Incremental Multi-source Entity Resolution for Knowledge Graph Completion

Alieh Saeedi, Eric Peukert, and Erhard Rahm

University of Leipzig & ScaDS.AI Dresden/Leipzig, Germany

Abstract. We present and evaluate new methods for incremental entity resolution as 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.

1 Introduction

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 [1]. 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 [2].

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 [3,4]. 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 our framework FAMER that supports a parallel ER for entities from multiple sources [5]. 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.

We make the following contributions:

- We propose 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.
- We propose 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.
- We provide parallel implementations of all methods for improved runtimes and high scalability to large datasets.
- We evaluate 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 [6] and with batch ER.

After a discussion of related work, we give an overview of the incremental methods within FAMER in Section 3. Section 4 presents the new methods in detail and Section 5 is the evaluation.

2 Related Work

ER and link discovery have been widely investigated and are the subject of several books and surveys [7][8][9][10][11]. Most previous approaches are static and focus on either finding duplicates in a single source or binary matching between entities of two sources. A few studies investigate multi-source ER and clustering [12][5][4][13].

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 [14][15][3] 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. [6] 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. [16] extend the approach of [6] by defining six filters to limit the number of cluster updates. The filters improve runtime but also reduce the quality. The evaluations in both [6] and [16] are limited to small single-source datasets. In our evaluation we will also consider the greedy approach of [6].

To our knowledge, there is no previous method for multi-source incremental entity clustering except the initial approach introduced in [4]. 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.

3 Overview of incremental ER with FAMER

For batch ER of multiple sources, FAMER applies the two configurable phases of *linking* and *clustering* [12][5]. The linking phase determines a similarity graph containing links between pairs of similar entities. This phase starts with blocking [8] so that only entities of the same block need to be compared with each other. Pairwise matching is typically based on the combined similarity of several properties and a threshold for the minimal similarity. The second phase uses the similarity graph to determine entity clusters. It supports several clustering schemes to group similar entities that represent the same real-world object. The most effective clustering approaches such as CLIP [5] assume duplicate-free sources so that a cluster should at most contain one entity per source. While the proposed incremental approaches are largely independent of the specific clustering scheme we analyze them here in combination with the optimized approaches

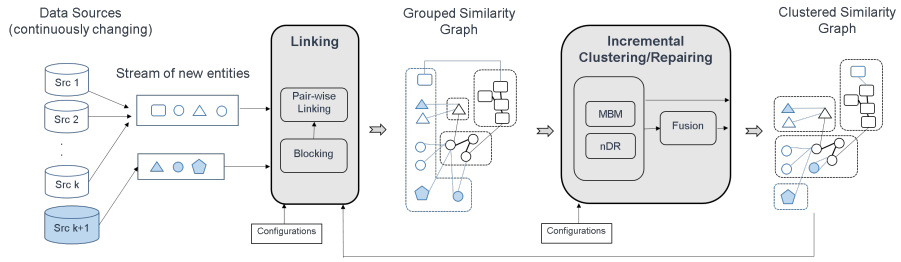


Fig. 1: FAMER workflow for incremental entity resolution

for duplicate-free data sources (assuming that dirty sources can first be deduplicated before their integration into a KG).

In this paper we propose significant extensions to FAMER for incremental ER. The corresponding workflow is indicated in Fig. 1. The approach uses a so-called *clustered similarity graph*, i.e., a similarity graph reflecting already determined clusters. 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 Fig. 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 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 details of the incremental clustering approaches are explained in Section 4.

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. The fusion approach is outlined in Section 4.2 and the impact of fusion on quality and runtime is evaluated in Section 5.

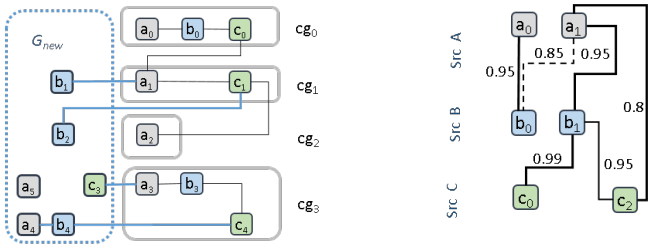
FAMER and the new approaches for incremental entity linking and clustering are implemented using the distributed execution framework Apache Flink. Hence, all match and clustering approaches can be executed in parallel on multiple machines. We will evaluate our methods for different datasets and different numbers of worker machines.

4 Incremental clustering approaches

We first define the main concepts in Section 4.1. We then describe the general incremental ER process in Section 4.2 and the base approach MB in Section 4.3. Finally, the repairing method is described in Section 4.4.

4.1 Concepts

Similarity graph: A similarity graph $\mathcal{G}$= ($\mathcal{E}$, $\mathcal{L}$) is a graph in which vertices of $\mathcal{E}$ represent entities and edges of $\mathcal{L}$ are links between similar entities. Edges have a property for the similarity value (real number in the interval [0,1]) indicating the degree of similarity. Since we assume duplicate-free sources in this paper, there are no edges between entities of the same source.
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. Fig. 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.
Cluster: A cluster has a unique cluster-id and consists of a group of entities that are meant to represent to the same real-world object. With the assumption of duplicate-free sources, we require *source-consistent* clusters, i.e. there should be at most one entity per source in a cluster so that all cluster members are from different sources.
Clustered similarity graph: A clustered similarity graph $\mathcal{CG}$ is a similarity graph $\mathcal{G}$ such that all of its entities are clustered. The same cluster-id is assigned to all vertices of the same cluster.
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 Fig. 5aa).
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 Fig. 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 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



(a) (b)

Fig. 2: a) sample grouped similarity graph b) Max-Both concept

the (n+1)-depth neighbors. For example in Fig. 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.

4.2 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.

As explained in Section 3, 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 Fig. 3a) and new entities to be integrated (shown in Fig. 3b). As typical for real-world data, the entity properties are partly erroneous. Fig. 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 Fig. 3c. 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. Fig. 4b and Fig. 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 sections 4.3 and 4.4). The output is an updated clustered graph $\mathcal{CG}_{updated}$ that includes the new entities within updated clusters.

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. Fig. 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 Fig. 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.

(a)

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

(b)

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

(c)

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

Fig. 3: Running example: existing entities, new entities and blocking

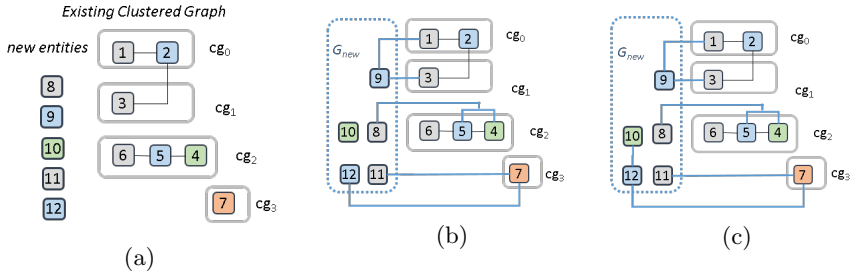


Fig. 4: a) Linking input b) w/o *new-input-linking* c) with *new-input-linking*

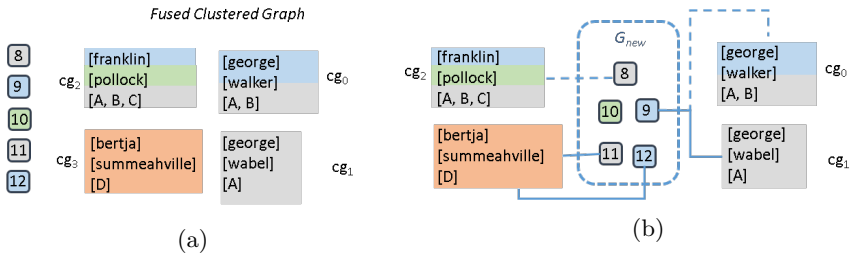


Fig. 5: a) Linking input b) linking output with fused clustered graph

4.3 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 Fig. 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.

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 Fig. 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.

Fig. 6 illustrates the max-both merge algorithm for the grouped similarity graph of Fig. 4c. The left part of the Fig. 6 shows the result after pre-clustering

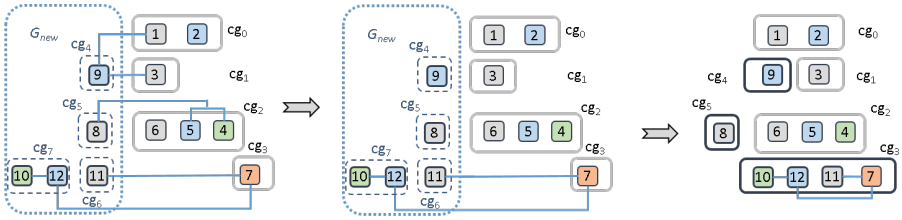


Fig. 6: Max-Both merge

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 Fig. 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 Fig. 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 (Fig. 5b). This leads to an incremental clustering result corresponding to the one for the max-both approach.

4.4 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 become 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.

Algorithm 1: 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** $\mathcal{CG}_{updated}$

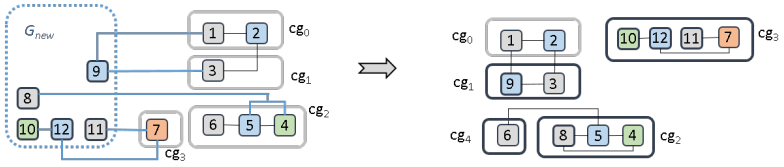


Fig. 7: 1-depth reclustering (1DR)

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 1 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 5 we used the CLIP algorithm that was shown in [5] to achieve better quality than other ER clustering approaches.

Fig. 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 Fig. 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.

Fig. 8a shows the output of Fig 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 Fig. 8b only cluster cg_1 is modified and

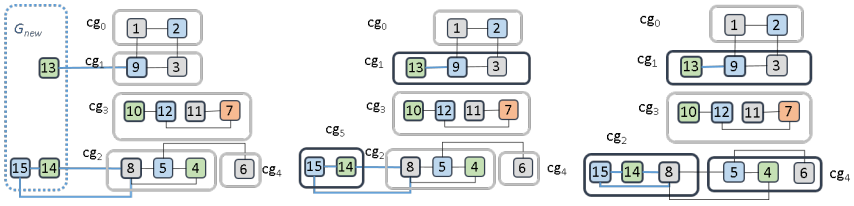


Fig. 8: a) 2nd increment input b)1DR output c) 2DR output

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 Fig. 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.

5 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 [6]. 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.

We use datasets from three domains with different numbers of duplicate-free sources. The datasets are publicly available and have been used in prior ER studies[1]. Table 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 G contains geographical real-world entities from four different data sources DBpedia (dp), Geonames (geo), Freebase (fb) and NYTimes (ny) and has already been used in the OAEI competition. For our evaluation, we focus on a subset of settlement entities as we had to manually determine the perfect clusters and thus the perfect match pairs. For the other larger evaluation datasets M and P we applied advanced data generation and corruption tools to be able to evaluate the ER quality and scalability for larger datasets and a controlled degree of corruption. M is based on real records about songs from the MusicBrainz database. We applied the DAPO data generator [17] to create five sources and duplicates for 50% of the original records in two to five sources as described for the initial evaluation of the FAMER framework [12][5]. P is based on real person records from the North-Carolina voter registry and synthetically generated duplicates using the tool GeCo [18]. We consider the configuration with 10 sources of 1 million entities each; i.e. we process up to 10 million person records.

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 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.

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

Table 1: Evaluation datasets

	general information				perfect result	
	domain	entity properties	#entity	#src	#clusters	#links
G	geography	label, longtitude, latitude	3,054	4	820	4,391
M	music	artist, title, album, year, length	19,375	5	10,000	16,250
P	persons	name, surname, suburb, postcode	10,000,000	10	6,625,848	14,995,973

For linking, we apply different configurations for each dataset (listed in Table 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.

Table 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 3: Linking configurations

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

5.1 Evaluation Results

Initially we evaluate the quality and robustness of our proposed methods for source-wise incremental ER. As described in Section 4, 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 dataset G where the four 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 [6]. In Fig. 9 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.

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 Fig. 9. 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 [6]. 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 clustering 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 Fig. 10 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 Fig. 10) 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 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.

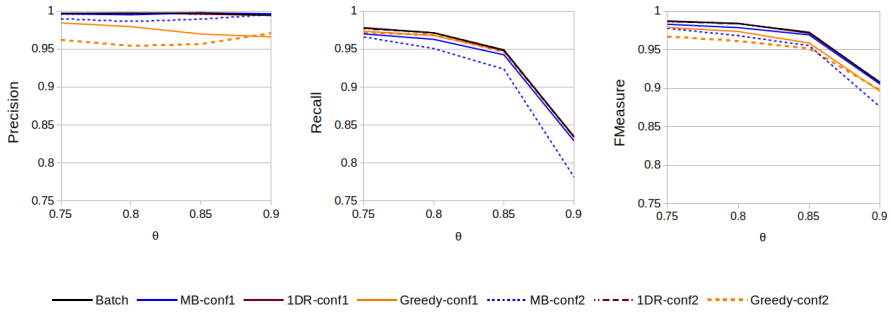


Fig. 9: Source-wise cluster quality for dataset G

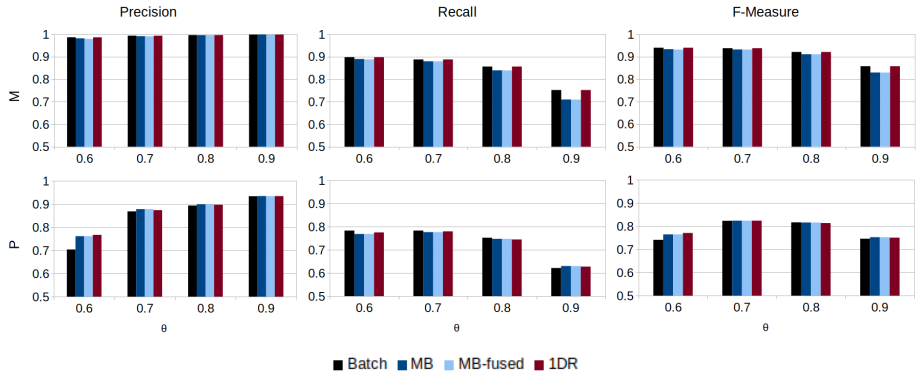


Fig. 10: Source-wise incremental ER for datasets M (1st row) and P (2nd row)

In Fig. 11 we report the F-Measure results for entity-wise incremental ER with the different increment configurations from Table 2. The results are reported for dataset M and 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 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.

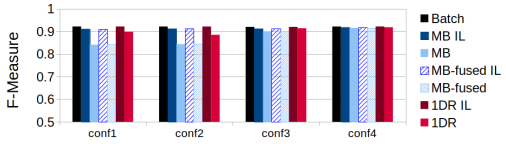


Fig. 11: F-Measure results for entity-wise incremental ER (dataset M)

5.2 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 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 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. Fig. 12 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.

Table 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

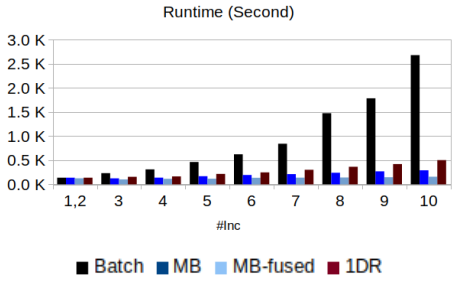


Fig. 12: Incremental runtimes

6 Conclusion and Outlook

Real-world data integration tasks such as the completion of knowledge graphs require efficient and effective incremental methods for entity resolution rather than batch-like approaches on static datasets. 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.

In future work, we plan to address further issues regarding knowledge graph completion such as the joint consideration of entities (and relationships) of different types, e.g., publications, authors and affiliations.

7 Acknowledgements

This work is partially funded by the German Federal Ministry of Education and Research under grant BMBF 01IS16026B in project ScaDS.AI Dresden/Leipzig.

References

1. Rahm E. The case for holistic data integration. In *ADBIS*. Springer, 2016.
2. Obraczka D., Saeedi A., and Rahm E. Knowledge graph completion with FAMER. In *Proc. DI2KG*, 2019.
3. Welch M., Sane A., and Drome C. Fast and accurate incremental entity resolution relative to an entity knowledge base. In *CIKM*, 2012.
4. Nentwig M. and Rahm E. Incremental clustering on linked data. In *ICDMW*. IEEE, 2018.
5. Saeedi A., Peukert E., and Rahm E. Using link features for entity clustering in knowledge graphs. In *ESWC*. Springer, 2018.
6. Gruenheid A. and et al. Incremental record linkage. *PVLDB*, 7(9), 2014.
7. Getoor L. and Machanavajjhala A. Entity resolution: theory, practice & open challenges. *PVLDB*, 5(12), 2012.
8. Christen P. *Data matching*. Springer, 2012.
9. Volz J., Bizer C., Gaedke M., and Kobilarov G. Silk-a link discovery framework for the web of data. *Ldow*, 538:53, 2009.
10. Nentwig M., Hartung M., Ngonga Ngomo A., and Rahm E. A survey of current link discovery frameworks. *Semantic Web*, 8(3), 2017.
11. Papadakis G. and et al. The return of jedai: End-to-end entity resolution for structured and semi-structured data. *PVLDB*, 11(12):1950–1953, 2018.
12. Saeedi A., Peukert E., and Rahm E. Comparative evaluation of distributed clustering schemes for multi-source entity resolution. In *ADBIS*. Springer, 2017.
13. Bellare K. and et al. Woo: A scalable and multi-tenant platform for continuous knowledge base synthesis. *PVLDB*, 6(11), 2013.
14. Benjelloun O. and et al. Swoosh: a generic approach to entity resolution. *VLDB Journal*, 18(1), 2009.
15. Costa G., Manco G., and Ortale R. An incremental clustering scheme for data de-duplication. *Data Mining and Knowledge Discovery*, 20(1), 2010.
16. do Nascimento D. and et al. Heuristic-based approaches for speeding up incremental record linkage. *Journal of Systems and Software*, 137, 2018.
17. Hildebrandt K., Panse F., Wilcke N., and Ritter N. Large-scale data pollution with Apache Spark. *IEEE Transactions on Big Data*, 2017.
18. Christen P. and Vatsalan D. Flexible and extensible generation and corruption of personal data. In *ACM CIKM*. ACM, 2013.