Bipartite Graph Matching Algorithms for Entity Resolution: An Empirical Evaluation [Experiment, Analysis & Benchmark Papers]

George Papadakis[1], Vasilis Efthymiou[2], Emanouil Thanos[3], Oktie Hassanzadeh[4]
[1]National and Kapodistrian University of Athens, Greece gpapadis@di.uoa.gr
[2]Foundation for Research and Technology - Hellas, Greece vefthym@ics.forth.gr
[3]Katholieke Universiteit Leuven, Belgium emmanouil.thanos@kuleuven.be
[4]IBM Research AI, USA hassanzadeh@us.ibm.com

ABSTRACT

Entity Resolution (ER) is the task of finding records that refer to the same real-world entities. A common scenario is when entities across two clean sources need to be resolved, which we refer to as Clean-Clean ER. In this paper, we perform an extensive empirical evaluation of 8 bipartite graph matching algorithms that take in as input a bipartite similarity graph and provide as output a set of matched entities. We consider a wide range of matching algorithms, including algorithms that have not previously been applied to ER, or have been evaluated only in other ER settings. We assess the relative performance of the algorithms with respect to accuracy and time efficiency over 10 established, real datasets, from which we extract >700 different similarity graphs. Our results provide insights into the relative performance of these algorithm as well as guidelines for choosing the best one, depending on the ER application at hand.

1 INTRODUCTION

Entity Resolution is a challenging, yet well-studied problem in data integration [5, 6, 21]. A common scenario is Clean-Clean ER (CCER) [37], where the two data sources to be integrated are both clean (i.e., duplicate-free), or are cleaned using single-source entity resolution frameworks. Example applications include Master Data Management [34], where a new clean source needs to be integrated into the clean reference data, and Knowledge Graph matching and completion [17, 43], where an existing clean knowledge base needs to be augmented with an external source.

We focus on methods that take advantage of a large body of work on blocking and matching algorithms, which efficiently compare the entities across two sources and provide as output pairs of entities along with a confidence or similarity score [7, 10, 37]. The output can then be used to decide which pairs should be matched. The simplest approach is specifying as duplicates all the pairs with a score higher than a given threshold. Choosing a single threshold fails to address the issue that in most cases the similarity scores vary significantly depending on the characteristics of the entities. More importantly, for CCER, this approach does not guarantee that each source entity can be matched with at most one other entity. If we view the output as a bipartite *similarity graph*, where the nodes are entity profiles and the edge weights are the matching scores between the candidate duplicates, what we need is finding a *matching* (or independent edge set [28]) so that each entity from one source is matched to at most one entity in the other source.

In this paper, we present the results of our thorough evaluation of efficient bipartite graph matching algorithms for CCER. To the best of our knowledge, our study is the first to primarily focus on bipartite graph matching algorithms, examining the relative performance of the algorithms in a variety of data sets and methods of creating the input similarity graph. Our goal is to answer the following questions:

- *Which bipartite graph matching algorithm is the most accurate one, which is the most robust one, and which offers the best balance between effectiveness and time efficiency?*
- *How well do the main algorithms scale?*
- *Which characteristics of the inputs graphs determine the absolute and the relative performance of the algorithms?*

By answering these questions we intend to facilitate the selection of the best algorithm for the data at hand.

In summary, we make the following contributions:

- We present an overview of eight efficient bipartite graph matching algorithms along with an analysis of their behavior and complexity. Some of the algorithms are adaptations of efficient graph clustering algorithms that have not been applied to CCER before.
- We organize the input of bipartite graph partitioning algorithms into a taxonomy that is based on the learning-free source of similarity scores/edge weights.
- We assess the relative performance of the matching algorithms with respect to effectiveness and time efficiency.
- We perform an extensive experimental analysis that involves 739 different similarity graphs from 10 established real-world CCER datasets, whose sizes range from several thousands to hundreds of million edges.
- We have publicly released the implementation of all algorithms as well as our experimental results.[1]

In what follows, we first discuss the preliminaries and a brief overview of related work in Section 2. We then present the matching algorithms we study in this paper in Section 3. Section 4 describes the generation of similarity graphs. Section 5 presents our experimental evaluation setup and Section 6 presents the results along with a detailed analysis. We conclude the paper with a summary of the key takeaways from our results.

2 PRELIMINARIES

We assume that an *entity profile* or simply *entity* is the description of a real-world object, provided as a set of attribute-value pairs in some *entity collection* V. The problem of Entity Resolution (ER) is to identify such entity profiles (called *matches* or *duplicates*) that correspond to the same real-world object, and place them in a common *cluster* c. In other words, the output of ER, ideally, is a

[1] See https://github.com/scify/JedAIToolkit for more details.

set of clusters C, each containing all the matching profiles that correspond to a single real-world entity.

In this paper, we focus on the case of *Clean-Clean ER* (CCER), in which we want to match profiles coming from two clean (i.e., duplicate-free) entity collections V_1 and V_2. This means that the resulting clusters should contain at most two profiles, one from each collection. *Singular clusters*, corresponding to profiles for which no match has been found, are also acceptable.

To generate this clustering, a typical ER pipeline [6] involves the steps of *(i) (meta-)blocking*, i.e., indexing steps that generate *candidate matching pairs*, this way reducing the otherwise quadratic search space of matches, *(ii) matching*, assigning a similarity score to each candidate pair, and *(iii) clustering*, which receives the scored candidate pairs and decides which pairs will be placed together in a cluster. In this work, we evaluate how different clustering methods for CCER perform, when the previous ER steps are fixed.

Problem Definition. The task of **Bipartite Graph Matching** receives as input a bipartite *similarity graph* $SE = (V_1, V_2, E)$, where V_1 and V_2 are two clean entity collections, and $E \subseteq V_1 \times V_2$ is the set of edges with weights in [0,1], corresponding to the similarity scores between entity profiles of the two collections. Its output comprises a set of equivalence clusters C, with each one containing one node $v_i \in V_1 \cup V_2$ or two nodes $v_i \in V_1$ and $v_j \in V_2$ that represent the same real-world object.

Figure 1 (a) shows an example of a bipartite similarity graph, in which node partitions (entity collections) are labeled as A (in orange) and B (in blue). The edges connect only A nodes to B nodes and are associated with a weight that reflects the similarity (matching likelihood) of the connected nodes. Figures 1(b) - (d) show three different outputs of CCER, in which nodes within the same oval (cluster) correspond to matches, i.e., descriptions of the same real-world entity.

2.1 Related work

There is a rich body of literature on various forms of ER [6, 37]. Following the seminal Fellegi-Sunter model for record linkage [12], a major focus of prior work has been on classifying pairs of input records as *match*, *non-match*, or *potential match*. While even some of the early work on record linkage incorporated a 1-1 matching constraint [50], the primary focus of prior work, especially the most recent one, has been on the effectiveness of the classification task, mainly by leveraging Machine [20] and Deep Learning [4, 27, 32].

Inspired by the recent progress and success of prior work on improving the efficiency of ER with blocking and filtering [40], we target ER frameworks where the output of the matching is used to construct a similarity graph that needs to be partitioned for the final step of entity resolution. Hassanzadeh et al. [18] also target such a framework, and perform an evaluation of various graph clustering algorithms for entity resolution. However, they target a scenario where input data sets are not clean or more than two clean sources are merged into a dirty source that contains duplicates in itself, and as a result each cluster could contain more than two records. We refer to this variation of ER as *Dirty ER* [6]. Some of the bipartite matching algorithms we use in this paper are adaptations of the graph clustering algorithms used in [18] for Dirty ER.

Gemmel et al. [15] present two algorithms for CCER as well as more algorithms for different ER settings (e.g., one-to-many and many-to-many). Both algorithms are covered by the clustering algorithms that are included in our study: the MutualFirstChoice is equivalent to our Exact clustering, while the Greedy algorithm is equivalent to UniqueMappingClustering. Finally, the MaxWeight method [15] utilises the exact solution of the maximum weight bipartite matching, for which an efficient heuristic approach is considered in our Best Assignment Heuristic Clustering.

FAMER [43] is a framework that supports multiple matching and clustering algorithms for multi-source ER. Although it studies some common clustering algorithms with those explored in this paper (e.g., Connected Components), the focus on multi-source ER, as opposed to our two-source ER (CCER), makes the direct comparison inapplicable. Additionally, the adaptation of FAMER's CLIP clustering to work in a CCER setting yields an algorithm equivalent to Unique Mapping Clustering.

Several algorithms have been proposed for weighted bipartite matching in the literature [23, 45, 46]. Our focus in this paper is on efficient algorithms that are significantly faster than the classic Hungarian matching algorithm (also known as the Kuhn-Munkres algorithm[24]), which does not scale to medium-sized datasets of ER, as its time complexity is cubic $O(n^3)$, where n is the total number of nodes in the input graph. An example of more recent matching algorithms that we do not include in this paper due to their time complexity is the work of 1-1 bipartite graph matching with minimum cumulative weights. In other words, they return the edges (matches) with the smallest sum of weights. The authors reduce the problem to a minimum cost flow problem, use the matching algorithm of [13] and provide an approximate solution in O($n^2 logn$).

Wang et al. [46] follow a reinforcement learning approach, based on a Q-learning [48] algorithm, for which a state is represented by the pair $(|L|, |R|)$, where $L \subseteq V_1, R \subseteq V_2$ are the nodes matched from the two partitions, and the reward is computed as the sum of the weights of the selected matches. We leave this algorithm outside the scope of this study, as we consider only learning-free methods, but we plan to further explore it in our future works.

Kriege et al. [22] present a linear approximation to the weighted graph matching problem, but for that, they require that the edge weights are assigned by a tree metric, i.e. a similarity measure that satisfies a looser version of the triangle inequality. In this work, we investigate algorithms that are agnostic to such similarity measure properties, assuming only that the weights are in [0,1], as is the case of most existing algorithms.

3 ALGORITHMS

We now describe the main graph matching algorithms for CCER. All of them are configured by a similarity threshold t, which prunes all edges with a lower weight. To describe their time complexity, we denote by $n = |V_1 \cup V_2|$ and $m = |E|$ the number of nodes and edges, respectively, in the bipartite similarity graph $G = (V_1, V_2, E)$ that is given as input. The implementation of all algorithms (in Java) is publicly available through the JedAI toolkit[1] [38].

Connected Components (CNC). This is the simplest algorithm: it discards all edges with a weight lower than the similarity threshold and then computes the transitive closure of the pruned similarity

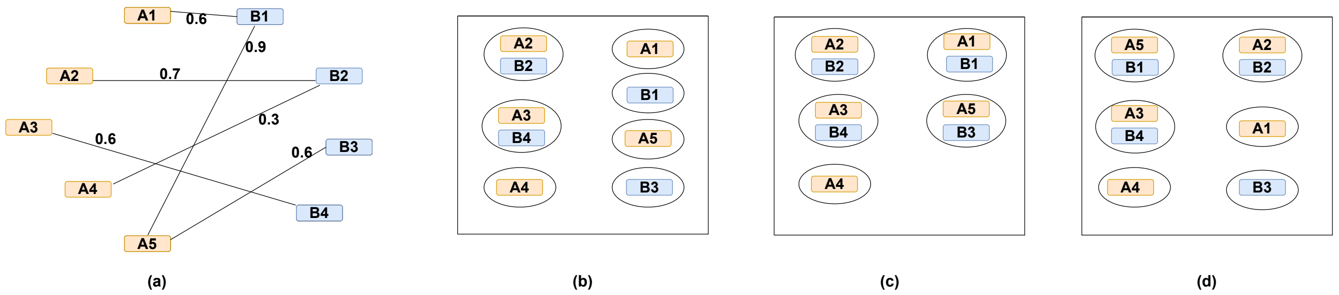


Figure 1: Example of processing a similarity graph: (a) the similarity graph constructed for a pair of clean entity collections (V_1 in orange and V_2 in blue), (b) the resulting clusters after applying CNC, (c) the resulting partitions/clusters assuming that the approximation algorithms RCA or BAH retrieved the optimal solution for the maximum weight bipartite matching or the assignment problem, and (d) the resulting clusters after applying UMC, BMC or EXC.

graph. In the output, it solely retains the partitions/clusters that contain two entities – one from each entity collection. Using a simple depth-first approach, its time complexity is $O(m)$ [8].

Ricochet Sequential Rippling Clustering (RSR). This algorithm is an adaptation of the homonymous method for Dirty ER in [18] such that it exclusively considers clusters with just one entity from each enitty collection. After pruning the edges weighted lower than t, RSR sorts all nodes from both V_1 and V_2 in descending order of the average weight of their adjacent edges. Whenever a new seed is chosen from the sorted list, the first adjacent vertex that is currently unassigned or is closer to the new seed than it is to the seed of its current partition is re-assigned to the new cluster. If a partition is reduced to a singleton after a re-assignment, it is placed in its nearest single-node cluster. The algorithm stops when all nodes have been considered. Its time complexity is $O(n~m)$ [49].

Row Column Assignment Clustering (RCA). This approach is based on the Row-Column Scan approximation method in [25] that solves the assignment problem. It requires two passes of the similarity graph, with each pass generating a candidate solution. In the first pass, each entity from V_1 creates a new partition, to which the most similar, currently unassigned entity from V_2 is assigned. Note that, in principle, any pair of entities can be assigned to the same partition at this step even if their similarity is lower than t, since the assignment problem assumes that each vertex from V_1 is connected to all vertices from V_2 (any "job" can be performed by all "men"). The clusters of pairs with similarity less than t are then discarded. In the second pass, the same procedure is applied to the entities/nodes of V_2. The value of each solution is the sum of the edge weights between the nodes assigned to the same (2-node) partition. The solution with the highest value is returned as output. Its time complexity is $O(|V_1|~|V_2|)$ [].

Best Assignment Heuristic (BAH). This algorithm applies a simple swap-based random-search algorithm to heuristically solve the maximum weight bipartite matching problem and uses the resulting solution to create the output partitions. Initially, each entity from the smaller entity collection is randomly connected to an entity from the larger one. In each iteration of the search process, two entities from the larger entity collection are randomly selected in order to swap their current connections. If the sum of the edge weights of the new pairs is higher than the previous pairs, the swap is accepted. The algorithm stops when a maximum number of search steps is reached or when a maximum run-time has been exceeded. In our case, the run-time limit has been set to 2 minutes.

Best Match Clustering (BMC). This algorithm is inspired from the Best Match strategy of [30], which solves the stable marriage problem [14], as simplified in BigMat [2]. For each entity of the one entity collection, this algorithm creates a new partition, in which the most similar, not-yet-clustered entity from the other entity collection is also placed – provided that the corresponding edge weight is higher than t. Note that the greedy heuristic for BMC introduced in [30] is the same, in principle, to Unique Mapping Clustering (see below). Note also that BMC is the only algorithm with an additional configuration parameter, apart from the similarity threshold: the entity collection that is used as the basis for creating partitions can be set to V_1 or V_2. In our experiments, we examine both options and retain the best one. Its time complexity is $O(m)$ [].

Exact Clustering (EXC). This algorithm is inspired from the Exact strategy of [30]. EXC places two entities in the same partition only if they are mutually the best matches, i.e., the most similar candidates of each other, and their edge weight exceeds t. This approach is basically a stricter, symmetric version of BMC and could also be conceived as a strict version of the reciprocity filter that was employed in [11, 39]. Its time complexity is $O(n~m)$.

Király's Clustering (KRC). This algorithm is an adaptation of the linear time 3/2 approximation to the maximum stable marriage problem, called "New Algorithm" in [19]. Intuitively, the entities of V_1 ("men" [19]) propose to the entities from V_2 with an edge weight higher than t ("women" [19]) to form a partition ("get engaged" [19]). The entities of V_2 accept a proposal under certain conditions (e.g., if it's the first proposal they receive), and the partitions and preferences are updated accordingly. Entities from V_1 get a second chance to make proposals and the algorithm terminates when all entities of V_1 are in a partition, or no more proposal chances are left. We omit some of the details (e.g., the rare case of "uncertain man"), due to space restrictions, and refer the reader to [19] for more information (e.g., the acceptance criteria for proposals). Its time complexity is $O(n + m~logm)$ [19].

Unique Mapping Clustering (UMC). This algorithm prunes all edges with a weight lower than t, sorts the remaining ones in decreasing weight/similarity and iteratively forms a partition for the top-weighted pair as long as none of its entities has already been matched to some other. This comes from the *unique mapping constraint* of CCER, i.e., the restriction that each entity from the one entity collection matches at most one entity from the other. Note that the *CLIP Clustering algorithm*, introduced for the multi-source ER problem in [44], is equivalent to UMC in the CCER case that we study. Its time complexity is $O(m\ logm)$ [].

Example. Figure 1 demonstrates an example of applying the above algorithms to the similarity graph in Figure 1(a). For all algorithms, we assume a weight threshold of 0.5.

CNC completely discards the 4-node connected component ($A1$, $B1, A5, B3$) and considers exclusively the valid partitions ($A2, B2$) and ($A3, B4$), as demonstrated in Figure 1(b).

Algorithms that aim to maximize the total sum of edge weights between the matched entities, such as RCA and BAH, will cluster $A1$ with $B1$ and $A5$ with $B3$, as shown in Figure 1(c), if they manage to find the optimal solution for the given graph. The reason is that this combination of edge weights yields a sum of $0.6 + 0.6 = 1.2$, which is higher than 0.9, i.e., the sum resulting from clustering $A5$ with $B1$ and leaving $A1$ and $B3$ as singletons.

UMC starts from the top-weighted edges, matching $A5$ with $B1$, $A2$ with $B2$ and $A3$ with $B4$; $A1$ and $B3$ are left as singletons, as shown in Figure 1(d), because their candidates have already been matched to other entities. The same output is produced by EXC, as the entities in each partition consider each other as their most similar candidate. For this reason, BMC also yields the same results assuming that V_2 (blue) is used as the basis entity collection.

The partitions generated by RSR and KRL depend on the sequence of adjacent vertices and proposals, respectively. Given, though, that higher similarities are generally more preferred than increasing total sum by both of these algorithms, the outcome in Figure 1 (d) is the most possible one for these algorithms, too.

4 SIMILARITY GRAPHS

Two types of methods can be used for the generation of the similarity graphs that constitute the input to the above algorithms:

(i) *learning-free* methods, which produce similarity scores in an unsupervised manner based on the content of the input entities, and

(ii) *learning-based* methods, which produce probabilistic similarities with the help of a training set [37].

In this work, we exclude the latter, focusing exclusively on learning-free methods. Thus, we make the most of the selected datasets, without sacrificing valuable parts for the construction of the training (and perhaps the validation) set. We also avoid depending on the fine-tuning of numerous configuration parameters, especially in the case of Deep Learning-based methods [47]. Besides, our goal is not to optimize the performance of the CCER process, but to investigate how the main graph matching algorithms perform under a large variety of real settings. For this reason, we produce a large number of similarity graphs per dataset, rather than generating synthetic data.

In this context, we do not apply any blocking method when producing these inputs. Instead, we consider all pairs of entities from different datasets with a similarity higher than 0. This allows for experimenting with a large variety of similarity graph sizes, which range from several thousand to hundreds of million edges. Besides, the role of blocking, i.e., the pruning of the entity pairs with very low similarity scores, is performed by the similarity threshold t that is employed by all algorithms.

The resulting similarity graphs differ in the number of edges and the corresponding weights, which were produced using different *similarity functions*. Each similarity function consists of two parts: (i) the *representation model*, and (ii) the *similarity measure*.

The **representation model** transforms a textual value into a model that is suitable for applying the selected similarity measure. Depending on the scope of these representations, we distinguish them into (i) *schema-agnostic* and (ii) *schema-based*. The former consider all attribute values in an entity description, while the latter consider only the value of a specific attribute. Depending on their form, we also distinguish the representations into (i) *syntactic* and (ii) *semantic*. The former operate on the original text of the entities, while the latter operate on vector transformations (embeddings) of the original text that aim to capture the actual connotation, leveraging external information that has been extracted from large and generic corpora through unsupervised learning.

The *schema-based syntactic representations* process each value as a sequence of characters or words and apply to mostly short textual values. For example, the attribute value "Joe Biden" can be represented as the set of tokens {'Joe', 'Biden'}, or the set of character 3-grams {'Joe', 'oe_', 'e_B', '_Bi', 'Bid', 'ide', 'den'}.

The *schema-agnostic syntactic representations* process the set of all individual attribute values. We use two types that have been widely applied to document classification tasks [1, 35]:

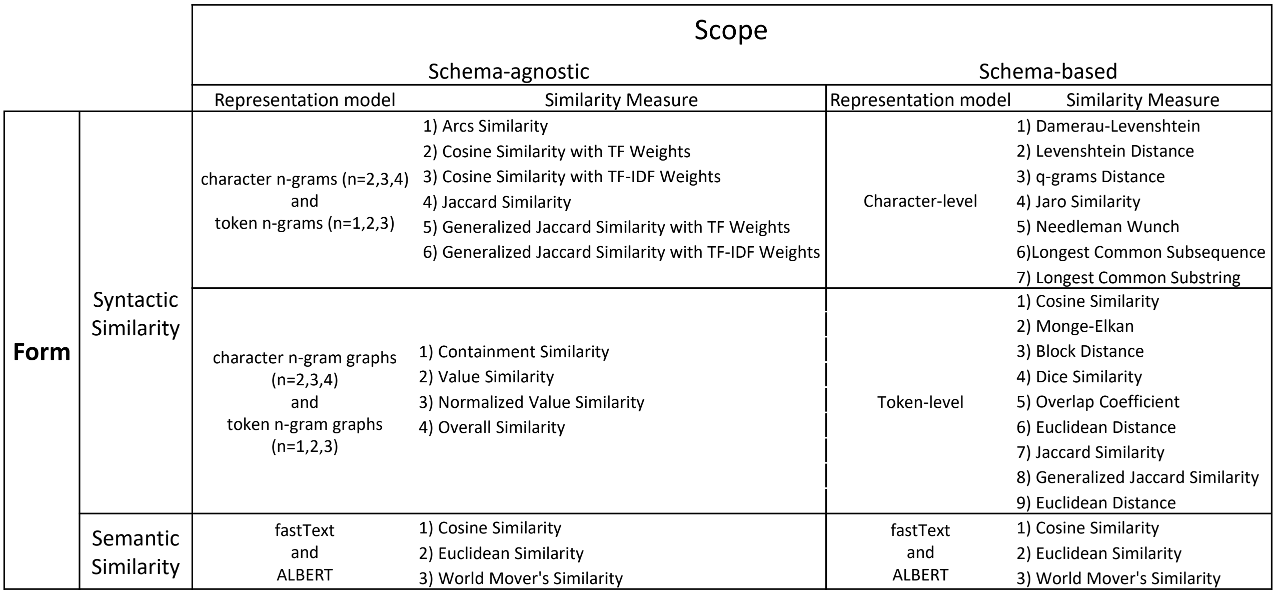
(i) an n-gram vector [29], whose dimensions correspond to character or token n-grams and are weighted according to their frequency (TF or TF-IDF score). This approach does not consider the order of n-gram appearances in each value.

(ii) an n-gram graph [16], which transforms each value into a graph, where the nodes correspond to character or token n-grams, the edges connect those co-occurring in a window of size n and the edge weights denote the n-gram's co-occurrence frequency. Thus, the order of n-grams in a value is preserved.

Following the previous example, the character 3-gram vector of "Joe Biden" would be a vector with as many dimensions as all the 3-grams appearing in the entity collection and fill it with zeros in all other places except the ones corresponding to the seven character 3-grams of "Joe Biden" listed above. For the places corresponding to those seven 3-grams, the value would be the TF or TF-IDF of each 3-gram. Similarly, a token 2-gram vector of "Joe Biden" would be all zeros, for each token 2-gram appearing in all the values, except for the place corresponding to the 2-gram 'Joe Biden', where its value would be 1. A character 3-gram graph would be a graph with seven nodes, one for each 3-gram listed above, connecting the node 'Joe' to the nodes 'oe_' and 'e_B', each with an edge of weight 1, the node 'oe_' to the nodes 'e_B' and '_Bi', etc.

Both approaches build an aggregate representation per entity: the n-gram vectors treat each entity as a "document" and adjust their weights accordingly, while the individual n-gram graphs of each value are merged into a larger "entity graph" through the update operator discussed in [16].

The *semantic representations* treat every text as a sequence of items (words or character n-grams) of arbitrary length and convert



		Scope			
		Schema-agnostic		Schema-based	
		Representation model	Similarity Measure	Representation model	Similarity Measure
Form	Syntactic Similarity	character n-grams (n=2,3,4) and token n-grams (n=1,2,3)	1) Arcs Similarity 2) Cosine Similarity with TF Weights 3) Cosine Similarity with TF-IDF Weights 4) Jaccard Similarity 5) Generalized Jaccard Similarity with TF Weights 6) Generalized Jaccard Similarity with TF-IDF Weights	Character-level	1) Damerau-Levenshtein 2) Levenshtein Distance 3) q-grams Distance 4) Jaro Similarity 5) Needleman Wunch 6)Longest Common Subsequence 7) Longest Common Substring
		character n-gram graphs (n=2,3,4) and token n-gram graphs (n=1,2,3)	1) Containment Similarity 2) Value Similarity 3) Normalized Value Similarity 4) Overall Similarity	Token-level	1) Cosine Similarity 2) Monge-Elkan 3) Block Distance 4) Dice Similarity 5) Overlap Coefficient 6) Euclidean Distance 7) Jaccard Similarity 8) Generalized Jaccard Similarity 9) Euclidean Distance
	Semantic Similarity	fastText and ALBERT	1) Cosine Similarity 2) Euclidean Similarity 3) World Mover's Similarity	fastText and ALBERT	1) Cosine Similarity 2) Euclidean Similarity 3) World Mover's Similarity

Figure 2: Taxonomy of the similarity functions we used to generate the similarity graphs. We use $n \in \{2, 3, 4\}$ for character and $n \in \{1, 2, 3\}$ for token n-grams for both vector and graph models, as in [1, 35]. The graph similarities are defined in [16].

it into a dense numeric vector based on learned external patterns. The closer the connotation of two texts is, the closer are their vectors. These representations come in two main forms, which apply uniformly to schema-agnostic and schema-based settings:

(i) The pre-trained embeddings of word- or character-level. Due to the highly specialized content of ER tasks (e.g., arbitrary alphanumerics in product names), the former, which include *word2vec* [31] and *GloVe* [41], suffer from a high portion of out-of-vocabulary tokens – these are words that cannot be transformed into a vector because they are not included in the training corpora [32]. This drawback is addressed by the character-level embeddings: *fastText* vectorizes a token by summing the embeddings of all its character n-grams [3]. For this reason, we exclusively consider the 300-dimensional fastText in the following.

(ii) Transformer-based language models [9] go beyond the shallow, context-agnostic pre-trained embeddings by vectorizing an item based on its context. In this way, they assign different vectors to homonyms, which share the same form, but different meaning (e.g., "bank" as a financial institution or as the border of a river). They also assign similar vectors to synonyms, which have different form, but almost the same meaning (e.g., "enormous" and "vast"). Several BERT-based language models have been applied to ER in [4, 27]. Among them, we exclusively consider the 768-dimensional ALBERT, due to its higher efficiency [26].

Every **similarity measure** receives as input two representation models and produces a score that is proportional to the likelihood that the respective entities correspond to the same real world object: the higher the score, the more similar are the input models and their textual values and the higher is the matching likelihood.

For each type of representation models, we considered a large variety of established similarity measures. We list them in Figure 2, and we formally define them in the Appendix [].

5 EXPERIMENTAL SETUP

All experiments were carried out on a server running Ubuntu 18.04.5 LTS with a 32-core Intel Xeon CPU E5-4603 v2 (2.20GHz), 128 GB of RAM and 1.7 TB HDD. All time experiments were executed on a single core. For the implementation of the schema-based syntactic similarity functions, we used the Simmetrics Java package[2]. For the schema-agnostic syntactic similarity functions, we used the implementation provided by the JedAI toolkit (the implementation of n-gram graphs and the corresponding graph similarities is based on the JInsect[3] Java toolkit). For the semantic representation models, we employed the Python sister package[4], which supports both fastText and ALBERT. For the computation of the semantic similarities, we used the Python scipy package.[5]

Datasets. In our experiments, we use 10 real-world, established datasets for ER, whose technical characteristics appear in Table 1. D_1, which was introduced in OAEI 2010[6], contains data about restaurants. D_2 matches products from the online retailers Abt and Buy [21]. D_3 interlinks products from Amazon and the Google Base data API (Google Pr.) [21]. D_4 contains data about publications from DBLP and ACM [21]. $D_5 - D_7$ contain data about television shows from TheTVDB.com (TVDB) and movies from IMDb and themoviedb.org (TMDb) [33]. D_8 contains data about products from Walmart and Amazon [32]. D_9 contains data about scientific publications from DBLP and Google Scholar (Scholar) [21]. D_{10} matches movies from IMDb and DBpedia [36]. All datasets are publicly available through the JedAI data repository.[7]

[2] https://github.com/Simmetrics/simmetrics
[3] https://github.com/ggianna/JInsect
[4] https://pypi.org/project/sister
[5] https://www.scipy.org
[6] http://oaei.ontologymatching.org/2010/im
[7] https://github.com/scify/JedAIToolkit/tree/master/data

Table 1: Technical characteristics of the real datasets for Clean-Clean ER in increasing number of computational cost. $|V_x|$ stands for the number of input entities, $|NVP_x|$ for the total number of name-value pairs, $|A_x|$ for the number of attributes and $|\bar{p}_x|$ for the average number of name-value pairs per entity profile in Dataset$_x$. $|D(V_1 \cap V_2)|$ denotes the number of duplicates in the ground-truth and $||V_1 \times V_2||$ the number of pairwise comparisons executed by the brute-force approach.

	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	D_{10}
Dataset$_1$	Rest.1	Abt	Amazon	DBLP	IMDb	IMDb	TMDb	Walmart	DBLP	IMDb
Dataset$_2$	Rest.2	Buy	Google Pr.	ACM	TMDb	TVDB	TVDB	Amazon	Scholar	DBpedia
$\lvert V_1\rvert$	339	1,076	1,354	2,616	5,118	5,118	6,056	2,554	2,516	27,615
$\lvert V_2\rvert$	2,256	1,076	3,039	2,294	6,056	7,810	7,810	22,074	61,353	23,182
NVP$_1$	1,130	2,568	5,302	10,464	21,294	21,294	23,761	14,143	10,064	$1.6 \cdot 10^5$
NVP$_2$	7,519	2,308	9,110	9,162	23,761	20,902	20,902	$1.14 \cdot 10^5$	$1.98 \cdot 10^5$	$8.2 \cdot 10^5$
$\lvert A_1\rvert$	7	3	4	4	13	13	30	6	4	4
$\lvert A_2\rvert$	7	3	4	4	30	9	9	6	4	7
$\lvert\bar{p}_1\rvert$	3.33	2.39	3.92	4.00	4.16	4.16	3.92	5.54	4.00	5.63
$\lvert\bar{p}_2\rvert$	3.33	2.14	3.00	3.99	3.92	2.68	2.68	5.18	3.24	35.20
$\lvert D(V_1 \cap V_2)\rvert$	89	1,076	1,104	2,224	1,968	1,072	1,095	853	2,308	22,863
$\lvert\lvert V_1 \times V_2\rvert\rvert$	$7.65 \cdot 10^5$	$1.16 \cdot 10^6$	$4.11 \cdot 10^6$	$6.00 \cdot 10^6$	$3.10 \cdot 10^7$	$4.00 \cdot 10^7$	$4.73 \cdot 10^7$	$5.64 \cdot 10^7$	$1.54 \cdot 10^8$	$6.40 \cdot 10^8$

Note that for the schema-based settings (both the syntactic and semantic ones), we used only the attributes that combine high coverage with high distinctiveness. That is, they appear in the majority of entities, while conveying a rich diversity of values, thus yielding high effectiveness. These attributes are for D_1: “name” and “phone”, for D_2: “name”, for D_3: “title”, for D_4: “title” and “authors”, for D_5: “modelno” and “title”, for D_6: “title” and “authors”, for D_7: “name” and “title”, for D_8: “title” and “name”, for D_9: “title” and “abstract”, and for D_{10}: “title”.

Evaluation Measures. In order to assess the relative performance of the above graph matching algorithms, we evaluate both their effectiveness and their time efficiency (and scalability). We measure their *effectiveness*, with respect to a ground truth of known matches, in terms of three measures:

- *Precision* denotes the portion of output partitions that involve two matching entities.
- *Recall* denotes that portion of partitions with two matching entities that are included in the output.
- *F-Measure* ($F1$) constitutes the harmonic mean of Precision and Recall: F1 = 2 · Precision · Recall / (Precision + Recall).

All measures are defined in $[0, 1]$, with higher values indicating higher effectiveness.

For *time efficiency*, we measure the average run-time of an algorithm for each setting, i.e., the time that intervenes between receiving the weighted similarity graph as input and returning the partitions as output, over 10 repeated executions.

Generation Process. To generate a large variety of input similarity graphs, we apply every similarity function described in Section 4 to all datasets in Table 1. We actually apply all combinations of representation models and similarity measures in Figure 2, thus yielding 60 schema-agnostic syntactic similarity graphs per dataset, 16 schema-based similarity graphs per attribute in each dataset, and 12 semantic similarity graphs per dataset. Note that we did not apply any fine-tuning to ALBERT, as our goal is not optimize ER performance, but rather to produce diverse inputs.

To estimate the algorithms' performance, we first apply min-max normalization to the edge weights of all similarity graphs, regardless of the similarity function that produced them, to ensure that they are restricted to $[0, 1]$. Next, we apply every algorithm to every input similarity graph by varying its similarity threshold from 0.05 to 1.0 with a step of 0.05. The largest threshold that achieves the highest F-Measure is selected as the optimal one, determining the performance of the algorithm for the particular input.

Table 2: The number of similarity graphs $|G|$ as well as their size, in terms of the average number of edges $|\bar{E}|$, per dataset.

	Syntactic Similarities				Semantic Similarities			
	Schema-based		Schema-ag.		Schema-based		Schema-ag.	
	$\lvert\mathrm{G}\rvert$	$\lvert\bar{\mathrm{E}}\rvert \cdot 10^6$	$\lvert\mathrm{G}\rvert$	$\lvert\bar{\mathrm{E}}\rvert \cdot 10^6$	$\lvert\mathrm{G}\rvert$	$\lvert\bar{\mathrm{E}}\rvert \cdot 10^6$	$\lvert\mathrm{G}\rvert$	$\lvert\bar{\mathrm{E}}\rvert \cdot 10^6$
D_1	20	0.16	46	0.72	8	0.26	2	0.76
D_2	12	1.05	47	0.64	2	1.16	2	1.16
D_3	14	2.89	53	2.65	2	4.11	2	4.11
D_4	27	4.49	24	3.84	12	5.99	4	6.00
D_5	24	5.81	48	11.92	12	8.22	2	30.64
D_6	25	8.39	45	10.99	12	12.31	2	39.81
D_7	26	2.80	42	12.21	12	36.44	5	46.99
D_8	26	28.10	47	37.31	2	37.70	-	-
D_9	20	119.18	46	77.56	6	154.26	2	154.36
D_{10}	13	250.73	43	317.17	2	378.51	-	-
Σ	207	-	441	-	70	-	21	-

Special care was taken to clean the experimental results from noise. We removed all similarity graphs where all matching entities had a zero edge weight. We also removed all noisy graphs, where all algorithms achieve an F-Measure lower than 0.25. Finally, we cleaned our data from duplicate inputs, i.e., similarity graphs that emanate from the same dataset but different similarity function and have the same number of edges, while at least two different algorithms achieve their best performance with the same similarity threshold, exhibiting almost identical effectiveness, i.e., the difference in F-Measure and precision or recall is less than 0.2%.

The characteristics of the retained similarity graphs appear in Table 2. Overall, there are 739 different similarity graphs, most of which rely on syntactic similarity functions and the schema-agnostic settings, in particular. The reason is that the semantic similarities assign relatively high similarity scores to most pairs of

Table 3: Average performance across all similarity graphs.

	Precision		Recall		F-Measure	
	μ	σ	μ	σ	μ	σ
CNC	**0.801**	0.185	0.403	0.257	0.490	0.237
RSR	0.615	0.228	0.455	0.239	0.499	0.216
RCA	0.590	0.224	0.502	0.238	0.518	0.211
BAH	0.548	0.236	0.383	0.282	0.408	0.246
BMC	0.631	0.212	0.582	0.221	0.586	0.196
EXC	0.735	0.197	0.544	0.242	0.591	0.199
KRC	0.696	0.200	0.597	0.223	**0.619**	0.187
UMC	0.645	0.212	**0.628**	0.212	0.618	0.193

entities, thus resulting in poor performance for all considered algorithms – especially in the schema-agnostic settings. Every dataset is represented by at least 58 similarity graphs, in total, while the average number of edges ranges from 160K to 379M. This large set of real-world similarity graphs allows for a rigorous testing of the graph matching algorithms under diverse conditions.

6 EXPERIMENTAL ANALYSIS

In this section, we analyze the experimental results, which we split into effectiveness results and time efficiency results. Our goal is not to declare a single winner, but rather to help identify the use cases for which one algorithm should be preferred over others.

6.1 Effectiveness Measures

The most important performance aspect of clustering algorithms is their ability to effectively distinguish the matching from the non-matching pairs. In this section, we examine this aspect, addressing the following questions:

QE(1): What is the trade-off between precision and recall that is achieved by each algorithm?
QE(2): Which algorithm is the most/least effective?
QE(3): How does the type of input affect the effectiveness of the evaluated algorithms?
QE(4): Which other factors affect their effectiveness?

To answer QE(1) and QE(2), we consider the average performance (μ) of all algorithms across all input similarity graphs, which is reported in Table 3. We observe that all algorithms emphasize on precision at the cost of lower recall. The most balanced algorithm is UMC, as it yields the smallest difference between the two measures (just 0.017). In contrast, CNC constitutes the most imbalanced algorithm, as its precision is almost double its recall. The former achieves the second best F-Measure, being very close to the top performer KRC, while the latter achieves the second worst F-Measure, surpassing only BAH. Note that BAH is the least robust with respect to all measures, as indicated by their standard deviation (σ), due to its stochastic functionality, while CNC, UMC and KRC are the most robust with respect to precision, recall and F-Measure, respectively. Among the other algorithms, EXC and BMC are closer to KRC and UMC, with the former achieving the third highest F1, while RSR and RCA lie closer to CNC, with RSR exhibiting the third lowest F1.

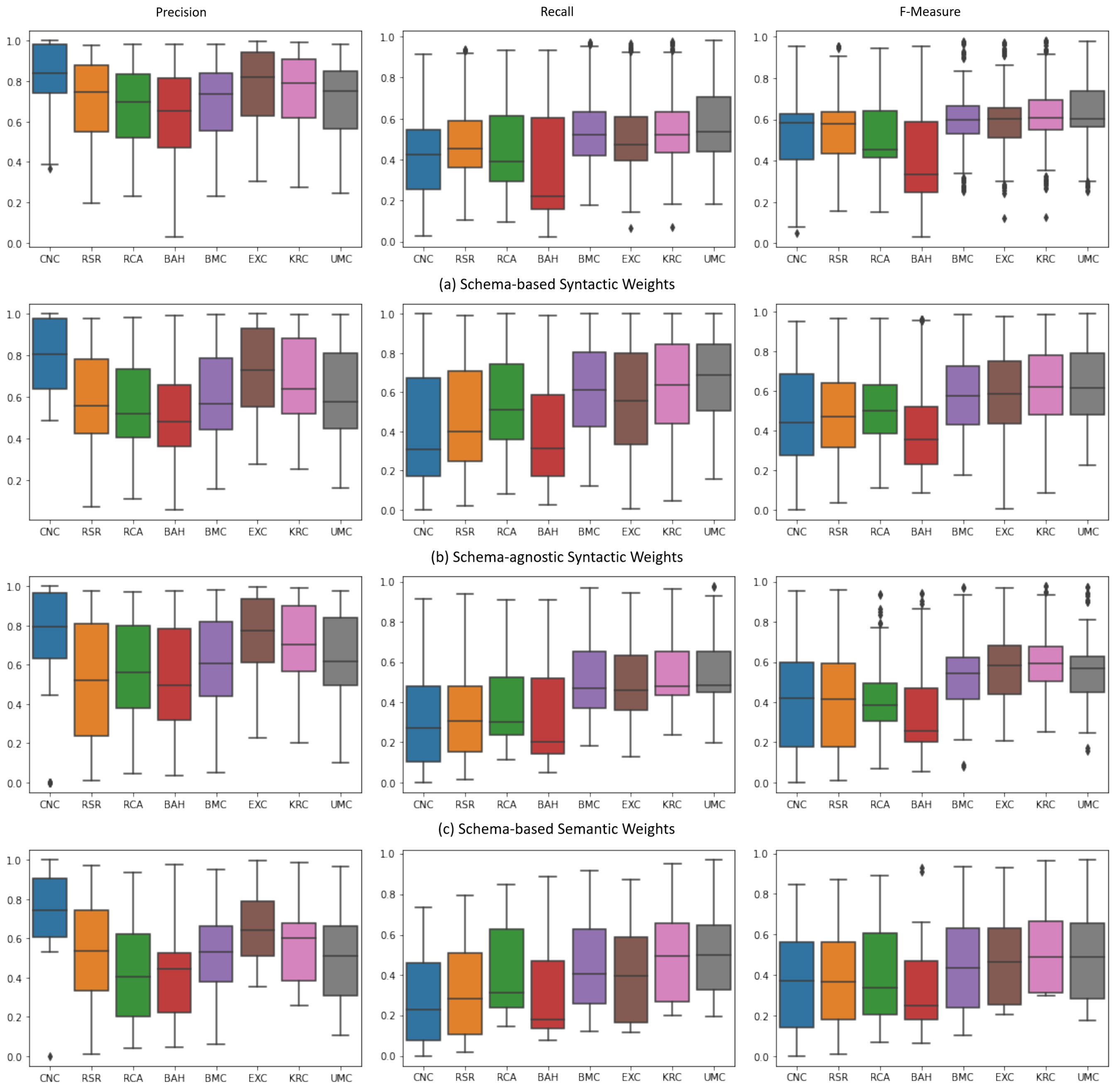
An interesting observation related to QE(2) is that the performance of all algorithms is highly correlated. The Pearson correlation between the precision, recall and F-Measure of all pairs of algorithms is consistently positive, taking values well above 0.6 in all cases. The correlation between EXC and KRC exceeds 0.96 for all three measures, while UMC is even more correlated with BMC. The only exception is BAH, whose correlation with all other algorithms fluctuates between 0.2 and 0.6, due to its stochastic nature. These patterns indicate that the higher the performance of an algorithm is over a particular similarity graph, the higher will be the performance of any other algorithm over the same input – with the sole exception of BAH.

Another interesting observation is that EXC consistently achieves higher precision and lower recall than BMC. This should be expected, given that EXC requires an additional reciprocity check before declaring that two entities match. We notice, however, that the gain in precision is greater than the loss in recall and, thus, EXC yields a higher F-Measure than BMC, on average. Note also that in the vast majority of cases, the best choice for the entity collection that BMC uses as the basis for creating clusters is the largest one.

To answer QE(3), Figure 3 presents the distribution of precision, recall and F-Measure of all algorithms across the four types of similarity graphs' origin. For the schema-based syntactic weights, we observe in Figure 3(a) that the average precision of all algorithms increases significantly in comparison to the one in Table 3 - from 4.0% (CNC) to 16.8% (BMC). For CNC and RSR, this is accompanied by an increase in average recall (by 7.7% and 5.1%, respectively), while for all other algorithms, the average recall drops between 4.8% (EXC) and 7.5% (KRC). This means that the schema-based syntactic similarities reinforce the imbalance between precision and recall in Table 3 in favor of the former for all algorithms except CNC and RSR. The average F-Measure drops only for KRC (by 0.2%), which is now outperformed by UMC. Similarly, BMC exceeds EXC in terms of average F1 (0.603 vs 0.599), because the increase in its mean precision is much higher than the decrease in its mean recall. Finally, it is worth noting that this type of input increases significantly the robustness of all algorithms, as the standard deviation of F1 drops by more than 12% for all algorithms, but the stochastic BAH.

The opposite patterns are observed for schema-agnostic syntactic weights in Figure 3(b): the imbalance between precision and recall is reduced, as on average, the former drops from 1.9% (EXC) to 5.3% (BAH), while the latter raises from 2.4% (RSR) to 7.6% (RCA). The imbalance is actually reversed for BMC and UMC, whose average recall (0.613 and 0.664, resp.) exceeds their average precision (0.606 and 0.622, resp.). The only exception is CNC, which increases both its average and average precision. Overall, there are minor, positive changes in the average F-Measure of most algorithms, with KRC and EXC retaining a minor edge over UMC and BMC, respectively.

Regarding the semantic similarity weights, we observe in Figure 3(c) similar patterns with the schema-based syntactic ones: precision increases for all algorithms except CNC and RSR, while recall drops in all cases. In this case, though, the latter change is stronger than the former, leading to lower average F-Measures than those in Table 3. In the case of schema-agnostic semantic weights, all measures in Figure 3(d) drop to a significant extend (>15% in most cases) when compared to Table 3. It is also remarkable that the standard deviation of all measures increases to a significant extent for both schema-based and schema-agnostic weights in relation to their syntactic counterparts, despite the fewer similarity graphs. As a result, the robustness of all algorithms over semantic weights



(a) Schema-based Syntactic Weights

(b) Schema-agnostic Syntactic Weights

(c) Schema-based Semantic Weights

(d) Schema-agnostic Semantic Weights

Figure 3: Precision (left), recall (center) and F-Measure (right) of all algorithms over similarity graphs with (a) schema-based syntactic, (b) schema-agnostic syntactic, (c) schema-based semantic, and (d) schema-agnostic semantic edge weights.

is limited. Nevertheless, KRC and EXC maintain a clear lead over UMC and MBC, respectively.

To answer QE(4), we distinguish the similarity graphs into three categories according to the portion of duplicates in their ground truth with respect to the size of $|V_1|$ and $|V_2|$:

(i) *Balanced* (BLC) are the entity collections where the vast majority of entities in V_i are matched with an entity in V_j (i=1 ∧ j=2 or i=2 ∧ j=1). This category includes all similarity graphs generated from D_2, D_4 and D_{10}.

(ii) *One-sided* (OSD) are the entity collections, where only the vast majority of entities in V_1 are matched with an entity from V_2, or vice versa. OSD includes all graphs stemming from D_3 and D_9.

(iii) *Scarce* (SCR) are the entity collections, where a small portion of entities in V_i are matched with an entity in V_j (i=1 ∧ j=2 or i=2 ∧ j=1). This category includes all graphs generated from D_1, D_5-D_8.

Table 4: The number of times each algorithm achieves the highest and second highest F1 for a particular similarity graph, $\#Top1$ and $\#Top2$, resp., as well as the average difference Δ (%) with the second highest F1 across all types of edge weights for balanced (BLC), one-sided (OSD) and scarce (SCR) entity collections. OVL stands for the overall sums or averages across all similarity graphs per category. Note that there are ties for both $\#Top1$ and $\#Top2$: 16 and 40, resp., over schema-based syntactic weights, 17 and 11, resp., over schema-agnostic syntactic weights, 9 and 2, resp., over schema-based semantic weights.

		Syntactic Similarities								Semantic Similarities							
		Schema-based				Schema-agnostic				Schema-based				Schema-agnostic			
		BLC	OSD	SCR	OVL	BLC	OSD	SCR	OVL	BLC	OSD	SCR	OVL	BLC	OSD	SCR	OVL
CNC	$\#Top1$	-	-	18	18	-	-	48	48	-	-	1	1	-	-	-	-
	Δ (%)	-	-	0.41	0.41	-	-	**7.59**	**7.59**	-	-	0.33	0.33	-	-	-	-
	$\#Top2$	-	-	8	8	-	-	8	8	-	-	4	4	-	-	-	-
RSR	$\#Top1$	-	-	4	4	-	-	1	1	-	-	1	1	-	-	-	-
	Δ (%)	-	-	**1.90**	1.90	-	-	0.51	0.51	-	-	0.33	0.33	-	-	-	-
	$\#Top2$	-	-	7	7	-	-	5	5	-	-	1	1	-	-	1	1
RCA	$\#Top1$	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	Δ (%)	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
	$\#Top2$	-	-	-	-	-	-	1	1	-	-	-	-	-	-	-	-
BAH	$\#Top1$	8	-	1	9	40	-	2	42	2	-	1	3	2	-	1	3
	Δ (%)	3.69	-	**1.90**	**3.49**	**5.55**	-	0.46	5.31	**12.72**	-	0.67	**8.70**	**13.96**	-	0.52	**9.48**
	$\#Top2$	8	-	3	11	7	5	6	18	-	-	2	2	-	-	-	-
BMC	$\#Top1$	-	-	6	6	-	-	7	7	-	-	2	2	-	-	-	-
	Δ (%)	-	-	0.57	0.57	-	-	1.41	1.41	-	-	0.21	0.21	-	-	-	-
	$\#Top2$	2	2	27	31	12	5	18	35	-	-	2	2	-	-	-	-
EXC	$\#Top1$	-	4	35	39	-	11	**79**	90	-	3	14	17	-	-	**5**	5
	Δ (%)	-	0.53	0.87	0.84	-	0.18	1.18	1.67	-	**1.62**	2.54	2.38	-	-	4.18	4.18
	$\#Top2$	-	10	31	41	-	19	72	91	-	**3**	**19**	22	-	**2**	2	4
KRC	$\#Top1$	**22**	**15**	**43**	**80**	16	**47**	**79**	**142**	**11**	**5**	**30**	**46**	**3**	**4**	4	**11**
	Δ (%)	1.36	1.60	0.35	0.87	4.15	2.54	4.05	3.56	2.92	1.47	**4.92**	4.07	4.06	**10.32**	**5.01**	6.68
	$\#Top2$	12	**17**	**57**	**86**	39	**40**	**87**	**166**	3	**3**	11	17	1	-	**5**	6
UMC	$\#Top1$	**22**	**15**	30	67	**58**	41	29	128	3	-	6	9	1	-	1	2
	Δ (%)	**4.99**	**1.75**	1.19	2.56	4.51	**3.21**	2.51	3.64	2.11	-	0.34	0.93	0.22	-	1.00	0.61
	$\#Top2$	**30**	5	28	63	**56**	30	42	128	**13**	2	9	**24**	**5**	**2**	3	**10**

We apply this categorization to the four main types of similarity graphs defined by Figure 2 and for each subcategory, we consider three new effectiveness measures:

(i) $\#Top1$ denotes the number of times an algorithm achieves the maximum F-Measure for a particular category of similarity graphs,

(ii) Δ (%) stands for the average difference (expressed as a percentage) between the highest and the second highest F1 across all similarity graphs of the same category, and

(iii) $\#Top2$ denotes the number of times an algorithm scores the second highest F1 for a particular category of similarity graphs.

Note that in case of ties, we increment $\#Top1$ and $\#Top2$ for all involved algorithms. Note also that these three effectiveness measures also allow for answering QE(2) in more detail.

The results for these measures are reported in Table 4. For schema-based syntactic weights, there is a strong competition between KRC and UMC for the highest effectiveness. Both algorithms achieve the maximum F1 for the same number of similarity graphs in the case of balanced and one-sided entity collections. For the former inputs, though, UMC exhibits consistently high performance, as it ranks second in almost all cases that it is not the top performer, unlike KRC, which comes second in 1/3 of these cases. Additionally, UMC achieves significantly higher Δ than KRC. For one-sided entity collections, KRC takes a minor lead over UMC: even though its Δ is slightly lower, it comes second three times more often than UMC. For scarce entity collections, KRC takes a clear lead over UMC, outperforming it with respect to both $\#Top1$ and $\#Top2$ to a large extent. UMC excels only with respect to Δ.

Among the remaining algorithms, CNC, RSR, BMC and EXC seem suitable only for scarce entity collections. RSR actually achieves the highest Δ, while EXC achieves the second highest $\#Top1$ and $\#Top2$, outperforming UMC. Regarding BAH, we observe that for balanced entity collections, it outperforms all algorithms for 15% of the similarity graphs, achieving the highest Δ and comes second for an equal number of inputs. This is in contrast to the poor average performance reported in Table 3, but is explained by its stochastic nature, which gives rise to an unstable performance, as indicated by the significantly higher σ than all other algorithms for all effectiveness measures.

In the case of schema-agnostic syntactic edge weights, UMC verifies its superiority over KRC for balanced entity collections with respect to all measures. KRC is actually outperformed by BAH, which achieves 2.5 times more often the top F1, while exhibiting the highest Δ among all algorithms. For one-sided entity collections, KRC excels with respect to $\#Top1$ and $\#Top2$, but UMC achieves significantly higher Δ, while EXC constitutes the third best algorithm overall, as for the schema-based syntactic edge weights. In the case of scarce entity collections, the two competing algorithms are KRC and EXC, as they share the highest $\#Top1$. Yet, the former achieves three times higher Δ and slightly higher $\#Top2$. Suprisingly, CNC ranks second in terms of $\#Top1$, while achieving the highest Δ by far, among all algorithms. As a result, UMC is left at the fourth place, followed by BMC.

For the semantic edge weights, we observe the following patterns: for the balanced entity collections, only KRC, BAH and UMC exhibit the highest performance for both schema-based and schema-agnostic weights. They excel in $\#Top1$, Δ and $\#Top2$, respectively. For one-sided entity collections, KRC is the dominant algorithm, especially in the case of schema-agnostic weights. For the schema-based ones, EXC consistently achieves very high performance, too. For scarce entity collections, there is a strong competition between KRC and EXC; the former consistently outperforms the latter with respect to Δ, while EXC excels in $\#Top1$ for schema-agnostic weights and in $\#Top2$ for schema-based ones.

We examined other patterns with respect to additional characteristics of the entity collections, such as the distribution of positive and negative weights (i.e., between matching and non-matching entities, respectively) and the domain (e-commerce for D_2, D_3 and D_8, bibliographic data for D_4 and D_9 as well as movies for D_5-D_8 and D_{10}). Yet, no clear patterns emerged in these cases.

6.2 Time Efficiency

The (relative) run-time of the evaluated algorithms is a crucial aspect for the task of Entity Resolution, due to the large size of the similarity graphs, which comprises thousands of entities/nodes and (hundreds of) millions of edges/entity pairs, as reported in Table 2. Below, we study this aspect along with the scalability of the considered algorithms over inputs of various types and sizes. More specifically, we examine the following aspects:

QT(1): Which algorithm is the fastest one?
QT(2): Which factors affect the run-time of the algorithms?
QT(3): How scalable are the algorithms to large input sizes?
QT(4): How does the relative run-time and/or scalability of the algorithms relates to their relative time complexity?

The average run-times over 10 executions of the evaluated algorithms per dataset and type of edge weights are reported in Table 5.

Table 5: The average run-time in milliseconds per clustering algorithm, dataset and type of input.

	CNC	RSR	RCA	BAH	BMC	EXC	KRC	UMC
D_1	2	10	7	1,889	2	3	14	4
D_2	7	22	23	1,149	9	55	77	44
D_3	17	41	66	2,265	21	29	198	68
D_4	26	63	157	2,111	34	61	392	273
D_5	35	87	643	2,734	43	48	344	110
D_6	51	89	713	2,997	52	56	456	78
D_7	19	42	932	3,141	24	24	167	27
D_8	171	257	1,386	122,644	176	203	1,905	476
D_9	1,277	1,238	2,847	146,630	1,675	20,795	14,925	2,857
D_{10}	1,391	2,833	24,300	129,113	1,827	2,028	139,458	10,006

(a) **Schema-based, syntactic inputs**

D_1	12	19	13	1,888	15	22	51	22
D_2	7	19	20	1,083	11	17	53	63
D_3	27	40	58	2,174	37	84	614	506
D_4	37	55	127	2,163	39	40	362	84
D_5	80	238	756	3,429	155	273	989	1,007
D_6	72	137	843	3,731	89	184	651	179
D_7	82	141	1,075	3,748	98	138	615	290
D_8	289	347	719	124,063	315	408	2,729	300
D_9	437	740	2,038	146,301	528	1,613	5,920	2,556
D_{10}	1,861	8,429	24,845	129,942	3,394	19,457	190,731	159,643

(b) **Schema-agnostic, syntactic inputs**

D_1	9	16	44	1,819	17	20	42	27
D_2	8	24	16	1,029	10	30	122	71
D_3	28	46	124	2,210	225	472	2,312	3,566
D_4	35	171	82	2,244	156	255	1,250	1,369
D_5	76	456	1462	2,874	230	375	1,295	1,637
D_6	118	752	628	3,202	188	684	2,135	2,781
D_7	26	126	550	2,872	91	145	521	669
D_8	236	286	180,242	122,843	258	3,382	2,036	368
D_9	833	15311	172,140	140,556	2,262	20,164	41,960	49,823
D_{10}	2,650	4,506	3,291	127,030	1,1824	5,003	340,066	1,031,259

(c) **Schema-based, semantic inputs**

D_1	14	64	108	1,640	15	16	76	24
D_2	8	35	26	959	48	91	275	473
D_3	23	36	207	1,836	29	336	3,256	4,010
D_4	34	62	152	1,884	46	54	2,733	218
D_5	171	242	883	2,202	202	504	1,746	406
D_6	436	305	1,496	2,545	565	1,010	3,716	3,684
D_7	207	865	1,510	2,972	2,384	3,432	8,156	13,743
D_8	-	-	-	-	-	-	-	-
D_9	824	1,138	402,886	143,788	1,140	23,784	51,590	4,339
D_{10}	-	-	-	-	-	-	-	-

(d) **Schema-agnostic, semantic inputs**

Regarding QT(1), we observe that all algorithms are quite fast, as they are all able to process huge similarity graphs (as in the case of D_9 and D_{10}) within minutes or even seconds. CNC is the fastest method almost in all cases, due to the simplicity of its approach. It is followed in close distance by BMC and RSR, with the former consistently outperforming the latter. EXC is also very time efficient, but as expected, it is always slower than BMC, due to the additional reciprocity check it involves. On the other extreme lies BAH, which constitutes by far the slowest method, yielding in many cases by 2 or even 3 orders of magnitude longer run-times. The reason is the large number of search steps we allow per dataset (100,000). For the largest datasets, its maximum run-time actually equals the run-time limit of 2 minutes, except for D_9, where the very large number of entities in V_2 delays the activation of the limit. The rest of the algorithms lie between these two extremes: KRC is the slowest one, on average, while UMC and RCA exhibit significantly lower run-times. Overall, among the most sophisticated and effective algorithms, EXC is faster than UMC, which is faster than KRC.

Regarding QT(2), two are the main factors that affect the reported run-times: (i) the theoretical time complexity of the algorithms, and (ii) the similarity threshold that is used for pruning the search space. Regarding the first factor, we observe that the run-times in Table 5 verify the time complexities described in Section 3. With $O(m)$, CNC and BMC are the fastest ones, followed by RSR and EXC with

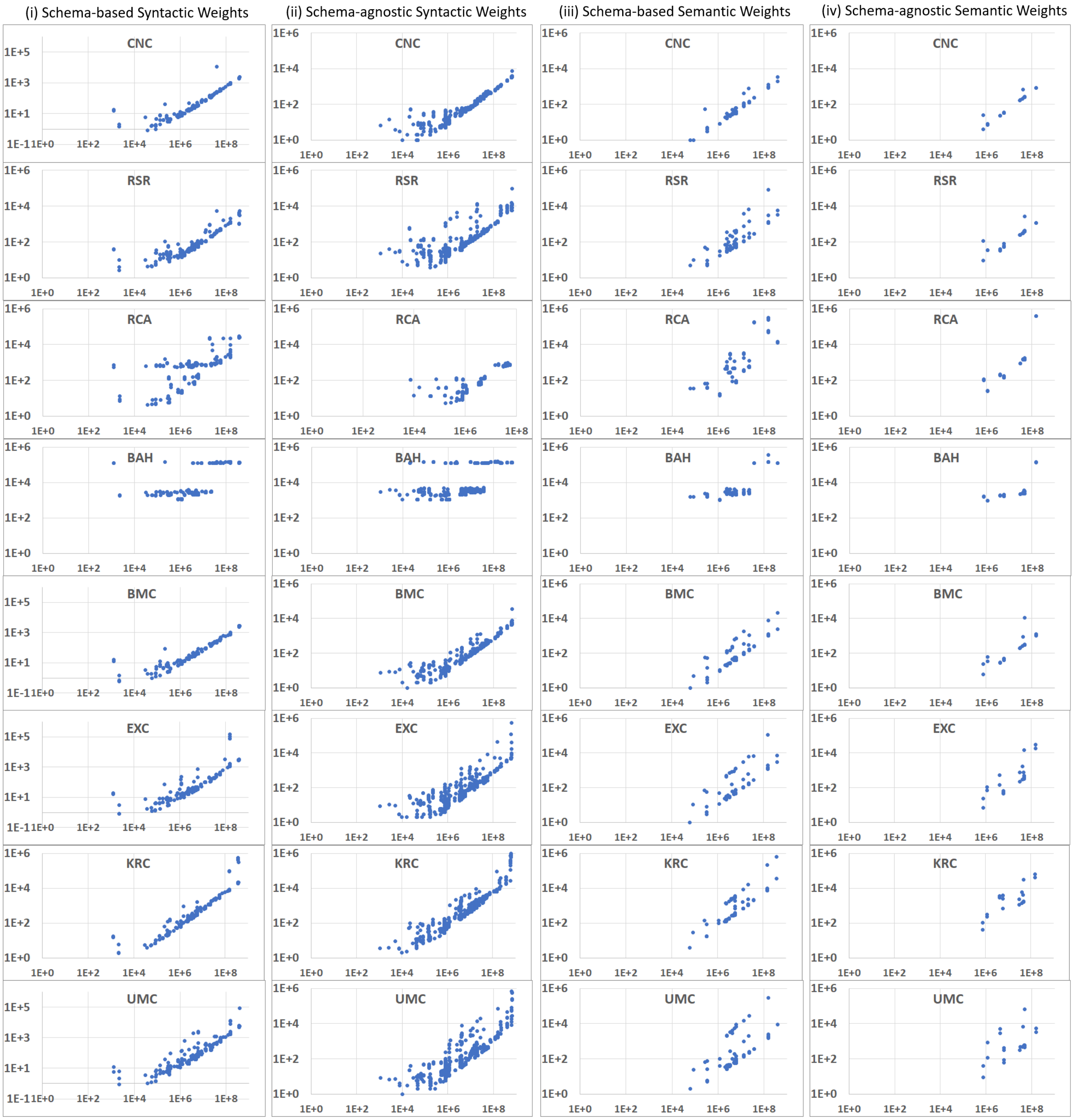


Figure 4: Scalability analysis of all algorithms over all similarity graphs with (i) schema-based syntactic, (ii) schema-agnostic syntactic, (iii) schema-based semantic and (iv) schema-agnostic semantic edge weights. The horizontal axis corresponds to the number of edges in the similarity graphs and the vertical one to the run-time in milliseconds (maximum value=16.7 min).

$O(n\ m)$, RCA with $O(|V_1|\ |V_2|)$, UMC with $O(m\ \ \log m)$ and KRC with $O(n + m\ \ \log m)$. BAH's run-time is dominated by the number of search steps and the run-time limit.

More interesting is the effect of similarity thresholds: the higher they are, the fewer edges are retained in the similarity graph and the faster is its processing. This applies

since many of the evaluated algorithms have similar worst-case time complexity of $O(n\ \ logn)$ or $O(n\ \ logm)$, an important factor that makes a difference in their relative run-times is the number of scans that they perform over the input similarity graph. For example, BMC requires a single pass over the nodes, while EXC and RCA require two passes. We can also confirm from the run-times that scanning the edges vs scanning the nodes of the input similarity graph makes a big difference, since the number of edges m (the Cartesian product $E_1 \times E_2$ of the edges in the two datasets) is typically greater than the number of nodes n by many orders of magnitude, as shown in Table 1. For example, UMC ($O(m\ logm)$) and KRC ($O(n\ +\ m\ logm)$) are much slower than RSR, BMC and RCA (not true!) ($O(n\ logm)$).

Regarding QT(3), Figure 4 presents the scalability of each algorithm over different input types and sizes. In this figure, every point corresponds to a different input similarity graph. Hence, different similarities on the same dataset yield different input sizes of non-zero similarities.

We can make the following observations: (a) The scalability of BAH and RCA, as shown in Figure 4, is not representative of the algorithms. The reason is that for BAH, we terminate the algorithm after pre-defined timeout or a fixed number of iterations (whichever comes first), thus, the step-resembling scalability graph. For RCA …(are run-time and scalability exps up-to-date?) (b) All other methods (except BAH and RCA) show a similar behavior with respect to scalability, with very few outlier points (points that deviate from the "central" curve). We observe more such deviations in UMC, KRC and EXC, which are also the methods that seem to scale well until roughly 10^8 edges (input comparisons).

We also note that, even if the datasets may seem small, given that we do not apply any blocking method, the size of the input to the clustering methods (reported in the last row of Table 1) ranges from hundreds of thousands to tens and hundreds of millions pairs.

Regarding QT(4), we observe that the time complexity of the algorithm fits well with the observed scalability presented in Figure 4. Again, for the same reasons explained in the discussion of QT(3), BAH and RCA are exceptions.

7 CONCLUSIONS

The most important patterns we draw from our experiments are the following:

There is no clear winner among the considered algorithms. Yet, KRC and UMC offer the most robust performance.

- Which clustering algorithm offers the best trade-off between effectiveness and time-efficiency?

Our focus in this work has been on entity resolution in an unsupervised setting where large enough training data is not available. Following the approach of Reas et al. [42], an interesting avenue for future work is a supervised learning system incorporating a variety of matching algorithms in a unified framework and effectively learning the right parameters for a given data set.

OUTDATED DRAFT - IGNORE:

- UMC and Kiraly seem to perform best overall (wrt F-Measure) for schema-based inputs, while (UMC and) RSR perform(s) slightly better for schema-agnostic inputs.
- CC has the highest Precision in both schema-based and schema-agnostic inputs. In schema-based methods, its Recall is a bit worse than most of the other methods, resulting in comparable F-Measures. This is not true, however for the schema-agnostic inputs, where CC has much lower Recall than the rest, which is also reflected in the low F-Measures. Hence, we recommend CC for cases in which Precision is more important than Recall and there is a fixed schema.
- UMC has the highest Recall, especially in schema-agnostics inputs, and comparable Precision results to most of the other methods, hence, we recommend UMC for cases in which Recall is more important than Precision, as well as for a good balance between Precision and Recall (as stated before).
- BAH has the worst results in terms of both Precision and Recall, as well as one of the slowest clustering methods, so we would not recommend using this method.
- While there is a bigger diversity across the methods in the Precision and Recall numbers, this is not reflected in the F-Measure, where most methods have comparable results, with a few exceptions (BAH, RCA, CC).
- CC is the fastest clustering method. RCA and BAH are the slowest. Regarding the top-performing methods wrt effectiveness, UMC is in most cases much faster than Kiraly.
- In all methods and all datasets, the best settings are achieved when the similarity thresholds for schema-agnostic inputs are much lower than for the schema-based ones. We also observe that the high Precision of CC is in line with the high similarity thresholds employed, while UMC and Kiraly work well with much lower thresholds.
- Scalability?

- Király's Clustering seems to outperform the rest in most cases (winner in 5 out of 10 datasets), with UMC and Center Clustering tied in second place (winners in 3 out of 10 datasets, each). In two datasets (Walmart-Amazon and IMDB-TMDB), there is a tie in the winning clustering algorithm.
- Simon White and q-gram are the best performing similarity measures in 8 out of 10 datasets.
- BAH and RCA are the slowest clustering methods in almost all datasets.
- CCCER is the fastest clustering algorithm in 7 out of 10 datasets.

Future work:

* Use meta-blocking scores as input

* Assess the robustness/sensitivity of clustering methods, i.e., how much their optimal performance changes if part of the input data is slightly different.

* Consider DER datasets and DER clustering algorithms

REFERENCES

[1] Fotis Aisopos, George Papadakis, Konstantinos Tserpes, and Theodora A. Varvarigou. 2012. Content vs. context for sentiment analysis: a comparative analysis over microblogs. In *ACM Conference on Hypertext and Social Media*. 187–196.

[2] Ali Assi, Hamid Mcheick, and Wajdi Dhifli. 2019. BIGMAT: A Distributed Affinity-Preserving Random Walk Strategy for Instance Matching on Knowledge Graphs. In *IEEE Big Data*. 1028–1033.

[3] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146.

[4] Ursin Brunner and Kurt Stockinger. 2020. Entity Matching with Transformer Architectures - A Step Forward in Data Integration. In *EDBT*. 463–473.

[5] Peter Christen. 2012. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection*. Springer.

[6] Vassilis Christophides, Vasilis Efthymiou, Themis Palpanas, George Papadakis, and Kostas Stefanidis. 2021. An Overview of End-to-End Entity Resolution for Big Data. *ACM Comput. Surv.* 53, 6 (2021), 127:1–127:42.

[7] Vassilis Christophides, Vasilis Efthymiou, and Kostas Stefanidis. 2015. *Entity Resolution in the Web of Data.* Morgan & Claypool Publishers.

[8] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. 2008. *Algorithms.* McGraw-Hill.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *NAACL-HLT.* 4171–4186.

[10] Xin Luna Dong and Divesh Srivastava. 2015. *Big Data Integration.* Morgan & Claypool Publishers.

[11] Vasilis Efthymiou, George Papadakis, Kostas Stefanidis, and Vassilis Christophides. 2019. MinoanER: Schema-Agnostic, Non-Iterative, Massively Parallel Resolution of Web Entities. In *EDBT.* OpenProceedings.org, 373–384.

[12] I. P. Fellegi and A. B. Sunter. 1969. A Theory for Record Linkage. *J. Amer. Statist. Assoc.* 64, 328 (1969), 1183–1210.

[13] Michael L. Fredman and Robert Endre Tarjan. 1987. Fibonacci heaps and their uses in improved network optimization algorithms. *J. ACM* 34, 3 (1987), 596–615.

[14] D. Gale and L. S. Shapley. 1962. College Admissions and the Stability of Marriage. *Am. Math. Mon.* 69, 1 (1962), 9–15.

[15] Jim Gemmell, Benjamin I. P. Rubinstein, and Ashok K. Chandra. 2011. Improving Entity Resolution with Global Constraints. *CoRR* abs/1108.6016 (2011).

[16] George Giannakopoulos, Vangelis Karkaletsis, George A. Vouros, and Panagiotis Stamatopoulos. 2008. Summarization system evaluation revisited: N-gram graphs. *ACM Trans. Speech Lang. Process.* 5, 3 (2008), 5:1–5:39.

[17] Claudio Gutierrez and Juan F. Sequeda. 2020. *Knowledge Graphs: A Tutorial on the History of Knowledge Graph's Main Ideas.* Association for Computing Machinery, 3509–3510. https://doi.org/10.1145/3340531.3412176

[18] Oktie Hassanzadeh, Fei Chiang, Renée J. Miller, and Hyun Chul Lee. 2009. Framework for Evaluating Clustering Algorithms in Duplicate Detection. *Proc. VLDB Endow.* 2, 1 (2009), 1282–1293.

[19] Zoltán Király. 2013. Linear Time Local Approximation Algorithm for Maximum Stable Marriage. *Algorithms* 6, 3 (2013), 471–484.

[20] Pradap Konda, Sanjib Das, Paul Suganthan G. C., AnHai Doan, Adel Ardalan, Jeffrey R. Ballard, Han Li, Fatemah Panahi, Haojun Zhang, Jeffrey F. Naughton, Shishir Prasad, Ganesh Krishnan, Rohit Deep, and Vijay Raghavendra. 2016. Magellan: Toward Building Entity Matching Management Systems. *Proc. VLDB Endow.* 9, 12 (2016), 1197–1208.

[21] Hanna Köpcke, Andreas Thor, and Erhard Rahm. 2010. Evaluation of entity resolution approaches on real-world match problems. *Proc. VLDB Endow.* 3, 1 (2010), 484–493.

[22] Nils M. Kriege, Pierre-Louis Giscard, Franka Bause, and Richard C. Wilson. 2019. Computing Optimal Assignments in Linear Time for Approximate Graph Matching. In *ICDM.* 349–358.

[23] Harold W. Kuhn. 2010. The Hungarian Method for the Assignment Problem. In *50 Years of Integer Programming 1958-2008 - From the Early Years to the State-of-the-Art.* Springer, 29–47.

[24] H. W. Kuhn and Bryn Yaw. 1955. The Hungarian method for the assignment problem. *Naval Res. Logist. Quart* (1955), 83–97.

[25] Jerome M Kurtzberg. 1962. On approximation methods for the assignment problem. *Journal of the ACM (JACM)* 9, 4 (1962), 419–439.

[26] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2020. ALBERT: A Lite BERT for Self-supervised Learning of Language Representations. In *ICLR.*

[27] Yuliang Li, Jinfeng Li, Yoshihiko Suhara, Jin Wang, Wataru Hirota, and Wang-Chiew Tan. 2021. Deep Entity Matching: Challenges and Opportunities. *ACM J. Data Inf. Qual.* 13, 1 (2021), 1:1–1:17.

[28] L. Lovasz and M. D. Plummer. [n.d.]. *Matching theory.*

[29] Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schütze. 2008. *Introduction to information retrieval.* Cambridge University Press.

[30] Sergey Melnik, Hector Garcia-Molina, and Erhard Rahm. 2002. Similarity Flooding: A Versatile Graph Matching Algorithm and Its Application to Schema Matching. In *ICDE.* 117–128.

[31] Tomás Mikolov, Ilya Sutskever, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Distributed Representations of Words and Phrases and their Compositionality. In *NIPS.* 3111–3119.

[32] Sidharth Mudgal, Han Li, Theodoros Rekatsinas, AnHai Doan, Youngchoon Park, Ganesh Krishnan, Rohit Deep, Esteban Arcaute, and Vijay Raghavendra. 2018. Deep Learning for Entity Matching: A Design Space Exploration. In *SIGMOD.* 19–34.

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

[34] Boris Otto and Andreas Reichert. 2010. Organizing Master Data Management: Findings from an Expert Survey. In *Proceedings of the 2010 ACM Symposium on Applied Computing (SAC).* 106–110. https://doi.org/10.1145/1774088.1774111

[35] George Papadakis, George Giannakopoulos, and Georgios Paliouras. 2016. Graph vs. bag representation models for the topic classification of web documents. *World Wide Web* 19, 5 (2016), 887–920.

[36] George Papadakis, Ekaterini Ioannou, Claudia Niederée, and Peter Fankhauser. 2011. Efficient entity resolution for large heterogeneous information spaces. In *WSDM.* 535–544.

[37] George Papadakis, Ekaterini Ioannou, Emanouil Thanos, and Themis Palpanas. 2021. *The Four Generations of Entity Resolution.* Morgan & Claypool Publishers.

[38] George Papadakis, Georgios M. Mandilaras, Luca Gagliardelli, Giovanni Simonini, Emmanouil Thanos, George Giannakopoulos, Sonia Bergamaschi, Themis Palpanas, and Manolis Koubarakis. 2020. Three-dimensional Entity Resolution with JedAI. *Inf. Syst.* 93 (2020), 101565.

[39] George Papadakis, George Papastefanatos, Themis Palpanas, and Manolis Koubarakis. 2016. Boosting the Efficiency of Large-Scale Entity Resolution with Enhanced Meta-Blocking. *Big Data Res.* 6 (2016), 43–63.

[40] George Papadakis, Dimitrios Skoutas, Emmanouil Thanos, and Themis Palpanas. 2020. Blocking and Filtering Techniques for Entity Resolution: A Survey. *ACM Comput. Surv.* 53, 2 (2020), 31:1–31:42. https://doi.org/10.1145/3377455

[41] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global Vectors for Word Representation. In *EMNLP.* 1532–1543.

[42] Russell Reas, Steve Ash, Rob Barton, and Andrew Borthwick. 2018. SuperPart: Supervised Graph Partitioning for Record Linkage. In *IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17-20, 2018.* IEEE Computer Society, 387–396. https://doi.org/10.1109/ICDM.2018.00054

[43] Alieh Saeedi, Markus Nentwig, Eric Peukert, and Erhard Rahm. 2018. Scalable Matching and Clustering of Entities with FAMER. *Complex Syst. Informatics Model. Q.* 16 (2018), 61–83.

[44] Alieh Saeedi, Eric Peukert, and Erhard Rahm. 2018. Using Link Features for Entity Clustering in Knowledge Graphs. In *ESWC (Lecture Notes in Computer Science),* Vol. 10843. Springer, 576–592.

[45] Justus Schwartz, Angelika Steger, and Andreas Weißl. 2005. Fast Algorithms for Weighted Bipartite Matching. In *WEA (Lecture Notes in Computer Science),* Vol. 3503. 476–487.

[46] Yansheng Wang, Yongxin Tong, Cheng Long, Pan Xu, Ke Xu, and Weifeng Lv. 2019. Adaptive Dynamic Bipartite Graph Matching: A Reinforcement Learning Approach. In *ICDE.* 1478–1489.

[47] Zhengyang Wang, Bunyamin Sisman, Hao Wei, Xin Luna Dong, and Shuiwang Ji. 2020. CorDEL: A Contrastive Deep Learning Approach for Entity Linkage. In *ICDM.*

[48] Christopher J. C. H. Watkins and Peter Dayan. 1992. Technical Note Q-Learning. *Mach. Learn.* 8 (1992), 279–292.

[49] Derry Tanti Wijaya and Stéphane Bressan. 2009. Ricochet: A family of unconstrained algorithms for graph clustering. In *International Conference on Database Systems for Advanced Applications.* Springer, 153–167.

[50] W. E. Winkler. 2006. *Overview of Record Linkage and Current Research Directions.* Technical Report. Bureau of the Census.

APPENDIX

Connected Components (CNC). This is the simplest algorithm: it discards all edges with a weight lower than the similarity threshold and then computes the transitive closure of the pruned similarity graph. In the output, it solely retains the partitions that contain two entities – one from each input dataset. Using a simple depth-first approach, its time complexity is $O(n + m)$ [8].

Ricochet Sequential Rippling Clustering (RSR). This algorithm is an adaptation of the homonymous method for Dirty ER in [18] such that it exclusively considers partitions with just one entity from each input dataset. After pruning the edges weighted lower than t, RSR sorts all nodes from both input datasets in descending order of the average weight of their adjacent edges (Line 7). Whenever a new seed is chosen from the sorted list, the first adjacent vertex that is currently unassigned or is closer to the new seed than it is to the seed of its current partition is re-assigned to the new partition (Lines 14-16). If a partition is reduced to a singleton after a re-assignment, either because the chosen vertex (Line 17) or the seed (Line 24) was previously in it, it is placed in its nearest single-node partition (Lines 30-39). The algorithm stops when all nodes have been considered. In the worst case the algorithm has

Algorithm 1: Ricochet SR CC Clustering

```
Input: Similarity Graph $SE = (V_1, V_2, E)$, similarity threshold $t$
Output: A set of Partitions $C = \{c_1, c_2, \ldots, c_n\}$
$C \leftarrow \emptyset$
$Center \leftarrow \emptyset$
foreach $v \in SE$ do
    $simWithCenter(v) \leftarrow 0$
    $Partition(v) \leftarrow \emptyset$                // Partitions of vertices
    $centerOf(v) \leftarrow v$
$Q \leftarrow SE.vertices(v, w(v))$       // $SE$'s vertices in desc. vertex weight
  $w(v) = \sum_{e \in adj(v)} e.sim / |adj(v)|$
while $Q \neq \emptyset$ do
    $v_i \leftarrow Q.pop()$                // the vertex with highest weight
    $ToReassign \leftarrow \emptyset$
    foreach $e = (v_i, v_j, sim) \in E : sim > t$ do // for all edges adjacent to
      $v_i$ with $sim > t$
        if $v_j \in Center$ then
            continue
        if $e.sim > simWithCenter(v_j)$ then
            $Partition(centerOf(v_j)).remove(v_j)$ // remove $v_j$ from its
              previous partition
            $Partition(v_i) \leftarrow Partition(v_i) \cup v_j$
            $ToReassign \leftarrow ToReassign \cup centerOf(v_j)$   // it is now a
              singleton
            $simWithCenter(v_j) \leftarrow e.sim$
            $centerOf(v_j) \leftarrow v_i$
            break
    if $|c_i| > 0$ then
        if $centerOf(v_i) \neq v_i$ then      // if $v_i$ was previously in another
          partition

            $Partition(centerOf(v_i)).remove(v_i)$          // remove it
            $ToReassign \leftarrow ToReassign \cup centerOf(v_i)$
        $Center \leftarrow Center \cup v_i$
        $Partition(v_i) \leftarrow Partition(v_i) \cup v_i$     // put $v_i$ in its partition
        $centerOf(v_i) \leftarrow v_i$
        $simWithCenter(v_i) \leftarrow 1$
    foreach $v_k \in ToReassign$ do
        $maxSim \leftarrow 0$
        $cMax \leftarrow v_k$
        foreach $e = (v_k, v_\ell, sim) \in E : sim > t$ do // find singleton with
          the highest similarity with $v_k$ to reassign it
            if $e.sim > maxSim$ and $|Partition(v_\ell)| < 2$ then
                $cMax \leftarrow v_\ell$
                $maxSim \leftarrow e.sim$
        if $maxSim > 0$ then
            $Partition(v_k) \leftarrow \emptyset$
            $Partition(cMax) \leftarrow Partition(cMax) \cup v_k$
foreach $v_i \in SE$ do
    if $|Partition(v_i)| > 0$ then
        $C \leftarrow C \cup Partition(v_i)$
return $C$
```

to iterate through n vertices and each time reassign n vertices to their most similar adjacent vertex, therefore its time complexity is $O(n\ m)$ [49].

Algorithm 2: Row Column Clustering

```
   Input: Similarity Graph $SE = (V_1, V_2, E)$, similarity threshold $t$
   Output: A set of partitions $C = \{c_1, c_2, \ldots, c_n\}$
1  $C_1 \leftarrow \emptyset$
2  $C_2 \leftarrow \emptyset$
3  $M_1 \leftarrow \emptyset$                              // matched nodes from $V_1$
4  $M_2 \leftarrow \emptyset$                              // matched nodes from $V_2$
5  $D_1 \leftarrow 0$                              // assignment value of $C_1$
6  $D_2 \leftarrow 0$                              // assignment value of $C_2$
7  foreach $v_i \in V_1$ do
8      $c_i \leftarrow \{v_i\}$               // create a new partition containing $v_i$
9      $Q_i \leftarrow V_2(sim(v_i))$        // a PQ of $V_2$'s nodes in desc. sim with $v_i$
10     while $Q_i \neq \emptyset$ do
11         $v_2 \leftarrow Q_i.pop()$
12         if $v_2 \notin M_2$ then // if $v_2$ is not yet matched
13             $c_i \leftarrow c_i \cup \{v_2\}$                 // add $v_2$ to partition $c_i$
14             $M_2 \leftarrow M_2 \cup \{v_2\}$
15             $D_1 \leftarrow D_1 + \{sim(v_1, v_2)\}$
16             break
17     $C_1 \leftarrow C_1 \cup \{c_i\}$
18 foreach $v_j \in V_2$ do
19     $c_j \leftarrow \{v_j\}$               // create a new partition containing $v_j$
20     $Q_j \leftarrow V_1(sim(v_j))$        // a PQ of $V_1$'s nodes in desc. sim with $v_j$
21     while $Q_j \neq \emptyset$ do
22         $v_1 \leftarrow Q_j.pop()$
23         if $v_1 \notin M_2$ then // if $v_1$ is not yet matched
24             $c_j \leftarrow c_j \cup \{v_1\}$                 // add $v_1$ to partition $c_j$
25             $M_1 \leftarrow M_1 \cup \{v_1\}$
26             $D_2 \leftarrow D_2 + \{sim(v_1, v_2)\}$
27             break
28     $C_2 \leftarrow C_2 \cup \{c_j\}$
29 if $D_1 > D_2$ then // get maximal assignment
30     $C = C_1$
       // create singular partitions for unmatched nodes $v$
31     foreach $v \in V_2 \setminus M_2$ do
32         $C \leftarrow C \cup \{\{v\}\}$
33 else
34     $C = C_2$
       // create singular partitions for unmatched nodes $v$
35     foreach $v \in V_1 \setminus M_1$ do
36         $C \leftarrow C \cup \{\{v\}\}$
37 foreach $c = \{v_i, v_j\} \in C$ do
38     if $sim(v_i, v_j) < t$ then // check similarities
39         $C = C \setminus c \cup \{\{v_i\}, \{v_j\}\}$
           // separate partition pairs with similarity less than $t$
40 return $C$
```

Row Column Assignment Clustering (RCA). This approach is based on the Row-Column Scan approximation method in [25] that solves the assignment problem. It requires two passes of the similarity graph, with each pass generating a candidate solution. In the first pass, each entity from the source dataset creates a new partition, to which the most similar, currently unassigned entity from the target dataset is assigned (Lines 11-16). Note that, in principle, any pair of entities can be assigned to the same partition at this step even if their similarity is lower than t, since the assignment problem assumes that each vertex from $V1$ is connected to all vertices from $V2$ (any "job" can be performed by all "men"). The partitions of pairs with similarity less than t are then discarded. In the second pass, the same procedure is applied to the entities/nodes of the target dataset. The value of each solution is the sum of the edge weights between the nodes assigned to the same (2-node) partition (Lines 17,28). The solution with the highest value is returned as output (Lines 29-36). At each pass the algorithm iterates over all nodes of one of the datasets searching for the node with maximum similarity from the other dataset, therefore its time complexity is $O(|V1|\ |V2|)$.

Best Assignment Heuristic (BAH). This algorithm applies a simple swap-based random-search algorithm to heuristically solve the Maximum Weight Bipartite Matching problem and uses the resulting solution to create the output partitions. Initially, each entity from the smaller input dataset is randomly connected to an entity from the larger input dataset (Line 9). In each iteration of the search process, two entities from the larger dataset are randomly selected (Lines 15-16) in order to swap their current connections. If the sum of the edge weights of the new pairs is higher than the previous pairs (Line 22), the swap is accepted (Lines 23-27). The algorithm stops when a maximum number of search steps is

Algorithm 3: Best Assignment Heuristic Clustering

```
Input: Similarity Graph $SE = (V_1, V_2, E) : |V_1| > |V_2|$, similarity
       threshold $t$, max number of moves $maxNumMoves$
Output: A set of partitions $C = \{c_1, c_2, \dots, c_n\}$
$C \leftarrow \emptyset$
$numMoves \leftarrow 0$
foreach $(v^1_i, v^2_j) \in (V_1 \times V_2)$ do
    $d(v^1_i, v^2_j) \leftarrow 0$        // initialize pair contributions
foreach $e = (v^1_i, v^2_j, sim) \in E$, with $e.sim > t$ do
    $d(v^1_i, v^2_j) \leftarrow sim$      // initialize pair contributions
foreach $v^1_i \in V_1, v^2_i \in V_2 : i \leq |V_2|$ do
    $c_i \leftarrow \{v^1_i, v^2_i\}$                 // initialize partitions
    $p(v^1_i) = v^2_i$
foreach $v^1_i \in V_1 : i > |V_2|$ do
    $c_i \leftarrow \{v^1_i\}$                      // initialize singletons
    $p(v^1_i) = null$
while $numMoves < maxNumMoves$ do
    $numMoves \leftarrow numMoves + 1$
    $i = nextRand(|V_1|)$
    $j = nextRand(|V_1|) : j \neq i$
    $D \leftarrow 0$
    if $p(v^1_i) \neq null$ then // check swaps
        $D \leftarrow d(v^1_j, p(v^1_i)) - d(v^1_i, p(v^1_i))$
    if $p(v^1_j) \neq null$ then // check swaps
        $D \leftarrow D + d(v^1_i, p(v^1_j)) - d(v^1_j, p(v^1_j))$
    if $D \geq 0$ then // if swaps increase assignment value
        $help \leftarrow p(v^1_j)$                     // perform swaps
        $p(v^1_j) \leftarrow p(v^1_i)$
        $p(v^1_i) \leftarrow help$
        $c_i \leftarrow \{v^1_i, p(v^1_i)\}$
        $c_j \leftarrow \{v^1_j, p(v^1_j)\}$
return $C$
```

reached or when a maximum run-time has been exceeded. In our case, the run-time has been set to 2 minutes.

Algorithm 4: Best Match Clustering

```
   Input: Similarity Graph $SE = (V_1, V_2, E)$, similarity threshold $t$
   Output: A set of clusters $C = \{c_1, c_2, \dots, c_n\}$
1  $C \leftarrow \emptyset$
2  $M_2 \leftarrow \emptyset$                          // matched nodes from $V_2$
3  foreach $v_i \in V_1$ do
4      $c_i \leftarrow \{v_i\}$   // create a new cluster containing $v_i$
5      $Q_i \leftarrow SE.edges(v_i, t)$ // $v_i$'s edges in desc. sim > $t$
6      while $Q_i \neq \emptyset$ do
7          $e \leftarrow Q_i.pop()$
8          if $e.v_2 \notin M_2$ then // if $v_2$ is not yet matched
9              $c_i \leftarrow c_i \cup \{e.v_2\}$      // add $v_2$ to cluster $c_i$
10             $M_2 \leftarrow M_2 \cup \{e.v_2\}$
11             break
12     $C \leftarrow C \cup \{c_i\}$
13 foreach $v_j \in V_2 \setminus M_2$ do // for each unmatched $V_2$ node
14     $C \leftarrow C \cup \{\{v_j\}\}$      // add a new singular cluster
15 return $C$
```

Best Match Clustering (BMC). This algorithm is inspired from the Best Match strategy of [30], which solves the Stable Marriage problem [14], as simplified in BigMat [2]. For each entity of the one dataset, this algorithm creates a new partition (Lines 4-5), in which the most similar, not-yet-clustered entity from the other dataset is also placed - provided that the corresponding edge weight is higher than t (Lines 6-12). Note that the greedy heuristic for BMC introduced in [30] is the same, in principle, to Unique Mapping Clustering (see below). Note also that BMC is the only algorithm with an additional configuration parameter, apart from the similarity threshold: the input dataset that is used as the basis for creating partitions can be set to the source or the target dataset. In our experiments, we examine both options and retain the best one. The algorithm iterates over the nodes of one of the datasets searching for its adjacent vertex with maximum similarity, therefore its time complexity is $O(m)$ [].

Exact Clustering (EXC). This algorithm is inspired from the Exact strategy of [30]. EXC places two entities in the same partition (Lines 10-18) only if they are mutually the best matches, i.e., the most similar candidates of each other, and their edge weight exceeds t (Lines 13, 15). This approach is basically a stricter, symmetric version of BMC and could also be conceived as a strict version of the reciprocity filter that was employed in [11, 39]. Its time complexity is $O(n\ m)$, since the algorithm iterates over each vertex of one dataset searching for its adjacent vertex with maximum similarity and then performs the same search for the latter vertex [].

Király's Clustering (KRC). This algorithm is an adaptation of the linear time 3/2 approximation to the Maximum Stable Marriage problem, called "New Algorithm" in [19]. Intuitively, the entities of the source dataset ("men" [19]) propose to the entities (Line 16) from the target dataset with an edge weight higher than t ("women" [19]) to form a partition ("get engaged" [19]). The entities

Algorithm 6: Király's Clustering

```
   Input: Similarity Graph $SE = (V_1, V_2, E)$, similarity threshold
          $t$
   Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$
 1 $C \leftarrow \emptyset$
 2 $freeM \leftarrow \emptyset$          // an initially empty linked list
 3 foreach $v_i \in V_1$ do // $V_1$ corresponds to men in [19]
 4     $Q1_i \leftarrow \emptyset$            // $v_i$'s edges in desc. sim > $t$
 5     $lastChance[i] \leftarrow false$
 6     $freeM.addLast(v_i)$        // keeps insertion order
 7 foreach $v_j \in V_2$ do // $V_2$ corresponds to women in [19]
 8     $Q2_j \leftarrow \emptyset$            // $v_j$'s edges in desc. sim > $t$
 9     $fiancé[j] \leftarrow null$
10 foreach $e = (v_i, v_j, sim) \in E$, with $e.sim > t$ do
11     $Q1_i.push(e)$
12     $Q2_j.push(e)$
13 while $freeM \neq \emptyset$ do
14     $v_i \leftarrow freeM.removeFirst()$  // in insertion order
15     if $Q1_i \neq \emptyset$ then
16         $v_j \leftarrow Q1_i.pop()$        // $v_i$'s preference is $v_j$
17         if $fiancé[j] = null$ then // $v_j$ is free
18             $C \leftarrow C \cup \{\{v_i, v_j\}\}$         // match $v_i$ to $v_j$
19         else
20             $v'_i \leftarrow fiancé[j]$     // $v_j$ was engaged to $v'_i$
21             if acceptsProposal($v_j$, $v_i$) then // refer
                to [19]
22                 $C \leftarrow C \setminus \{\{v'_i, v_j\}\}$    // $v'_i$ and $v_j$ break
                     up
23                 $freeM.addLast(v'_i)$          // $v'_i$ is free
                     again
24                 $C \leftarrow C \cup \{\{v_i, v_j\}\}$    // match $v_i$ to $v_j$
25                 $fiancé[j] \leftarrow v_i$   // $v_j$ gets engaged to
                     $v_i$
26     else
27         if $lastChance(v_i) = false$ then
28             $lastChance[i] \leftarrow true$    // $2^{nd}$ chance for
                 $v_i$
29             $Q1_i \leftarrow recoverInitialQueue(v_i)$
30             $freeM.addLast(v_i)$
31 return $C$
```

Algorithm 7: Unique Mapping Clustering

```
   Input: Similarity Graph $SE = (V_1, V_2, E)$, similarity
          threshold $t$
   Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$
 1 $C \leftarrow \emptyset$
 2 $M_1 \leftarrow \emptyset$                        // matched nodes from $V_1$
 3 $M_2 \leftarrow \emptyset$                        // matched nodes from $V_2$
 4 $Q \leftarrow \emptyset$
 5 foreach $e = (v_i, v_j, sim) \in E$ do
 6     if $e.sim > t$ then
 7         $Q.put(e)$  // a PQ of edges in desc. sim > $t$
 8 while $Q \neq \emptyset$ do
 9     $e \leftarrow Q.pop()$     // the entity pair with highest
         sim
10     if $e.v_i \notin M_1$ and $e.v_j \notin M_2$ then // $v_i$, $v_j$ not
         matched
11         $C \leftarrow C \cup \{\{e.v_i, e.v_j\}\}$
12         $M_1 \leftarrow M_1 \cup \{e.v_i\}$
13         $M_2 \leftarrow M_2 \cup \{e.v_j\}$
   // create singular clusters for unmatched nodes $v$
14 foreach $v \in (V_1 \cup V_2) \setminus (M_1 \cup M_2)$ do
15     $C \leftarrow C \cup \{\{v\}\}$
```

Algorithm 5: Exact Clustering

```
   Input: Similarity Graph $SE = (V_1, V_2, E)$, similarity threshold $t$
   Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$
 1 $C \leftarrow \emptyset$
 2 $M_2 \leftarrow \emptyset$                        // matched nodes from $V_2$
 3 foreach $i \in |V_1|$ do
 4     $Q1_i \leftarrow \emptyset$           // initialize a PQ in desc. sim
 5 foreach $j \in |V_2|$ do
 6     $Q2_j \leftarrow \emptyset$           // initialize a PQ in desc. sim
 7 foreach $e = (v_i, v_j, sim) \in E$, with $e.sim > t$ do
 8     $Q1_i.push(e)$
 9     $Q2_j.push(e)$
10 foreach $v_i \in V_1$ do
11     $c_i \leftarrow \{v_i\}$  // create a new cluster containing $v_i$
12     $e \leftarrow Q1_i.pop()$              // the best edge for $v_i$
13     $v_j \leftarrow e.v_j$           // the best match for $v_i$ is $v_j$
14     $e_2 \leftarrow Q2_j.pop()$             // the best edge for $v_j$
15     if $e_2.v_i = v_i$ then // if the best match for $v_j$ is $v_i$
16         $c_i \leftarrow c_i \cup \{v_j\}$          // add $v_j$ to cluster $c_i$
17         $M_2 \leftarrow M_2 \cup \{e.v_j\}$
18     $C \leftarrow C \cup \{c_i\}$
19 foreach $v_j \in V_2 \setminus M_2$ do // for each unmatched $V_2$ node
20     $C \leftarrow C \cup \{\{v_j\}\}$      // add a new singular cluster
21 return $C$
```

of the target dataset accept a proposal under certain conditions (e.g., if it's the first proposal they receive - Line 17), and the partitions and preferences are updated accordingly (Lines 18, 22, 24). Entities from the source dataset get a second chance to make proposals (Lines 5, 27-30) and the algorithm terminates when all entities of the first dataset are in a partition (Line 13), or no more proposal chances are left (Line 27). We omit some of the details (e.g., the rare case of "uncertain man"), due to space restrictions, and refer the reader to [19] for more information (e.g., the acceptance criteria for proposals). Its time complexity is $O(n + m\ logm)$ [19].

Unique Mapping Clustering (UMC). This algorithm prunes all edges with a weight lower than t, sorts the remaining ones in decreasing weight/similarity (Lines 5-7) and iteratively forms a partition for the top-weighted pair (Line 9), as long as none of its entities has already been matched to some other (Line 10). This comes from the *unique mapping constraint* of CCER, i.e., the restriction that each entity from the one input dataset matches at most one entity from the other. Note that the *CLIP Clustering algorithm*, introduced for the multi-source ER problem in [44], is equivalent to UMC when there are only two input datasets (i.e., in the CCER case that we study). Its time complexity is $O(m\ logm)$, required for sorting all edges[].