "Multi-pass Sorted Neighborhood Blocking with MapReduce". In: *Comput. Sci. Res. Dev.* 27.1 (2012), pp. 45–63. DOI: 10.1007/s00450-011-0177-x.

[91] Lars Kolb, Andreas Thor, and Erhard Rahm. "Parallel Sorted Neighborhood Blocking with MapReduce". In: *Datenbanksysteme für Business, Technologie und Web (BTW), 14. Fachtagung des GI-Fachbereichs "Datenbanken und Informationssysteme" (DBIS), 2.-4.3.2011 in Kaiserslautern, Germany*, ed. by Theo Härder et al. Vol. P-180. LNI. GI, 2011, pp. 45–64. URL: https://dl.gi.de/20.500.12116/19619.

[92] Pradap Konda et al. "Magellan: Toward Building Entity Matching Management Systems over Data Science Stacks". In: *Proc. VLDB Endow.* 9.13 (2016), pp. 1581–1584. DOI: 10.14778/3007263.3007314.

[93] Hanna Köpcke, Andreas Thor, and Erhard Rahm. "Comparative Evaluation of Entity Resolution Approaches with FEVER". In: *Proc. VLDB Endow.* 2.2 (2009), pp. 1574–1577. DOI: 10.14778/1687553.1687595.

[94] Hanna Köpcke, Andreas Thor, and Erhard Rahm. "Learning-Based Approaches for Matching Web Data Entities". In: *IEEE Internet Comput.* 14.4 (2010), pp. 23–31. DOI: 10.1109/MIC.2010.58.

[95] Frank R. Kschischang, Brendan J. Frey, and Hans-Andrea Loeliger. "Factor Graphs and the Sum-Product Algorithm". In: *IEEE Trans. Inf. Theory* 47.2 (2001), pp. 498–519. DOI: 10.1109/18.910572.

[96] Harold W. Kuhn. "The Hungarian Method for the Assignment Problem". In: *Naval research logistics quarterly* 2.1-2 (1955), pp. 83–97. DOI: 10.1002/nav.3800020109.

[97] Karen Kukich. "Techniques for Automatically Correcting Words in Text". In: *ACM Comput. Surv.* 24.4 (1992), pp. 377–439. DOI: 10.1145/146370.146380.

[98] Simon Lacoste-Julien, Konstantina Palla, Alex Davies, Gjergji Kasneci, Thore Graepel, and Zoubin Ghahramani. "SIGMa: Simple Greedy Matching for Aligning Large Knowledge Bases". In: *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2013, Chicago, IL, USA, August 11-14, 2013*, ed. by Inderjit S. Dhillon et al. ACM, 2013, pp. 572–580. DOI: 10.1145/2487575.2487592.

[99] Kyong-Ha Lee, Yoon-Joon Lee, Hyunsik Choi, Yon Dohn Chung, and Bongki Moon. "Parallel Data Processing with MapReduce: A Survey". In: *SIGMOD Record* 40.4 (2011), pp. 11–20. DOI: 10.1145/2094114.2094118.

[100] Maurizio Lenzerini. "Data Integration: A Theoretical Perspective". In: *Proceedings of the Twenty-first ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems, June 3-5, Madison, Wisconsin, USA*, ed. by Lucian Popa et al. ACM, 2002, pp. 233–246. DOI: 10.1145/543613.543644.

[101] Stefan Lerm. "Verteiltes Clustering für Multi-Source Entity Resolution Gemischter Datensammlungen aus Duplikatfreien und Duplikatbehafteten Quellen". MA thesis. University of Leipzig, 2021. URL: https://dbs.uni-leipzig.de/file/Masterarbeit_Stefan_Lerm_zweiseitig.pdf.

[102] Stefan Lerm, Alieh Saeedi, and Erhard Rahm. "Extended Affinity Propagation Clustering for Multi-source Entity Resolution". In: *Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings*, ed. by Kai-Uwe Sattler et al. Vol. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, pp. 217–236. DOI: 10.18420/btw2021-11.

[103] Jure Leskovec, Anand Rajaraman, and Jeffrey David Ullman. *Mining of massive data sets*. Cambridge university press, 2020.

[104] Jinfeng Li et al. "A Comparison of General-purpose Distributed Systems for Data Processing". In: *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*. 2016, pp. 378–383. DOI: 10.1109/BigData.2016.7840626.

[105] Jixue Liu, Selasi Kwashie, Jiuyong Li, Lin Liu, and Michael Bewong. "Linking Graph Entities with Multiplicity and Provenance". In: *CoRR* abs/1908.04464 (2019). arXiv: 1908.04464.

[106] Xiaonan Liu, Meijuan Yin, Junyong Luo, and Wuping Chen. "An Improved Affinity Propagation Clustering Algorithm for Large-scale Data Sets". In: *Ninth International Conference on Natural Computation, ICNC 2013, Shenyang, China, July 23-25, 2013*, ed. by Haiying Wang et al. IEEE, 2013, pp. 894–899. DOI: 10.1109/ICNC.2013.6818103.

[107] Yongtao Ma and Thanh Tran. "TYPiMatch: Type-specific Unsupervised Learning of Keys and Key Values for Heterogeneous Web Data Integration". In: *Sixth ACM International Conference on Web Search and Data Mining, WSDM 2013, Rome, Italy, February 4-8, 2013*, ed. by Stefano Leonardi et al. ACM, 2013, pp. 325–334. DOI: 10.1145/2433396.2433439.

[108] Abdullah-Al Mamun, Robert Aseltine, and Sanguthevar Rajasekaran. "Efficient Record Linkage Algorithms Using Complete Linkage Clustering". In: *PloS one* 11.4 (2016), e0154446. DOI: 10.1371/journal.pone.0154446.

[109] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. "Distributed Representations of Words and Phrases and Their Compositionality". In: *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, ed. by Christopher J. C. Burges et al. 2013, pp. 3111–3119. URL: https://proceedings.neurips.cc/paper/2013/hash/9aa42b31882ec039965f3c4923ce901b-Abstract.html.

[110] Alvaro E. Monge. "Matching Algorithms within a Duplicate Detection System". In: *IEEE Data Eng. Bull.* 23.4 (2000), pp. 14–20. URL: http://sites.computer.org/debull/A00DEC-CD.pdf.

[111] Alvaro E. Monge and Charles Elkan. "The Field Matching Problem: Algorithms and Applications". In: *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA*, ed. by Evangelos Simoudis et al. AAAI Press, 1996, pp. 267–270. URL: http://www.aaai.org/Library/KDD/1996/kdd96-044.php.

[112] Erwan Moreau, François Yvon, and Olivier Cappé. "Robust Similarity Measures for Named Entities Matching". In: *COLING 2008, 22nd International Conference on Computational Linguistics, Proceedings of the Conference, 18-22 August 2008, Manchester, UK*, ed. by Donia Scott et al. 2008, pp. 593–600. URL: https://www.aclweb.org/anthology/C08-1075/.

[113] Sidharth Mudgal et al. "Deep Learning for Entity Matching: A Design Space Exploration". In: *Proceedings of the 2018 International Conference on Management of Data, SIGMOD Conference 2018, Houston, TX, USA, June 10-15, 2018*, ed. by Gautam Das et al. ACM, 2018, pp. 19–34. DOI: 10.1145/3183713.3196926.

[114] James Munkres. "Algorithms for the Assignment and Transportation Problems". In: *Journal of the society for industrial and applied mathematics* 5.1 (1957), pp. 32–38. URL: https://www.jstor.org/stable/2098689.

[115] Fionn Murtagh. "A Survey of Recent Advances in Hierarchical Clustering Algorithms". In: *Comput. J.* 26.4 (1983), pp. 354–359. DOI: 10.1093/comjnl/26.4.354.

[116] Fionn Murtagh and Pedro Contreras. "Algorithms for Hierarchical Clustering: An Overview". In: *Wiley Interdiscip. Rev. Data Min. Knowl. Discov.* 2.1 (2012), pp. 86–97. DOI: 10.1002/widm.53.

[117] Fionn Murtagh and Pedro Contreras. "Methods of Hierarchical Clustering". In: *CoRR* abs/1105.0121 (2011). arXiv: 1105.0121.

[118] Dimas C. Nascimento, Carlos Eduardo Santos Pires, and Demetrio Gomes Mestre. "Heuristic-based Approaches for Speeding up Incremental Record Linkage". In: *J. Syst. Softw.* 137 (2018), pp. 335–354. DOI: 10.1016/j.jss.2017.11.074.

[119] Felix Naumann and Melanie Herschel. *An Introduction to Duplicate Detection*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2010. DOI: 10.2200/S00262ED1V01Y201003DTM003.

[120] Markus Nentwig, Anika Groß, Maximilian Möller, and Erhard Rahm. "Distributed Holistic Clustering on Linked Data". In: *On the Move to Meaningful Internet Systems. OTM 2017 Conferences - Confederated International Conferences: CoopIS, C&TC, and ODBASE 2017, Rhodes, Greece, October 23-27, 2017, Proceedings, Part II*, ed. by Hervé Panetto et al. Vol. 10574. Lecture Notes in Computer Science. Springer, 2017, pp. 371–382. DOI: 10.1007/978-3-319-69459-7_25.

[121] Markus Nentwig, Anika Groß, and Erhard Rahm. "Holistic Entity Clustering for Linked Data". In: *IEEE International Conference on Data Mining Workshops, ICDM Workshops 2016, December 12-15, 2016, Barcelona, Spain*, ed. by Carlotta Domeniconi et al. IEEE Computer Society, 2016, pp. 194–201. DOI: 10.1109/ICDMW.2016.0035.

[122] Markus Nentwig and Erhard Rahm. "Incremental Clustering on Linked Data". In: *2018 IEEE International Conference on Data Mining Workshops, ICDM Workshops, Singapore, Singapore, November 17-20, 2018*, ed. by Hanghang Tong et al. IEEE, 2018, pp. 531–538. DOI: 10.1109/ICDMW.2018.00084.

[123] Howard B Newcombe, James M Kennedy, SJ Axford, and Allison P James. "Automatic Linkage of Vital Records". In: *Science* 130.3381 (1959), pp. 954–959.

[124] Howard B. Newcombe and James M. Kennedy. "Record Linkage: Making Maximum Use of the Discriminating Power of Identifying Information". In: *Commun. ACM* 5.11 (1962), pp. 563–566. DOI: 10.1145/368996.369026.

[125] Axel-Cyrille Ngonga Ngomo and Sören Auer. "LIMES - A Time-Efficient Approach for Large-Scale Link Discovery on the Web of Data". In: *IJCAI 2011, Proceedings of the 22nd International Joint Conference on Artificial Intelligence, Barcelona, Catalonia, Spain, July 16-22, 2011*, ed. by Toby Walsh. IJCAI/AAAI, 2011, pp. 2312–2317. DOI: 10.5591/978-1-57735-516-8/IJCAI11-385.

[126] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. "EAGLE: Efficient Active Learning of Link Specifications Using Genetic Programming". In: *The Semantic Web: Research and Applications - 9th Extended Semantic Web Conference, ESWC 2012, Heraklion, Crete, Greece, May 27-31, 2012. Proceedings*, ed. by Elena Simperl et al. Vol. 7295. Lecture Notes in Computer Science. Springer, 2012, pp. 149–163. DOI: 10.1007/978-3-642-30284-8_17.

[127] Axel-Cyrille Ngonga Ngomo, Mohamed Ahmed Sherif, and Klaus Lyko. "Unsupervised Link Discovery Through Knowledge Base Repair". In: *Proc. ESWC* (2014),

ed. by Valentina Presutti et al., pp. 380–394. DOI: 10.1007/978-3-319-07443-6_26.

[128] Frank Nielsen. *Introduction to HPC with MPI for Data Science*. Undergraduate Topics in Computer Science. Springer, 2016. ISBN: 978-3-319-21902-8. DOI: 10.1007/978-3-319-21903-5.

[129] Jordi Nin, Victor Muntés-Mulero, Norbert Martínez-Bazan, and Josep Lluís Larriba-Pey. "On the Use of Semantic Blocking Techniques for Data Cleansing and Integration". In: *Eleventh International Database Engineering and Applications Symposium (IDEAS 2007), September 6-8, 2007, Banff, Alberta, Canada*, ed. by Bipin C. Desai et al. IEEE Computer Society, 2007, pp. 190–198. DOI: 10.1109/IDEAS.2007.4318104.

[130] Natalya Fridman Noy, Yuqing Gao, Anshu Jain, Anant Narayanan, Alan Patterson, and Jamie Taylor. "Industry-scale Knowledge Graphs: Lessons and Challenges". In: *Commun. ACM* 62.8 (2019), pp. 36–43. DOI: 10.1145/3331166.

[131] Daniel Obraczka, Alieh Saeedi, and Erhard Rahm. "Knowledge Graph Completion with FAMER (DI2KG Challenge Winner)". In: *Proceedings of the 1st International Workshop on Challenges and Experiences from Data Integration to Knowledge Graphs co-located with the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining (KDD 2019), Anchorage, Alaska, August 5, 2019*, ed. by Donatella Firmani et al. Vol. 2512. CEUR Workshop Proceedings. CEUR-WS.org, 2019. URL: http://ceur-ws.org/Vol-2512/paper1.pdf.

[132] Daniel Obraczka, Jonathan Schuchart, and Erhard Rahm. "EAGER: Embedding-Assisted Entity Resolution for Knowledge Graphs". In: *CoRR* abs/2101.06126 (2021). arXiv: 2101.06126.

[133] Benjamin Okner. "Data Matching and Merging: An Overview". In: *Annals of Economic and Social Measurement, Volume 3, number 2*. NBER, 1974, pp. 347–352. URL: https://www.nber.org/system/files/chapters/c10114/c10114.pdf.

[134] Sarah Oppold and Melanie Herschel. "Provenance for Entity Resolution". In: *Provenance and Annotation of Data and Processes - 7th International Provenance and Annotation Workshop, IPAW 2018, London, UK, July 9-10, 2018, Proceedings*, ed. by Khalid Belhajjame et al. Vol. 11017. Lecture Notes in Computer Science. Springer, 2018, pp. 226–230. DOI: 10.1007/978-3-319-98379-0_25.

[135] Xinghao Pan, Dimitris S. Papailiopoulos, Samet Oymak, Benjamin Recht, Kannan Ramchandran, and Michael I. Jordan. "Parallel Correlation Clustering on Big Graphs". In: *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, ed. by Corinna Cortes et al. 2015, pp. 82–90. arXiv: 1507.05086.

[136] George Papadakis, Ekaterini Ioannou, Claudia Niederée, Themis Palpanas, and Wolfgang Nejdl. "Eliminating the Redundancy in Blocking-based Entity Resolution Methods". In: *Proceedings of the 2011 Joint International Conference on Digital Libraries, JCDL 2011, Ottawa, ON, Canada, June 13-17, 2011*, ed. by Glen Newton et al. ACM, 2011, pp. 85–94. DOI: 10.1145/1998076.1998093.

[137] George Papadakis, Ekaterini Ioannou, Themis Palpanas, Claudia Niederée, and Wolfgang Nejdl. "A Blocking Framework for Entity Resolution in Highly Heterogeneous Information Spaces". In: *IEEE Trans. Knowl. Data Eng.* 25.12 (2013), pp. 2665–2682. DOI: 10.1109/TKDE.2012.150.

[138] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. *The Four Generations of Entity Resolution*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2021. DOI: 10.2200/S01067ED1V01Y202012DTM064.

[139] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. "Scaling Entity Resolution to Large, Heterogeneous Data with Enhanced Meta-blocking". In: *Proceedings of the 19th International Conference on Extending Database Technology, EDBT 2016, Bordeaux, France, March 15-16, 2016, Bordeaux, France, March 15-16, 2016*, ed. by Evaggelia Pitoura et al. OpenProceedings.org, 2016, pp. 221–232. DOI: 10.5441/002/edbt.2016.22.

[140] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. "A Survey of Blocking and Filtering Techniques for Entity Resolution". In: *CoRR* abs/1905.06167 (2019). arXiv: 1905.06167.

[141] George Papadakis et al. "Three-dimensional Entity Resolution with JedAI". In: *Information Systems* 93 (2020), p. 101565. DOI: 10.1016/j.is.2020.101565.

[142] Heiko Paulheim. "Knowledge Graph Refinement: A Survey of Approaches and Evaluation Methods". In: *Semantic Web* 8.3 (2017), pp. 489–508. DOI: 10.3233/SW-160218.

[143] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "Glove: Global Vectors for Word Representation". In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, ed. by Alessandro Moschitti et al. ACL, 2014, pp. 1532–1543. DOI: 10.3115/v1/d14-1162.

[144] Maria Pershina, Mohamed Yakout, and Kaushik Chakrabarti. "Holistic Entity Matching Across Knowledge Graphs". In: *2015 IEEE International Conference on Big Data, Big Data 2015, Santa Clara, CA, USA, October 29 - November 1, 2015*. IEEE Computer Society, 2015, pp. 1585–1590. DOI: 10.1109/BigData.2015.7363924.

[145] Catia Pesquita, Daniel Faria, Emanuel Santos, and Francisco M. Couto. "To Repair or Not to Repair: Reconciling Correctness and Coherence in Ontology Reference Alignments". In: *Proceedings of the 8th International Conference on Ontology Matching*. OM'13 (2013), pp. 13–24. URL: http://dl.acm.org/citation.cfm?id=2874493.2874495.

[146] Erhard Rahm. "The Case for Holistic Data Integration". In: *Advances in Databases and Information Systems - 20th East European Conference, ADBIS 2016, Prague, Czech Republic, August 28-31, 2016, Proceedings*. 2016, pp. 11–27. DOI: 10.1007/978-3-319-44039-2_2.

[147] Erhard Rahm and Hong Hai Do. "Data Cleaning: Problems and Current Approaches". In: *IEEE Data Eng. Bull.* 23.4 (2000), pp. 3–13. URL: http://sites.computer.org/debull/A00DEC-CD.pdf.

[148] Banda Ramadan, Peter Christen, Huizhi Liang, and Ross W. Gayler. "Dynamic Sorted Neighborhood Indexing for Real-Time Entity Resolution". In: *ACM J. Data Inf. Qual.* 6.4 (2015), 15:1–15:29. DOI: 10.1145/2816821.

[149] Vibhor Rastogi, Nilesh N. Dalvi, and Minos N. Garofalakis. "Large-Scale Collective Entity Matching". In: *Proc. VLDB Endow.* 4.4 (2011), pp. 208–218. DOI: 10.14778/1938545.1938546.

[150] David Reinsel, John Gantz, and John Rydning. *Data Age 2025 - The Digitization of the World From Edge to Core*. Tech. rep. IDC, Nov. 2018. URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[151] Marko A. Rodriguez and Peter Neubauer. "Constructions from Dots and Lines". In: *CoRR* abs/1006.2361 (2010). DOI: 10.1002/bult.2010.1720360610.

[152] Lior Rokach and Oded Maimon. "Clustering Methods". In: *The Data Mining and Knowledge Discovery Handbook*, ed. by Oded Maimon et al. Springer, 2005, pp. 321–352.

[153] Christopher Rost, Andreas Thor, and Erhard Rahm. "Temporal Graph Analysis using Gradoop". In: *Datenbanksysteme für Business, Technologie und Web (BTW 2019), 18. Fachtagung des GI-Fachbereichs „Datenbanken und Informationssysteme" (DBIS), 4.-8. März 2019, Rostock, Germany, Workshopband*, ed. by Holger Meyer et al. Vol. P-290. LNI. Gesellschaft für Informatik, Bonn, 2019, pp. 109–118. DOI: 10.18420/btw2019-ws-11.

[154] Christopher Rost et al. "Distributed Temporal Graph Analytics with GRADOOP". In: *The VLDB Journal* (2021), pp. 1–27. DOI: 10.1007/s00778-021-00667-4.

[155] M. Ali Rostami, Alieh Saeedi, Eric Peukert, and Erhard Rahm. "Interactive Visualization of Large Similarity Graphs and Entity Resolution Clusters". In: *Proceedings of the 21th International Conference on Extending Database Technology, EDBT 2018, Vienna, Austria, March 26-29, 2018.* 2018, pp. 690–693. DOI: 10.5441/002/edbt.2018.86.

[156] Alieh Saeedi, Lucie David, and Erhard Rahm. "Matching Entities from Multiple Sources with Hierarchical Agglomerative Clustering". In: *submitted to ic3k* (2021).

[157] Alieh Saeedi, Markus Nentwig, Eric Peukert, and Erhard Rahm. "Scalable Matching and Clustering of Entities with FAMER". In: *CSIMQ* 16 (2018), pp. 61–83. DOI: 10.7250/csimq.2018-16.04.

[158] Alieh Saeedi, Eric Peukert, and Erhard Rahm. "Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution". In: *Advances in Databases and Information Systems - 21st European Conference, ADBIS 2017, Nicosia, Cyprus, September 24-27, 2017, Proceedings*. 2017, pp. 278–293. DOI: 10.1007/978-3-319-66917-5_19.

[159] Alieh Saeedi, Eric Peukert, and Erhard Rahm. "Incremental Multi-source Entity Resolution for Knowledge Graph Completion". In: *The Semantic Web - 17th International Conference, ESWC 2020, Heraklion, Crete, Greece, May 31-June 4, 2020, Proceedings*, ed. by Andreas Harth et al. Vol. 12123. Lecture Notes in Computer Science. Springer, 2020, pp. 393–408. DOI: 10.1007/978-3-030-49461-2_23.

[160] Alieh Saeedi, Eric Peukert, and Erhard Rahm. "Using Link Features for Entity Clustering in Knowledge Graphs". In: *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*. 2018, pp. 576–592. DOI: 10.1007/978-3-319-93417-4_37.

[161] Seref Sagiroglu and Duygu Sinanc. "Big Data: A Review". In: *2013 International Conference on Collaboration Technologies and Systems, CTS 2013, San Diego, CA, USA, May 20-24, 2013*, ed. by Geoffrey Charles Fox et al. IEEE, 2013, pp. 42–47. DOI: 10.1109/CTS.2013.6567202.

[162] Anish Das Sarma, Xin Dong, and Alon Y. Halevy. "Bootstrapping Pay-As-You-Go Data Integration Systems". In: *Proceedings of the ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, Vancouver, BC, Canada, June 10-12, 2008*, ed. by Jason Tsong-Li Wang. ACM, 2008, pp. 861–874. DOI: 10.1145/1376616.1376702.

[163] Ziad Sehili, Florens Rohde, Martin Franke, and Erhard Rahm. "Multi-Party Privacy Preserving Record Linkage in Dynamic Metric Space". In: *Datenbanksysteme für Business, Technologie und Web (BTW 2021), 19. Fachtagung des GI-Fachbereichs, Datenbanken und Informationssysteme" (DBIS), 13.-17. September 2021, Dresden, Germany, Proceedings*, ed. by Kai-Uwe Sattler et al. Vol. P-311. LNI. Gesellschaft für Informatik, Bonn, 2021, pp. 257–278. DOI: 10.18420/btw2021-13.

[164] Hermes Senger et al. "BSP Cost and Scalability Analysis for MapReduce Operations". In: *Concurr. Comput. Pract. Exp.* 28.8 (2016), pp. 2503–2527. DOI: 10.1002/cpe.3628.

[165] Giovanni Simonini, Sonia Bergamaschi, and H. V. Jagadish. "BLAST: a Loosely Schema-aware Meta-blocking Approach for Entity Resolution". In: *Proc. VLDB Endow.* 9.12 (2016), pp. 1173–1184. DOI: 10.14778/2994509.2994533.

[166] Amit Singhal. *Introducing the Knowledge Graph: things, not strings*. 2012. URL: https://blog.google/products/search/introducing-knowledge-graph-things-not/ (visited on 05/16/2012).

[167] Andrew S. Tanenbaum and Maarten van Steen. *Distributed systems - principles and paradigms, 2nd Edition*. Pearson Education, 2007. ISBN: 978-0-13-239227-3.

[168] Jeffrey D. Ullman. "Designing Good MapReduce Algorithms". In: *XRDS* 19.1 (2012), pp. 30–34. DOI: 10.1145/2331042.2331053.

[169] Stijn Marinus Van Dongen. "Graph Clustering by Flow Simulation". PhD thesis. University of Utrecht, 2000. URL: http://dspace.library.uu.nl/handle/1874/848.

[170] Jorge Veiga, Roberto R. Expósito, Xoán C. Pardo, Guillermo L. Taboada, and Juan Touriño. "Performance Evaluation of Big Data Frameworks for Large-scale Data Analytics". In: *2016 IEEE International Conference on Big Data, BigData 2016, Washington DC, USA, December 5-8, 2016*. 2016, pp. 424–431. DOI: 10.1109/BigData.2016.7840633.

[171] Julius Volz, Christian Bizer, Martin Gaedke, and Georgi Kobilarov. "Silk - A Link Discovery Framework for the Web of Data". In: *Proceedings of the WWW2009 Workshop on Linked Data on the Web, LDOW 2009, Madrid, Spain, April 20, 2009*, ed. by Christian Bizer et al. Vol. 538. CEUR Workshop Proceedings. CEUR-WS.org, 2009. URL: http://ceur-ws.org/Vol-538/ldow2009_paper13.pdf.

[172] Hongzhi Wang, Jianzhong Li, and Hong Gao. "Efficient Entity Resolution Based on Subgraph Cohesion". In: *Knowl. Inf. Syst.* 46.2 (2016), pp. 285–314. DOI: 10.1007/s10115-015-0818-7.

[173] Qing Wang, Jingyi Gao, and Peter Christen. "A Clustering-Based Framework for Incrementally Repairing Entity Resolution". In: *Proc.PAKDD* (2016), ed. by James Bailey et al., pp. 283–295. DOI: 10.1007/978-3-319-31750-2_23.

[174] Qing Wang, Klaus-Dieter Schewe, and Woods Wang. "Provenance-Aware Entity Resolution: Leveraging Provenance to Improve Quality". In: *Database Systems for Advanced Applications - 20th International Conference, DASFAA 2015, Hanoi, Vietnam, April 20-23, 2015, Proceedings, Part I*, ed. by Matthias Renz et al. Vol. 9049. Lecture Notes in Computer Science. Springer, 2015, pp. 474–490. DOI: 10.1007/978-3-319-18120-2_28.

[175] Yandong Wang, Robin Goldstone, Weikuan Yu, and Teng Wang. "Characterization and Optimization of Memory-Resident MapReduce on HPC Systems". In: *2014 IEEE 28th International Parallel and Distributed Processing Symposium, Phoenix, AZ, USA, May 19-23, 2014*. IEEE Computer Society, 2014, pp. 799–808. DOI: 10.1109/IPDPS.2014.87.

[176] Joe H Ward Jr. "Hierarchical grouping to optimize an objective function". In: *Journal of the American statistical association* 58.301 (1963), pp. 236–244.

[177] Michael J. Welch, Aamod Sane, and Chris Drome. "Fast and Accurate Incremental Entity Resolution Relative to an Entity Knowledge Base". In: *21st ACM International Conference on Information and Knowledge Management, CIKM'12, Maui, HI, USA, October 29 - November 02, 2012*. 2012, pp. 2667–2670. DOI: 10.1145/2396761.2398719.

[178] Steven Euijong Whang, David Marmaros, and Hector Garcia-Molina. "Pay-As-You-Go Entity Resolution". In: *IEEE Trans. Knowl. Data Eng.* 25.5 (2013), pp. 1111–1124. DOI: 10.1109/TKDE.2012.43.

[179] Derry Tanti Wijaya and Stéphane Bressan. "Ricochet: A Family of Unconstrained Algorithms for Graph Clustering". In: *Database Systems for Advanced Applications, 14th International Conference, DASFAA 2009, Brisbane, Australia, April 21-23, 2009. Proceedings*, ed. by Xiaofang Zhou et al. Vol. 5463. Lecture Notes in Computer Science. Springer, 2009, pp. 153–167. DOI: 10.1007/978-3-642-00887-0_13.

[180] Moritz Wilke and Erhard Rahm. "Towards Multi-modal Entity Resolution for Product Matching". In: *32nd GI-Workshop on Foundations of Databases (Grundlagen von Datenbanken)*. 2021. URL: https://dbs.uni-leipzig.de/en/publication/title/towards_multi_modal_entity_resolution_for_product_matching.

[181] William E Winkler and Yves Thibaudeau. *An Application of the Fellegi-Sunter Model of Record Linkage to the 1990 US Decennial Census*. Tech. rep. RR1991/09, US Bureau of the Census, Washington, DC, 1991. URL: https://www.seagate.com/files/www-content/our-story/trends/files/idc-seagate-dataage-whitepaper.pdf.

[182] Dongkuan Xu and Yingjie Tian. "A Comprehensive Survey of Clustering Algorithms". In: *Annals of Data Science* 2.2 (2015), pp. 165–193. DOI: 10.1007/s40745-015-0040-1.

[183] Rui Xu and Donald C. Wunsch II. "Survey of Clustering Algorithms". In: *IEEE Trans. Neural Networks* 16.3 (2005), pp. 645–678. DOI: 10.1109/TNN.2005.845141.

[184] Yan Yan, Stephen Meyles, Aria Haghighi, and Dan Suciu. "Entity Matching in the Wild: A Consistent and Versatile Framework to Unify Data in Industrial Applications". In: *Proceedings of the 2020 International Conference on Management of Data, SIGMOD Conference 2020, online conference [Portland, OR, USA], June 14-19,*

2020, ed. by David Maier et al. ACM, 2020, pp. 2287–2301. DOI: 10.1145/3318464.3386143.

[185] Matei Zaharia et al. "Resilient Distributed Datasets: A Fault-Tolerant Abstraction for In-Memory Cluster Computing". In: *Proceedings of the 9th USENIX Symposium on Networked Systems Design and Implementation, NSDI 2012, San Jose, CA, USA, April 25-27, 2012*, ed. by Steven D. Gribble et al. USENIX Association, 2012, pp. 15–28. URL: https://www.usenix.org/conference/nsdi12/technical-sessions/presentation/zaharia.

[186] Wei Zhang, Hao Wei, Bunyamin Sisman, Xin Luna Dong, Christos Faloutsos, and David Page. "AutoBlock: A Hands-off Blocking Framework for Entity Matching". In: *WSDM '20: The Thirteenth ACM International Conference on Web Search and Data Mining, Houston, TX, USA, February 3-7, 2020*, ed. by James Caverlee et al. ACM, 2020, pp. 744–752. DOI: 10.1145/3336191.3371813.

[187] Xiangliang Zhang. "Contributions to Large Scale Data Clustering and Streaming with Affinity Propagation. Application to Autonomic Grids". PhD thesis. PARIS: University PARIS-SUD, 2010. URL: https://www.lri.fr/~marc/thesesTAO/ZhangPhD.pdf.

[188] Chen Zhao and Yeye He. "Auto-EM: End-to-end Fuzzy Entity-Matching Using Pre-trained Deep Models and Transfer Learning". In: *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, ed. by Ling Liu et al. ACM, 2019, pp. 2413–2424. DOI: 10.1145/3308558.3313578.