# Incremental Clustering on Linked Data

Markus Nentwig* and Erhard Rahm†
Department of Computer Science
Leipzig University
*nentwig@informatik.uni-leipzig.de, †rahm@informatik.uni-leipzig.de

***Abstract*—Data integration in the Web of Data is not limited to the pairwise linking of entities but often requires to cluster entities of different sources, e. g., within knowledge graphs. Such entity clustering should not only be scalable to large data volumes and many sources but also be dynamic to deal with continuously changing sources and the ability to incorporate new sources. Previous entity clustering approaches are mostly static focusing on the one-time linking and clustering of entities from few sources. In this paper, we propose and evaluate new scalable approaches for incremental entity clustering that support the continuous addition of new entities and data sources. The implementation is based on the distributed processing framework Apache Flink. A detailed performance evaluation with real and synthetically customized datasets shows the effectiveness and scalability of the incremental clustering approaches.**

***Index Terms*—Linked Data; clustering; incremental; distributed processing; Apache Flink**

## I. Introduction

Identifying and linking equivalent entities in different sources is a main challenge within the Web of Data and many approaches have been developed to address this problem [1]. In addition to a pairwise linking of sources there is an increasing need to integrate equivalent entities in a more holistic manner such that equivalent entities from arbitrary many sources are clustered together, e. g., within knowledge graphs [2]. Such a clustering facilitates the combination of the property values of all clustered entities, e. g., persons, products, cities etc., for enriched data representation.

Entity clustering is a challenging problem since the degree of semantic data heterogeneity and differences in data quality increase with the number of sources from which equivalent entities should be clustered. In addition to achieving high match and cluster quality it is also important to provide high efficiency and scalability to large data volumes and many sources. Finally, there is a strong need for dynamic or incremental entity clustering that is not limited to a one-time computation of entity clusters but that can continuously update entity clusters to cope with changing and new entities, including the incorporation of additional sources (see Fig. 1).

Previous approaches to entity clustering [3], [4] including our own ones [5], [6] are mostly static, i. e., they determine entity clusters a single time from a fixed number of static data sources. The clustering approaches use as input a set of binary `owl:sameAs` links connecting equivalent entities of different sources. These links either exist already in the Web of Data or have to be computed as a preparatory step for entity clustering. The links are commonly organized within a similarity graph

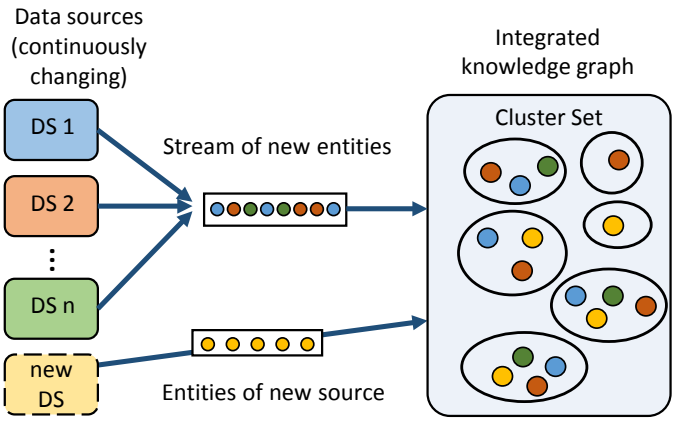



Figure 1: Incremental update of entity clusters, e. g., in knowledge graphs.

where the vertices correspond to the entities and the edges to the similarity links. Unfortunately, the overhead to determine similarity graphs and thus for static entity clustering is very high and grows quadratically with both the number and size of the sources[1]. This holds even if one exploits common performance optimizations such as blocking [7] which reduces the number of entity comparisons only by a constant factor $k$ when we partition the entities of a source into $k$ blocks (since an entity only needs to be compared with the entities of one block per source). As a result, the scalability of static entity clustering is likely limited to a smaller number of sources of moderate size.

We thus argue for the use of dynamic or incremental entity clustering that can continuously update entity clusters for new entities and new sources without having to completely recompute entity clusters. One approach could be to use a static clustering scheme to create an initial set of clusters and use a separate approach to keep the clusters up-to-date. We favor however a single dynamic approach that treats the entities of one of the sources as an initial set of clusters and incrementally adds further entities from the same or other sources similar as sketched in [2]. This approach also promises a reduced runtime compared to a static clustering of many sources since entity clusters can incrementally be created in several steps without the need of the expensive creation of large similarity graphs.

[1]For $n$ sources with $m$ entities each, we have to compare each entity of a source with $(n-1) \cdot m$ entities given a total of $\frac{n \cdot (n-1)}{2} \cdot m^2$ comparisons if we link every entity pair only once.

Specifically we make the following contributions:

- We propose incremental approaches for clustering entities from multiple sources where new entities either result in new clusters or are added to existing clusters. The addition of new entities to clusters is set-oriented so that new entities are considered together for an optimized assignment compared to the isolated addition of individual entities. Furthermore, we provide a separate approach to add entities of a single source.
- We provide scalable implementations of the incremental clustering schemes based on the distributed processing framework Apache Flink.
- We evaluate the incremental approaches for datasets of three domains in terms of cluster quality and runtime efficiency. We also provide a comparison to static entity clustering.

In Section II we define the problem and provide background information on entity clustering. In Section III we explain our proposed approach describing two strategies to realize incremental clustering which are then evaluated in Section IV. Finally, we briefly discuss related work in Section V and conclude in Section VI.

## II. Problem definition

While data sources may contain entities of many kinds we focus on clustering entities of a specific type of interest, e. g., cities, persons and music songs for our evaluation datasets. We assume further that there are no duplicates within data sources but only between data sources.

An *entity cluster* or just cluster groups a set of entities that are assumed to represent the same real-world entity. For each cluster we determine a so-called *cluster representative* fused from the properties of all entities in the cluster. Specific properties of the cluster representatives are used to determine the similarity between a new entity and an existing cluster as a basis for deciding whether the entity should be added to the cluster. This approach avoids the comparison of new entities with all members of a cluster thereby limiting the match overhead. For each cluster we also maintain the ids of the sources from where entities of the cluster originate. We call a cluster *source-consistent* if it contains at most one entity per source, otherwise source-inconsistent since it violates the assumption of duplicate-free sources.

The task of *entity clustering* has as input a set of data sources and determines and maintains a set of source-consistent and disjoint entity clusters such that all entities within a cluster match with each other and different clusters refer to different real-world objects. This entity clustering should be efficient and scalable to many sources and large data volumes.

For static entity clustering the input is a fixed set of sources and the clustering is performed only once. For dynamic or incremental clustering, however, the number and contents of data sources can continuously change so that the entity clusters are to be adapted accordingly to correctly reflect the current state of the input data. While entities of a source may be added, changed or deleted, we focus on the addition of new entities as the most complex case for maintaining entity clusters. Similarly, we only consider the addition of new sources but not their removal.

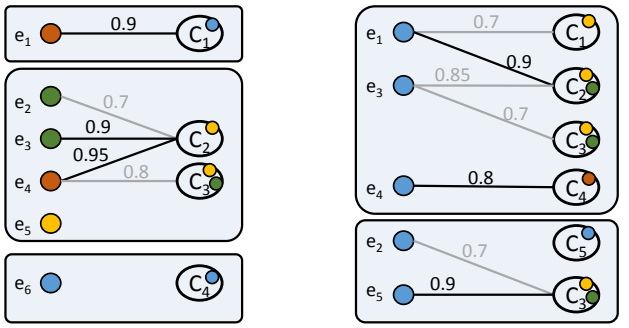
The problem of *incremental clustering* thus has as input an existing (possibly empty) set of source-consistent and disjoint entity clusters as well as a set of new entities and has to create a new set of source-consistent and disjoint entity clusters that include the new entities. In the general case, we have to find a $n$:1 assignment between the new entities and the existing clusters since an entity is added to at most one of the previously existing clusters (or to a new cluster) and since several entities (of different sources) may be added to the same cluster. For the special case where all new entities are from the same source (e. g., when a new data source is to be incorporated) we have to find a 1:1 assignment where each existing cluster is extended by at most one new entity to avoid source-inconsistent clusters.

The 1:1 assignment problem between two sets of entities has already been studied extensively in the past and approximate solutions such as stable marriage [8], Hungarian algorithm [9] and the so-called Max-Both approach [10] have been proposed for its solution. A recent study [11] has compared these alternatives and found the overall best effectiveness and runtime efficiency for Max-Both where an element $e_i$ of the first set is assigned to the best matching entity (cluster) $c_j$ of the second set only if $e_i$ is also the top match for $c_j$, e. g., there is no other entity for which the similarity with $c_j$ is higher. We will therefore follow the Max-Both approach in our source-specific incremental matching.

## III. Incremental Clustering

For assigning a new entity to a cluster we determine all cluster candidates which do not have already an entity of the respective source (to avoid source-inconsistent clusters) and for which a domain-specific similarity function $f_{\text{sim}}$ on selected properties exceeds a minimum threshold $t_{\text{min}}$. To limit the effort for finding the cluster candidates and improve the performance of our incremental clustering we apply standard blocking [7] to partition both the clusters and new entities into several blocks with a blocking function $f_{\text{blocking}}$ on selected properties. For example, person entities could be partitioned based on a fixed-length prefix of their surnames so that we only consider clusters for which the cluster representative has the same prefix than the new person entity. Blocking does not only reduce the number of comparisons but also facilitates a parallel implementation to further improve runtime since the clustering for different blocks are independent and can thus be performed in parallel.

A straight-forward approach would consider each new entity $e$ in isolation and add it to the cluster with the highest similarity above $t_{\text{min}}$ that does not yet contain another entity from $e$'s source. If no such cluster exists, a new cluster will be created for $e$. While this is a reasonable approach, it may not be the best one if there are several entities to be added at the same time. For the example in Fig. 2a, the assignment




(a) Base approach (b) Source-specific approach

Figure 2: Cluster scenarios for base and source-specific approaches (colors imply different data sources, all links are assumed exceed the minimal similarity threshold).

of entity $e_2$ to cluster $c_2$ would mean that no further entity of $e_2$'s source could be added to $c_2$ thereby preventing that the more similar entity $e_3$ is assigned to $c_2$. This problem is especially pronounced if all new entities are from the same source where we have to find a near-optimal 1:1 assignment.

In the following we will therefore present two set-based approaches for incremental entity clustering that try to avoid sub-optimal cluster assignments. We start with the general approach that applies for new entities of different sources and outline then the source-specific approach to determine a 1:1 assignment between new entities and clusters. Finally we discuss the use of Apache Flink for the distributed implementation of the approaches. We assume that the set of existing entity clusters is non-empty. This is not a limitation since the approach can easily be bootstrapped by using the entities of one of the sources as the initial clusters.

### A. Base approach

The pseudocode for set-based incremental clustering for entities of different sources is shown in Algorithm 1. It uses as input the set of new entities $\mathcal{E}_{\text{new}}$, the set of existing clusters $\mathcal{C}_{\text{exist}}$ as well as blocking function $f_{\text{blocking}}$, similarity function $f_{\text{sim}}$ and the minimum similarity threshold $t_{\text{min}}$. Initially, a cluster is created for each new entity (cluster set $\mathcal{C}_{\text{new}}$) and assigned to one of the blocks according to $f_{\text{blocking}}$.

The main processing is performed independently (in parallel) for each of the blocks with new entities. First, we compare every new entity with every cluster representative of a block using $f_{\text{sim}}$ and create candidate links $(c_{\text{new}}, c_{\text{exist}}, sim)$ within set $\mathcal{L}_i$ with all links that exceed threshold $t_{\text{min}}$ and where $c_{\text{exist}}$ does not yet contain an entity from the source of $c_{\text{new}}$. To avoid sub-optimal cluster assignments we then sort the $\mathcal{L}_i$ links and process them in descending order of their similarity so that the cluster assignments with the highest similarity can be realized first. A new entity from cluster $c_{\text{new}}$ is only added to cluster $c_{\text{exist}}$ if it does not lead to a source-inconsistent cluster. The addition to the cluster (function $add$) also involves adapting the cluster representative. Since there are generally several

**Algorithm 1:** Set-based incremental entity clustering (base approach)

**Input:** Existing clusters $\mathcal{C}_{\text{exist}}$, new entities $\mathcal{E}_{\text{new}}$, similarity function $f_{\text{sim}}$, blocking function $f_{\text{blocking}}$, minimum similarity threshold $t_{\text{min}}$
**Output:** cluster set $\mathcal{C}$

```
1  C_new ← createInitClusters(E_new, f_blocking)
2  C_exist ← addBlockingInfo(C_exist, f_blocking)
3  for block i in Parallel do
4      L_i ← getClusterCandidates(
         C_exist, C_new, f_sim, t_min)
5      L_sorted ← sortLinkSims(L_i)
6      foreach (c_new, c_exist, sim) ∈ L_sorted do
7          if c_exist ∉ C_new then
8              continue()
9          if isSrcConsistent(c_new, c_exist) then
10             c_exist.add(c_new)
11             C_new.remove(c_new)
12 return C_exist ∪ C_new
```

candidate clusters for a new entity we can ignore all further links for an assigned $c_{\text{new}}$. We achieve this by removing $c_{\text{new}}$ from the set $\mathcal{C}_{\text{new}}$ of not yet assigned entities and check in the beginning of each iteration (line 7) whether the current entity still needs to be assigned. The final result consists of the adapted cluster set $\mathcal{C}_{\text{exist}}$ as well as the singleton clusters for new entities that remain in $\mathcal{C}_{\text{new}}$.

Fig. 2a shows an example for the base approach with three blocks where the determined assignments are indicated by thick lines. We observe that entity $e_3$ is assigned to $c_2$ since it is considered before $e_2$ due of its higher similarity. The assignment of $e_2$ is not considered since its source is now already represented in $c_2$. Entity $e_4$ has two cluster candidates but after the assignment to $c_2$ it is removed from $\mathcal{C}_{\text{new}}$ so that its further cluster candidates like $c_3$ are ignored.

### B. Source-specific incremental entity clustering

We now deal with incremental clustering of entities from a single source where we determine a 1:1 assignment to the existing clusters based on the Max-Both approach. The pseudocode in Algorithm 2 uses the same input parameters than before and determines source-consistent candidate links between the new entities and the existing clusters in parallel within partitioned blocks. To implement Max-Both we determine two sets of top links ($\mathcal{L}_{\text{left}}, \mathcal{L}_{\text{right}}$) that contain in $\mathcal{L}_{\text{left}}$ for each new entity the most similar cluster and in $\mathcal{L}_{\text{right}}$ for each cluster the most similar new entity. The assignment is then only done for links that have the maximal similarity from both sides. These links can simply be determined by computing the intersection of $\mathcal{L}_{\text{left}}$ and $\mathcal{L}_{\text{right}}$ (line 11).

For the example of Fig. 2b the algorithm assigns only three entities to existing clusters for which the Max-Both condition is satisfied. Note that the base approach would have assigned entity $e_3$ to cluster $c_3$ while this assignment is not taken for Max-Both since $e_3$ has a higher similarity with $c_2$. In general, Max-Both leads thus to fewer cluster additions than the base

**Algorithm 2:** Source-specific incremental entity clustering

**Input:** Existing clusters $\mathcal{C}_{\text{exist}}$, new entities $\mathcal{E}_{\text{new}}$, similarity function $f_{\text{sim}}$, blocking function $f_{\text{blocking}}$, minimum similarity threshold $t_{\text{min}}$
**Output:** cluster set $\mathcal{C}$

```
C_new ← createInitClusters(E_new, f_blocking)
C_exist ← addBlockingInfo(C_exist, f_blocking)
for block i in Parallel do
    L ← getClusterCandidates(C_exist, C_new, f_sim, t_min)
    L_left = ∅
    L_right = ∅
    foreach c_new_i ∈ C_new do
        L_left.add(L.getBestLeftCandidate(c_new_i))
    foreach c_exist_i ∈ C_exist do
        L_right.add(L.getBestRightCandidate(c_exist_i))
    L_max-both ← L_left ∩ L_left
    foreach (c_new, c_exist, sim) ∈ L_max-both do
        c_exist.add(c_new)
        C_new.remove(c_new)
return C_exist ∪ C_new
```

approach in order to achieve a high precision with only safe cluster additions.

### C. Distributed implementation

The incremental clustering is implemented using the distributed processing framework Apache Flink [12]. Thus, execution of operations is parallelized on a shared nothing cluster while computations are based on in-memory data structures. Our implementation uses the Flink DataSet API with data transformations like filter, join, union, group-by or aggregations (relational databases) and map, flat-map and reduce (MapReduce paradigm). We also use the Gelly API for graph processing to maintain the links for cluster assignment. As mentioned we process all blocks in parallel and also support load balancing according the Block Split scheme [13] to deal with skewed block sizes and split the processing of large blocks with many entities and clusters to several machines.

## IV. Evaluation

We first describe the evaluation datasets and setup and then analyze effectiveness and runtime efficiency of the proposed incremental clustering approaches.

### A. Datasets and Setup

We use five datasets of three domains (geography, music, persons) that have already been used for static entity clustering [4], [14]. Table I gives an overview of the datasets including available properties and number of entities. The smallest dataset DS-G1 comprises real geographic data on settlements from four knowledge bases DBpedia, GeoNames, Freebase and NYTimes. The datasets for the music and person domains are based on real data which are then synthetically changed with the DaPo tool [15] to introduce corruptions of property values as well as duplicates across sources. DS-M1 and DS-M2 originate from MusicBrainz[2] while DS-P1 and DS-P2 are based on publicly available voter data from North Carolina. The largest dataset, DS-P2, has 10 sources and 10 million entities. The datasets with their perfect cluster result can be obtained from our website [3].

Table II shows the blocking and similarity functions applied in the experiments. For blocking we apply load-balanced standard blocking using a prefix of a single property (geographic and music datasets) or the concatenated prefixes of two properties (person datasets) as a blocking key. We consider prefixes of different lengths to vary the block sizes and thus the number of necessary comparisons and achievable recall. For similarity computation we mostly compute the Cosine Trigram similarity for string attributes; for the geographical entities we additionally consider the normalized geographic distance. For the music datasets we apply Cosine Trigram on the concatenation of three property values. For DS-P1 and DS-P2 we determine the average of four property similarities. For all datasets, a match requires that the computed similarity values meet a variable minimum similarity threshold $t_{\text{min}}$.

The quality of the incrementally determined entity clusters is compared with the perfect cluster results that are available for our datasets. Each cluster corresponds to a number of match links (correspondences) by assuming that each pair of entities within a cluster matches with each other. So the resulting link sets of the computed clusters and the perfect clusters are used to determine the standard metrics precision, recall and their harmonic mean, F-measure, to measure cluster quality. Table I includes the number of clusters and links of the perfect cluster result as well as the number of correct links and F-measure for the best incremental clustering configurations. Hence, the incremental clustering schemes achieve F-Measure results of more than 95% for the two smaller datasets DS-G1 and DS-M1 and of more than 80% for the larger datasets.

The experiments are carried out on a cluster with 16 workers, each of them equipped with a Intel Xeon E5-2430 6x 2.5 GHz, 48 GiB RAM, 2x 4 TiB SATA disks and 1 Gbit Ethernet connection. The machines operate on OpenSUSE 13.2 and Flink 1.5.0. All experiments are carried out three times to determine the average execution time.

### B. Experimental Results

We will now analyze the cluster quality and runtime of incremental clustering for our datasets. Mostly, we consider the incremental addition of entire sources and thus the use of source-specific incremental clustering. However, we also examine the partial addition of data sources and compare source-specific clustering with the base approach as well as with static entity clustering. Finally, we will analyze the speedup behavior w.r.t the number of workers.

*1) Effect of data source ordering:* The proposed incremental approaches are likely dependent on the order in which

[2]https://musicbrainz.org/

[3]https://dbs.uni-leipzig.de/research/projects/object_matching/famer

| | general information | | | | perfect result | | results best configuration | | |
|---|---|---|---|---|---|---|---|---|---|
| | domain | entity properties | #entities | #sources | # clusters | # links | conf($t_{\text{min}}$, bk) | # correct links | F-measure |
| DS-G1 | geography | name, longitude, latitude | 3,054 | 4 | 820 | 4,391 | conf(0.4, 1) | 4,109 | 0.983 |
| DS-M1 | music | artist, title, album, year, length | 19,375 | 5 | 10,000 | 16,250 | conf(0.5, 1) | 15,066 | 0.955 |
| DS-M2 | | | 1,937,500 | 5 | 1,000,000 | 1,624,503 | conf(0.7, 1) | 1,396,520 | 0.874 |
| DS-P1 | persons | name, surname, suburb, postcode | 5,000,000 | 5 | 3,500,840 | 3,331,384 | conf(0.7, 6) | 2,720,479 | 0.814 |
| DS-P2 | | | 10,000,000 | 10 | 6,625,848 | 14,995,973 | conf(0.7, 6) | 11,847,678 | 0.805 |

Table I: Overview of evaluation datasets.

| dataset | blocking key | similarity function |
|---|---|---|
| DS-G1 | prefixLength(name): 1 | Trigram (name) + normalized geographical distance |
| DS-M1/ DS-M2 | prefixLength: 1-5 of (artist + title + album) | Trigram (artist + title + album) |
| DS-P1/ DS-P2 | prefixLength(surname): 1-3 + prefixLength(name): 1-3 | avg(Trigram (name) + Trigram (surname) + Trigram (suburb) + Trigram (postcode)) |

Table II: Used blocking and similarity functions.

sources and entities are added to the integrated set of clusters. This is because a wrong addition of entities to clusters impacts the cluster representatives and thus all further cluster decisions. We thus evaluated different orderings for the incremental addition of data sources for our datasets. We found that the ordering had little impact for the synthetically generated music and person datasets because the sources are largely of the same size and quality.

However, for the real dataset DS-G1 we observed significant differences for different orderings. In general we could achieve very good clustering quality for this small dataset with the best F-measure of $98.4\%$ (precision $99.98\%$, recall $96.8\%$) which meets the best results for static entity clustering [6], [14]. The top result is achieved by the sequence (Freebase, GeoNames, NYTimes, DBpedia). We analyzed all possible order permutations and found that source DBPedia should not be used in the beginning since it lacks the geographical coordinates for $57\,\%$ of its records while the other sources have the coordinates for almost all entities. In Table III we show for each of the four sources the average F-measure results for all permutations where the source has either been used in the beginning (as first or second source) or at the end (source no. 3 or 4) in the sequence of added sources. We observe that the use of DBpedia in the beginning leads to much lower quality than its use towards the end (average F-measure of $94.4\,\%$ vs. $98.2\,\%$) since the missing information can lead to many wrong match decisions. By contrast, using the other sources in the beginning leads to generally better F-measure than using them last. We conclude that data quality (w.r.t. the properties used for matching) has a high impact and that incremental clustering should start with high quality data sources.

For comparison, we also used the base approach for incremental clustering rather than the source-specific approach. As expected the base approach resulted in a somewhat reduced cluster quality with $97.3\%$ F-measure for the best sequence (instead of $98.4\%$). We further investigated a more dynamic change scenario without adding entire sources. For this we initially add $80\%$ of the entities of the four sources (randomly selected) followed by two further batches of new entities each consisting of $10\%$ of the entities of the four sources. The base approach for incremental clustering achieved a F-Measure of $97.0\%$ for this scenario indicating that the partial addition of sources does not significantly reduce cluster quality.

*2) Large-scale source-specific incremental clustering:* We now analyze source-specific clustering for the larger music and person datasets (DS-M2, DS-P2) for different blocking approaches and similarity thresholds $t_{\text{min}}$. The obtained precision, recall and F-measure results are shown in Fig. 3 where the values for $t_{\text{min}}$ (x-axes) range between $0.6$ and $0.9$ and the prefix lengths for the blocking keys lie between 1 and 5 for DS-M2 and between 2 and 8 for DS-P2. As expected, precision increases and recall decreases with growing $t_{\text{min}}$ while F-Measure is relatively stable across different threshold values with $t_{\text{min}} = 0.7$ allowing for the best result. Despite the large data volume ($0.4$ to $1$ million entities per source) and the integration of 5 – 10 sources F-measure reaches relatively good values of up to $87.41\%$ for DS-M2 and up to $80.47\%$ for DS-P2.

The choice of blocking key substantially impacts cluster quality and runtime for both datasets. For DS-M2 precision is not affected by the blocking key so that it remains good even for smaller prefixes (larger blocks) probably favored by the high precision of Max-Both. As a result, the best F-measure is achieved for the smallest prefix length $B = 1$ (largest blocks) supporting the highest recall. The downside of this selection, however, is the large execution time which is hugely dependent on the chosen blocking key. As shown in Table IV the runtime for $B = 1$ is more than 5 times as high than for $B = 2$ and more than a factor 40 slower than for $B = 4$. For the larger dataset DS-P2 with 10 sources, precision suffers from larger block sizes (small prefix lengths) while recall is worst for longer prefix values. So we obtain the best precision but lowest recall for prefix length 8, while prefix length 6 allows for the best compromise value and thus the best F-Measure. For prefix length 6, we analyse the effect of the block size distribution. We therefore partition the blocks in three block size ranges: blocks with 1-100, 101-1000 and more than $1000$ entities per block. Only $0.03\,\%$ of all blocks

| Source | Freebase | | GeoNames | | NYTimes | | DBpedia | |
|---|---|---|---|---|---|---|---|---|
| Position for clustering | 1./2. | 3./4. | 1./2. | 3./4. | 1./2. | 3./4. | 1./2. | 3./4. |
| Avg. F-measure | 0.971 | 0.955 | 0.970 | 0.964 | 0.974 | 0.952 | 0.944 | 0.982 |

Table III: Effect of source ordering for DS-G1.

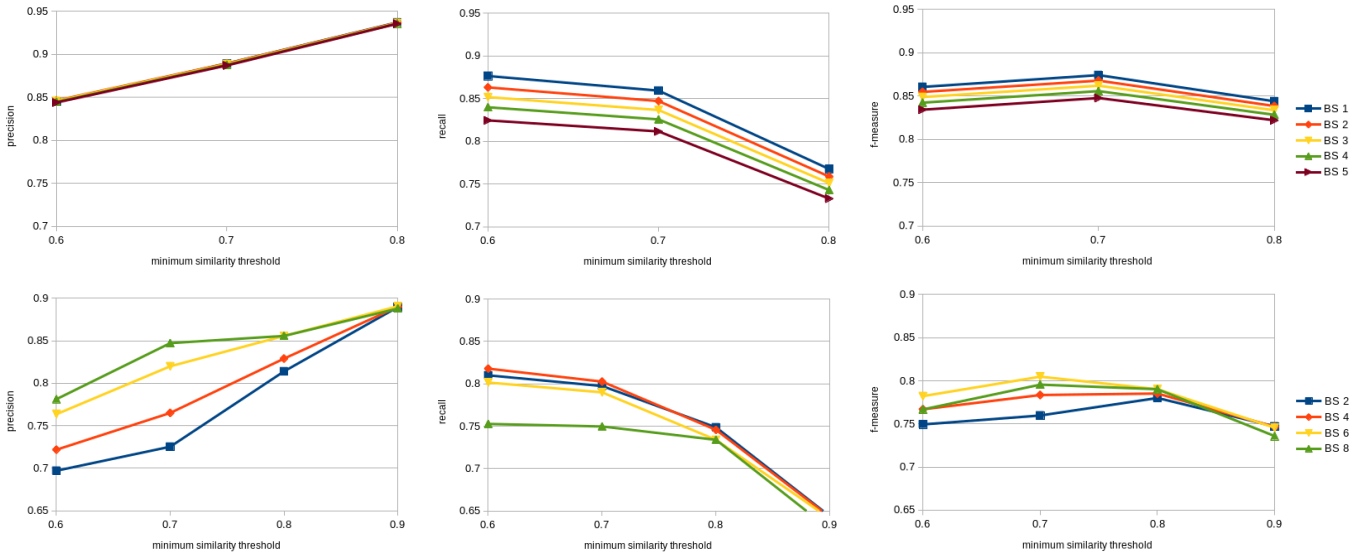



Figure 3: Precision, recall and F-measure for music dataset DS-M2 (top) and dataset DS-P2 for domain person (bottom).

| | DS-M2 | | | | DS-P2 | | | |
|---|---|---|---|---|---|---|---|---|
| bk length | 4 | 3 | 2 | 1 | 8 | 6 | 4 | 2 |
| $t_{\text{min}}$ 0.6 | 1071 | 2044 | 8292 | 46346 | 447 | 690 | 3621 | 67681 |
| $t_{\text{min}}$ 0.7 | 1093 | 2077 | 8062 | 46678 | 431 | 706 | 3338 | 72301 |
| $t_{\text{min}}$ 0.8 | 1139 | 2210 | 8875 | 49154 | 442 | 754 | 3203 | 96855 |

Table IV: Runtime (s) for different blocking key (bk) lengths and $t_{\text{min}}$ values (16 workers).

| | runtime (s) | | | | |
|---|---|---|---|---|---|
| bk length | 5 | 4 | 3 | 2 | 1 |
| source 1 + source 2 | 129 | 170 | 315 | 1183 | 6456 |
| add source 3 | 173 | 235 | 440 | 1692 | 9862 |
| add source 4 | 218 | 311 | 600 | 2319 | 13481 |
| add source 5 | 272 | 377 | 722 | 2868 | 16879 |
| total | 792 | 1093 | 2077 | 8062 | 46678 |

Table V: Runtimes (s) for single steps with source-specific clustering for DS-M2 with different blocking key (bk) lengths and $t_{\text{min}} = 0.7$.

contain $> 1000$ entities (biggest block contains 6617 entities) while these blocks contain $4.6\,\%$ of all entities - these blocks need most of the time for the actual comparison. $1.7\,\%$ of all blocks are within the range 101-1000, but contain $35.3\,\%$ of the entities. Finally, $98.3\,\%$ of the blocks have 1-100 entities ($60.2\,\%$ of all entities belong to these blocks). For bigger prefix length, the distribution is likely to have more and smaller blocks reducing computational effort but also reducing the possibility to compare potentially similar entities with each other. The runtime differences (Table IV) are again substantial where blocking with prefix length 6 achieves a fast clustering of only $706\,\text{s}$ ($12\,\text{min}$) for $t_{\text{min}} = 0.7$ which is two orders of magnitude faster than with prefix length 2.

This is especially remarkable since the runtime typically increases strongly with the number of sources; each new source increases the number of clusters and thus the match overhead heavily. We illustrate this in Table V for dataset DS-M2 showing how the total runtime is distributed over the additions of the different sources. We observe that the time to add a source increases continuously so that adding the 5th source is more than twice as slow than adding the second source. This is because $25\%$ of the entities in each source have no duplicates for DS-M2[4] thereby resulting in the creation of additional clusters that have to be considered in the subsequent cluster decisions.

*3) Comparison of incremental and static clustering:* We now compare the runtime and cluster quality for the two incremental approaches with static entity clustering for datasets DS-M2 and DS-P1. We focus the analysis on a specific blocking key (length 4 for DS-M2, length 6 for DS-P1), similarity threshold ($t_{\text{min}}$ 0.7) and use the same blocking and similarity functions to determine the similarity graphs for static entity clustering. We consider two state-of-the art approaches for multi-source entity clustering on the similarity graph, namely

[4] $50\%$ of the clusters in DS-M2 (500K) are singletons while the other half of the clusters has between 2 and 5 duplicate entities.

| | DS-M2 | | | | DS-P1 | | | |
|---|---|---|---|---|---|---|---|---|
| | Incremental | | Static | | Incremental | | Static | |
| | Base | Source | CLIP | SplitMerge | Base | Source | CLIP | SplitMerge |
| runtime (s) | 4148 | 1093 | 1748 + 68 | 1748 + 659 | 614 | 217 | 107 + 101 | 107 + 752 |
| precision | 0.744 | 0.888 | 0.848 | 0.838 | 0.553 | 0.811 | 0.850 | 0.798 |
| recall | 0.844 | 0.826 | 0.815 | 0.822 | 0.836 | 0.817 | 0.821 | 0.851 |
| F-measure | 0.791 | 0.856 | 0.831 | 0.830 | 0.665 | 0.814 | 0.835 | 0.824 |

Table VI: Incremental vs. static clustering for DS-M2 and DS-P1 with blocking key length 4 and 6 resp. with $t_{\text{min}} = 0.7$.

the CLIP approach of [6] and the SplitMerge approach of [1], [5].

Table VI shows the obtained runtime and F-measure results where the runtime of static entity clustering consists of the sum of the time to create the similarity graph and the time for clustering itself. We observe that source-specific incremental clustering reaches not only substantially better F-measure than the base incremental approach but also much better runtime due to the fast Max-Both approach to determine 1:1 assignments. The 1:1 assignments of source-specific incremental matching lead to dramatically improved precision over the base approach, especially for DS-P1 the precision increases from $55.3\%$ to $81.1\%$; that more than outweighs the somewhat reduced recall thereby achieving a substantial improvement in F-Measure ($85.6\%$ vs. $79.1\%$ for DS-M2 and $81.4\%$ vs. $66.5\%$ for DS-P1). At the same time this better quality is achieved by a much lower runtime with an improvement by factor $3.8$ for DS-M2 and factor $2.8$ for DS-P1.

In comparison to static clustering, we expect better runtime and a reduced match quality for incremental clustering since we only optimize cluster assignment for a subset of entities at a time. For DS-M2, the time to generate the similarity graph for static entity clustering ($1748\,\text{s}$) is already higher than for the entire source-specific incremental clustering despite the use of blocking key length 4 and therefore small blocks. For DS-P1 we used blocking keys of length 6 which reduces the overhead for the graph generation for the static approach to only $107\,\text{s}$. Therefore, overall runtime for graph generation and CLIP ($107\,\text{s}$ + $101\,\text{s}$) is about the same than for the source-based incremental approach ($217\,\text{s}$) while the static approach SplitMerge is slower by a factor of $4$. The base incremental approach is mostly slower than the static approaches. However, the runtime for the incremental approaches is the sum for all source additions while each incremental addition of entity sets and sources is typically faster than a complete static (re-) computation of clusters.

In terms of match quality we observe that the base incremental approach is always inferior to the static approaches but that the source-specific incremental approach is actually better for DS-M2 ($2.5\%$ higher F-measure) and only slightly worse for DS-P1 ($1.0$-$2.1\%$ lower F-measure than SplitMerge and CLIP). We can thus conclude that the source-specific incremental clustering reaches comparable cluster quality than the static approaches with typically faster runtime and support for dynamic addition of new entities and sources.

| | runtime (s) | | | |
|---|---|---|---|---|
| cluster size | P1 b4 | P1 b6 | P2 b4 | P2 b6 |
| 1 | 9751 | 860 | 32093 | 3648 |
| 2 | 5468 | 732 | 17583 | 2275 |
| 4 | 2997 | 430 | 9346 | 1494 |
| 8 | 1785 | 320 | 5677 | 1023 |
| 16 | 1100 | 217 | 3338 | 706 |

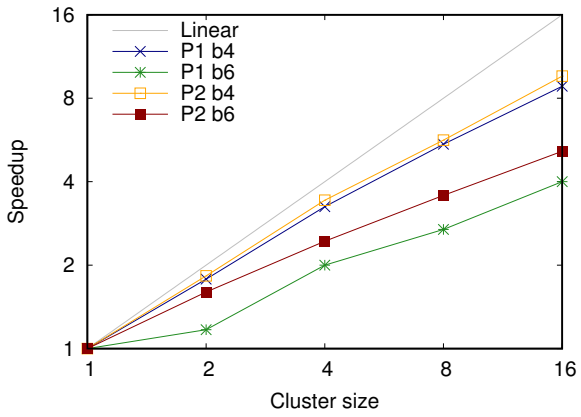



Figure 4: Execution times (top) in seconds and speedup (bottom) for datasets DS-P1 (P1) and DS-P2 (P2) for blocking key lengths 4 and 6.

*4) Speedup behavior:* We finally analyze the speedup behavior of (source-specific) incremental clustering w.r.t. clusters of different sizes. For this experiment, we focus on the two largest datasets DS-P1 (5 sources) and DS-P2 (10 sources). In both cases we consider two blocking configurations with prefix length 4 and 6. The resulting runtime and speedup values for using 1 to 16 workers are shown in Figure 4. The results show that we can always improve runtime by using more workers where the speedup is best for the most expensive configurations with bigger blocks (prefix length 4). For DS-P2 and we achieve near-linear speedups of $5.7$ for 8 workers and $9.6$ for 16 workers in this case. For prefix length 6, the runtime to incrementally cluster 5 and 10 million entities is less than 4 and 12 minutes respectively for 16 workers and thus remarkably fast.

## V. Related Work

Linking entities is widely investigated and approaches and prototypes are surveyed for entity resolution as well as link discovery [1], [7], [16]. Previous approaches to cluster entities (including connected components and correlation clustering) were mostly static and often considered only entities from a single source [3]. Static clustering of entities from multiple sources has been studied in [4]–[6], [17]. An overview of previous research for matching both schemas (ontologies) and entities from multiple sources for holistic data integration is provided in [2].

Relatively little work has been published on incremental entity resolution. Most previous approaches including [18], [19] follow the straight-forward approach to consider new entities in isolation and add it to the cluster with the highest similarity above a threshold.

The incremental clustering approach of Gruenheid et al. [20] is more sophisticated. They determine and incrementally update a similarity graph to determine the best cluster assignments. Their approaches are also able to repair previous cluster decisions. The evaluation focuses on changes for two single datasets of small size thus leaving open the scalability to large datasets and several data sources which is the focus of our work.

## VI. Conclusions

We proposed and evaluated scalable and incremental approaches to cluster matching entities from multiple sources. In contrast to static entity clustering, the new approaches can continuously integrate additional sources and entities and avoid the expensive computation of similarity graphs for entities from many sources. To optimize the cluster assignments we consider sets of new entities together. Especially promising is the source-specific incremental clustering determining a 1:1 assignment between entities and existing clusters using a Max-Both strategy. The evaluation for datasets from three domains showed that the proposed approaches are highly effective and efficient and often faster than with static entity clustering. The source-specific approach outperforms the base approach in both cluster quality and runtime and should thus be applied whenever possible. The use of blocking and parallel processing adds to the efficiency and scalability of the incremental approaches so that millions of entities from up to ten sources could be clustered within a few minutes.

In future work we plan to investigate the use of repair strategies during incremental clustering, e. g., to correct earlier errors by reassigning some cluster members.

## Acknowledgments

This research is supported by the Deutsche Forschungsgemeinschaft (DFG) grant RA 497/19-2.

## References

[1] M. Nentwig, M. Hartung, A. N. Ngomo, and E. Rahm, "A survey of current Link Discovery frameworks," *Semantic Web*, vol. 8, no. 3, pp. 419–436, 2017.

[2] E. Rahm, "The Case for Holistic Data Integration," in *Proc. ADBIS*, 2016, pp. 11–27.

[3] O. Hassanzadeh, F. Chiang, R. J. Miller, and H. C. Lee, "Framework for Evaluating Clustering Algorithms in Duplicate Detection," *PVLDB*, vol. 2, no. 1, pp. 1282–1293, 2009.

[4] A. Saeedi, E. Peukert, and E. Rahm, "Comparative Evaluation of Distributed Clustering Schemes for Multi-source Entity Resolution," in *Proc. ADBIS*, 2017.

[5] M. Nentwig, A. Groß, and E. Rahm, "Holistic Entity Clustering for Linked Data," in *Proc. ICDM Workshops*, 2016, pp. 194–201.

[6] A. Saeedi, E. Peukert, and E. Rahm, "Using Link Features for Entity Clustering in Knowledge Graphs," in *Proc. ESWC*. Springer, 2018, pp. 576–592.

[7] P. Christen, *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*, ser. Data-Centric Systems and Applications. Springer, 2012.

[8] D. Gale and L. S. Shapley, "College Admissions and the Stability of Marriage," *The American Mathematical Monthly*, vol. 120, no. 5, pp. 386–391, 2013.

[9] J. Munkres, "Algorithms for the assignment and transportation problems," *Journal of the society for industrial and applied mathematics*, vol. 5, no. 1, pp. 32–38, 1957.

[10] H. H. Do and E. Rahm, "COMA - A System for Flexible Combination of Schema Matching Approaches," in *Proc. VLDB*, 2002, pp. 610–621.

[11] M. Franke, Z. Sehili, M. Gladbach, and E. Rahm, "Post-processing Methods for High Quality Privacy-preserving Record Linkage," in *Proc. 13th Int. Workshop on Data Privacy Management (DPM)*, ser. LNCS. Springer, 2018.

[12] P. Carbone, A. Katsifodimos, S. Ewen, V. Markl, S. Haridi, and K. Tzoumas, "Apache Flink™: Stream and Batch Processing in a Single Engine," *IEEE Data Eng. Bull.*, vol. 38, no. 4, 2015.

[13] L. Kolb, A. Thor, and E. Rahm, "Load Balancing for MapReduce-based Entity Resolution," in *Proc. ICDE*, 2012, pp. 618–629.

[14] M. Nentwig, A. Groß, M. Möller, and E. Rahm, "Distributed Holistic Clustering on Linked Data," in *Proc. OTM*, 2017, pp. 371–382.

[15] K. Hildebrandt, F. Panse, N. Wilcke, and N. Ritter, "Large-Scale Data Pollution with Apache Spark," *IEEE Transactions on Big Data*, 2017.

[16] H. Köpcke and E. Rahm, "Frameworks for entity matching: A comparison," *Data & Knowledge Engineering*, vol. 69, no. 2, pp. 197 – 210, 2010.

[17] K. Bellare, C. Curino, A. Machanavajihala, P. Mika, M. Rahurkar, and A. Sane, "Woo: A scalable and multi-tenant platform for continuous knowledge base synthesis," *PVLDB*, vol. 6, no. 11, pp. 1114–1125, 2013.

[18] M. J. Welch, A. Sane, and C. Drome, "Fast and Accurate Incremental Entity Resolution Relative to an Entity Knowledge Base," in *Proc. CIKM*, 2012, pp. 2667–2670.

[19] G. Costa, G. Manco, and R. Ortale, "An incremental clustering scheme for data de-duplication," *Data Min. Knowl. Discov.*, vol. 20, no. 1, pp. 152–187, 2010.

[20] A. Gruenheid, X. L. Dong, and D. Srivastava, "Incremental Record Linkage," *PVLDB*, vol. 7, no. 9, pp. 697–708, 2014.