Bipartite Graph Matching Algorithms for Clean-Clean Entity Resolution: An Empirical Evaluation

George Papadakis[1], Vasilis Efthymiou[2], Emmanouil 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]KU Leuven, Belgium emmanouil.thanos@kuleuven.be
[4]IBM Research, 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 algorithms and guidelines for choosing the best one, depending on the data at hand.

1 INTRODUCTION

Entity Resolution is a challenging, yet well-studied problem in data integration [4, 22]. A common scenario is Clean-Clean ER (CCER) [5], 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 [37], where a new clean source needs to be integrated into the clean reference data, and Knowledge Graph matching and completion [17, 44], 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 [5, 9]. 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 [30]) 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 input 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:

- In Section 3, 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.
- In Section 4, 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.
- In Section 5, 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.
- In Section 6, we assess the relative performance of the matching algorithms with respect to effectiveness and time efficiency.
- We have publicly released the implementation of all algorithms as well as our experimental results.[1]

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 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 CCER pipeline [5] involves the following three steps:

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

(1) *(meta-)blocking* applies one or more indexing methods that generate *candidate matching pairs*, this way reducing the otherwise quadratic search space of matches.
(2) *matching* assigns a similarity score to each candidate pair.
(3) *bipartite graph matching* receives the scored candidate pairs and decides which ones belong to the same cluster.

In this work, we evaluate how different methods for the last step perform, when the previous ones are fixed.

Problem Definition. The task of **Bipartite Graph Matching** receives as input a bipartite *similarity graph* $G = (V_1, V_2, E)$, 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 partitions/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 nodes from A to B and are associated with a weight that reflects the similarity (matching likelihood) of the adjacent nodes. Figures 1(b)–1(d) show three different outputs of CCER, in which nodes within the same oval (cluster) correspond to matching entities.

Related work. There is a rich body of literature on ER [4, 5]. 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 an 1-1 matching constraint [51], the primary focus of prior work, especially the most recent one, has been on the effectiveness of the classification task, mainly by leveraging Machine [21] and Deep Learning [3, 28, 34].

Inspired by the recent progress and success of prior work on improving the efficiency of ER with blocking and filtering [41], 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; as a result, each cluster could contain more than two records. We refer to this variation of ER as *Dirty ER* [5]. Some of the bipartite matching algorithms we use in this paper are adaptations of the graph clustering algorithms used in [18].

More recent clustering methods for Dirty ER were proposed in [10]. After estimating the connected components, *Global Edge Consistency Gain* iteratively switches the label of edges so as to maximize the overall consistency, i.e,. the number of triangles with the same label in all edges. *Maximum Clique Clustering* ignores edge weights and iteratively removes the maximum clique along with its vertices until all nodes have been assigned to an equivalence cluster. This approach is generalized by *Extended Maximum Clique Clustering*, which removes maximal cliques from the similarity graph and enlarges them by adding edges that are incident to a minimum portion of their nodes.

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.

Table 1: Configuration parameters per algorithm.

Algor.	Similarity Threshold t	Other
CNC	✓	×
RSR	✓	×
RCA	✓	×
BAH	✓	1) maximum search steps (10,000) 2) maximum run-time per search step (2 min.)
BMC	✓	node partition used as basis
EXC	✓	×
KRC	✓	×
UMC	✓	×

FAMER [44] 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), our focus on bipartite graphs, which do not support multi-source settings, makes the direct comparison inapplicable. Note, though, that adapting FAMER's top-performing algorithm, i.e., CLIP clustering, to work in a CCER setting yields an algorithm equivalent to Unique Mapping Clustering.

Wang et al. [47] follow a reinforcement learning approach, based on a Q-learning [49] 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 exclude this algorithm from this study, as we consider only learning-free methods, but we plan to further explore it in our future works.

Kriege et al. [23] 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 consider algorithms satisfying the following selection criteria:

(1) They are crafted for bipartite similarity graphs, which apply exclusively to CCER. Algorithms for the types of graphs that correspond to Dirty and Multi-source ER have been examined in [18] and [44], respectively.
(2) Their functionality is learning-free in the sense that they do not learn a pruning model over a set of labelled instances. We only use the ground-truth of real matches to optimize their internal parameter configuration.
(3) Their time complexity is not worse than the brute-force approach of ER, $O(n^2)$, where $n = |V_1 \cup V_2|$ is the number of nodes in the bipartite similarity graph $G = (V_1, V_2, E)$.
(4) Their space complexity is $O(n + m)$, where $m = |E|$ is the number of edges in the given similarity graph.

Due to the third criterion, we exclude the classic Hungarian algorithm, also known as the Kuhn-Munkres algorithm [24], whose time complexity is cubic, $O(n^3)$. For the same reason, we exclude the work of Schwartz et al. [46] on 1-1 bipartite graph matching with minimum cumulative weights, which reduces the problem to a minimum cost flow problem and uses the matching algorithm of Fredman & Tarjan [13] to provide an approximate

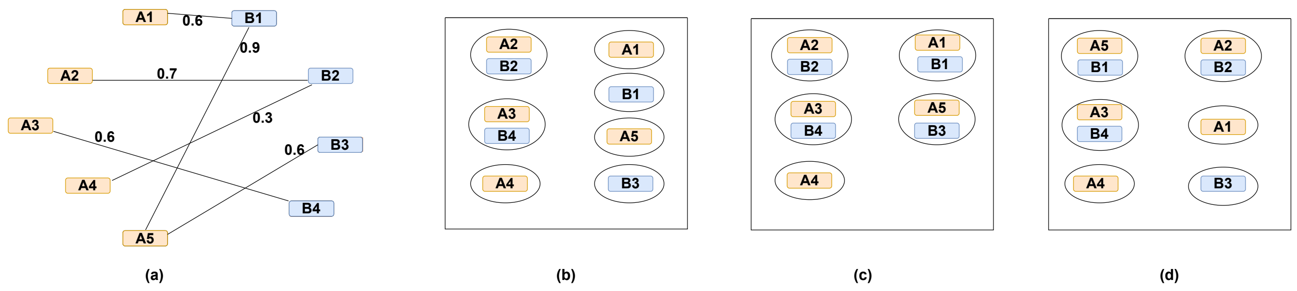


Figure 1: Example of processing a similarity graph: (a) the similarity graph constructed for a pair of clean entity collections (V_1 in orange and V_2 in blue), (b) the resulting clusters after applying CNC, (c) the resulting partitions/clusters assuming that the approximation algorithms RCA or BAH retrieved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

solution in O($n^2 logn$). Note that most of the considered algorithms depend on m, the number of edges in the similarity graph, which is equal to n^2 in the worst case. In practice, though, the value of m is determined by the similarity threshold t, which is used by each algorithm to prune all edges with a lower weight. For reasonable thresholds, $O(n) \leq m \ll O(n^2)$.

Below, we describe the selected algorithms. Table 1 summarizes their configuration parameters. Their implementation (in Java) is publicly available through the JedAI toolkit [40].

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/clusters that contain two entities – one from each entity collection. Using a simple depth-first approach, its time complexity is $O(m)$ [6].

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 entity 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 $O(n\ m)$ [50].

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 $O(|V_1|\ |V_2|)$ [38].

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 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 [32], which solves the stable marriage problem [14], as simplified in BigMat [1]. 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 [32] is the same, in principle, to Unique Mapping Clustering (see below). Note also that an additional configuration parameter is the entity collection that is used as the basis for creating partitions, which 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 $O(m)$ [38].

Exact Clustering (EXC). This algorithm is inspired from the Exact strategy of [32]. 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]. 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 [20]. Intuitively, the entities of V_1 ("men" [20]) propose to the entities from V_2 with an edge weight higher than t ("women" [20]) to form a partition ("get engaged" [20]). 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 [20, 38] for more information (e.g., the acceptance criteria for proposals). Its time complexity is $O(n + m\ logm)$ [20].

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 with at most one entity from the other. Note that the *CLIP Clustering algorithm*, introduced for the multi-source ER problem in [45], is equivalent to UMC in the CCER case that we study. Its time complexity is $O(m\ logm)$ [38].

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 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 [4]:

(1) *learning-free* methods, which produce similarity scores in an unsupervised manner based on the content of the input entities, and
(2) *learning-based* methods, which produce probabilistic similarities based on a training set.

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 [48]. 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 (i) the *representation model*, and (ii) the *similarity measure*.

Representation model. It transforms a textual value into a model that is suitable for applying the selected similarity measure. Depending on the <u>scope</u> 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 <u>*form*</u>, we also distinguish them 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 its 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 of models that have been widely applied to document classification tasks [39]:

(1) an n-gram vector [31], 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.
(2) 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 sparse vector with as many dimensions as all the 3-grams appearing in the entity collection and 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, 'oe_' to 'e_B' and '_Bi', etc. See [38] for more details.

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]. For both approaches, we consider $n \in \{2, 3, 4\}$ for character and $n \in \{1, 2, 3\}$ for token n-grams.

The *semantic representations* treat every text as a sequence of items (words or character n-grams) of arbitrary length and convert 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 to both 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* [33] and *GloVe* [42], 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 [34]. This drawback is addressed by the character-level embeddings: *fastText* vectorizes a token by summing the embeddings of all its character n-grams [2]. For this reason, we exclusively consider the 300-dimensional fastText in the following.

(ii) Transformer-based language models [8] 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 [3, 28]. They do suffer from out-of-vocabulary tokens, but to the best of our knowledge, there is no established character-level language model that could address this issue, as fastText does for pre-trained embeddings. Among them, we exclusively consider the 768-dimensional ALBERT, due to its higher efficiency [26].

Similarity measure. It is a function that receives as input two representation models and produces a score 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, thus, the higher is the matching likelihood.

For each type of representation models, we considered a large variety of established similarity measures. The following are combined with the character-level schema-based representation models: Damerau-Levenshtein, Levenshtein and q-grams distance, Jaro Similarity, Needleman Wunch, Longest Common Subsequence and Longest Common Subsequence. To the token-level, schema-based models we apply: Cosine, Dice and (Generalized) Jaccard similarity as well as Monge-Elkan, Overlap Coefficient, Block and Euclidean distance. The schema-agnostic n-gram vectors are coupled with Arcs and Jaccard similarity as well as with Cosine and Generalized Jaccard similarity with TF or TF-IDF weights. For the n-gram graphs, we consider Containment, (Normalized) Value and Overall similarity. Finally, the semantic similarity models are combined with Cosine, Euclidean and World Mover's similarity. We formally define these measures in the Appendix of [38].

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 toolkit[3]). 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 2, where $|V_x|$ 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, which was introduced in OAEI 2010[6], contains data about restaurants. D_2 matches products from the online retailers Abt.com and Buy.com [22]. D_3 interlinks products from Amazon and the Google Base data API (Google Pr.) [22]. D_4 contains data about publications from DBLP and ACM [22]. $D_5 - D_7$ contain data about television shows from TheTVDB.com (TVDB) and movies from IMDb and themoviedb.org (TMDb) [36]. D_8 contains data about products from Walmart and Amazon [34]. D_9 contains data about scientific publications from DBLP and Google Scholar [22]. D_{10} matches movies from IMDb and DBpedia [40] (note that D_{10} contains a different snapshot of IMDb movies than D_5 and D_6). All datasets are publicly available through the JedAI data repository.[7]

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 "name" and "phone" for D_1, "name" for D_2, "title" for D_3, "title" and "authors" for D_4, "modelno" and "title" for D_5, "title" and "authors" for D_6, "name" and "title" for D_7, "title" and "name" for D_8, "title" and "abstract" for D_9, and "title" for D_{10}.

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* ($F1$) is the harmonic mean of Precision and Recall.

All are defined in $[0, 1]$. Higher values show 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 2. We actually apply all combinations of representation models and similarity measures, 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

[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 2: Technical characteristics of the real datasets for Clean-Clean ER in increasing computational cost.

	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
$\|V_1\|$	339	1,076	1,354	2,616	5,118	5,118	6,056	2,554	2,516	27,615
$\|V_2\|$	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	$1.14{\cdot}10^5$	$1.98{\cdot}10^5$	$8.2{\cdot}10^5$
$\|A_1\|$	7	3	4	4	13	13	30	6	4	4
$\|A_2\|$	7	3	4	4	30	9	9	6	4	7
$\|\bar{p}_1\|$	3.33	2.39	3.92	4.00	4.16	4.16	3.92	5.54	4.00	5.63
$\|\bar{p}_2\|$	3.33	2.14	3.00	3.99	3.92	2.68	2.68	5.18	3.24	35.20
$\|D(V_1 \cap V_2)\|$	89	1,076	1,104	2,224	1,968	1,072	1,095	853	2,308	22,863
$\|\|V_1 \times V_2\|\|$	$7.65{\cdot}10^5$	$1.16{\cdot}10^6$	$4.11{\cdot}10^6$	$6.00{\cdot}10^6$	$3.10{\cdot}10^7$	$4.00{\cdot}10^7$	$4.73{\cdot}10^7$	$5.64{\cdot}10^7$	$1.54{\cdot}10^8$	$6.40{\cdot}10^8$

Table 3: The number of similarity graphs $|G|$ as well as their size, in terms of the average number of edges $|\bar{E}|$, per dataset. In parenthesis, the ratio of $|\bar{E}|$ to $||V_1 \times V_2||$ in Table 2.

	Syntactic Similarities				Semantic Similarities			
	Schema-based		Schema-ag.		Schema-based		Schema-ag.	
	|G|	$\|\bar{E}\|{\cdot}10^6$	|G|	$\|\bar{E}\|{\cdot}10^6$	|G|	$\|\bar{E}\|{\cdot}10^6$	|G|	$\|\bar{E}\|{\cdot}10^6$
D_1	20	0.16 (21.2%)	46	0.72 (93.5%)	8	0.26 (34.3%)	2	0.76 (100%)
D_2	12	1.05 (90.5%)	47	0.64 (55.1%)	2	1.16 (100%)	2	1.16 (100%)
D_3	14	2.89 (70.5%)	53	2.65 (64.5%)	2	4.11 (100%)	2	4.11 (100%)
D_4	27	4.49 (74.8%)	24	3.84 (64.0%)	12	5.99 (99.8%)	4	6.00 (100%)
D_5	24	5.81 (18.7%)	48	11.92 (38.5%)	12	8.22 (26.7%)	2	30.64 (98.8%)
D_6	25	8.39 (21.0%)	45	10.99 (27.5%)	12	12.31 (30.8%)	2	39.81 (99.6%)
D_7	26	2.80 (05.9%)	42	12.21 (25.8%)	12	36.44 (07.8%)	5	46.99 (99.3%)
D_8	26	28.10 (49.8%)	47	37.31 (66.2%)	2	37.70 (67.0%)	-	-
D_9	20	119.18 (77.2%)	46	77.56 (50.2%)	6	154.26 (100%)	2	154.36 (100%)
D_{10}	13	250.73 (39.2%)	43	317.17 (49.5%)	2	378.51 (59.0%)	-	-
Σ	207	-	441	-	70	-	21	-

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.[8] The largest threshold that achieves the highest $F1$ is selected as the optimal one, determining the performance of the algorithm for the particular input.

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 functions 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 3. Overall, there are 739 different similarity graphs, most of which rely on syntactic similarity functions and the schema-agnostic settings, in particular. The semantic similarities assign relatively high similarity scores to most pairs of 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.

[8]Preliminary experiments showed that there is no significant difference in the experimental results when using a smaller step size like 0.01. Thus, we set it to 0.05 to reduce the effort for the experiments, due to the large number of algorithms, inputs and datasets they involve.

Table 4: Macro-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

6 EXPERIMENTAL ANALYSIS

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 macro-average performance (μ) of all algorithms across all input similarity graphs, which is reported in Table 4. We observe that all algorithms emphasize 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

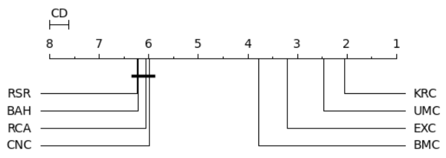


Figure 2: Nemenyi diagram based on F-Measure.

former achieving the third highest F1, while RSR and RCA lie closer to CNC, with RSR exhibiting the third lowest F1.

To assess the statistical significance of these patterns, we perform an analysis [19] based on their F-Measure over the 739 paired samples. In more detail, we first perform the non-parametric Friedman test [7] and reject the null hypothesis (with $\alpha = 0.05$) that the differences between the evaluated methods are statistically insignificant. Then, we perform a post-hoc Nemenyi test [35] to identify the critical distance (CD = 0.37) between the methods. The Nemenyi diagram based on F-Measure, which appears in Figure 2, shows that there are no significant differences among the methods with the worst ranks with respect to F-Measure (CNC, RCA, BAH, and RSR). All other differences are significant, with KRC, UMC, EXC, and BMC, landing in the top ranking positions (in that order).

We have also performed the same analysis for Precision and for Recall [38] (the same CD of 0.37 applies). The Precision-based ranking (reporting Mean Rank, MR) of the methods is: CNC (MR=1.28), EXC (MR=2.5), KRC (MR=3.7), UMC (MR=4.81), BMC (MR=5.3), RSR (MR=5.66), BAH (MR=6.12), and RCA (MR=6.64); hence, there is no significant difference between BMC and RSR, while all other differences are significant. The Recall-based ranking of the methods is: UMC (MR=1.77), KRC (MR=2.44), BMC (MR=3.15), EXC (MR=4.34), RCA (MR=5.46), RSR (MR=5.92), BAH (MR=5.93), and CNC (MR=7), i.e., there is no significant difference between RSR and BAH, while all other differences are significant.

An interesting observation drawn from these patterns 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, BMC works best when choosing the smallest entity collection as the basis for creating clusters.

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 4 - 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 4 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: average precision increases for all algorithms except CNC and RSR, while average 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 4. 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 4. 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 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 $|V_2|$:

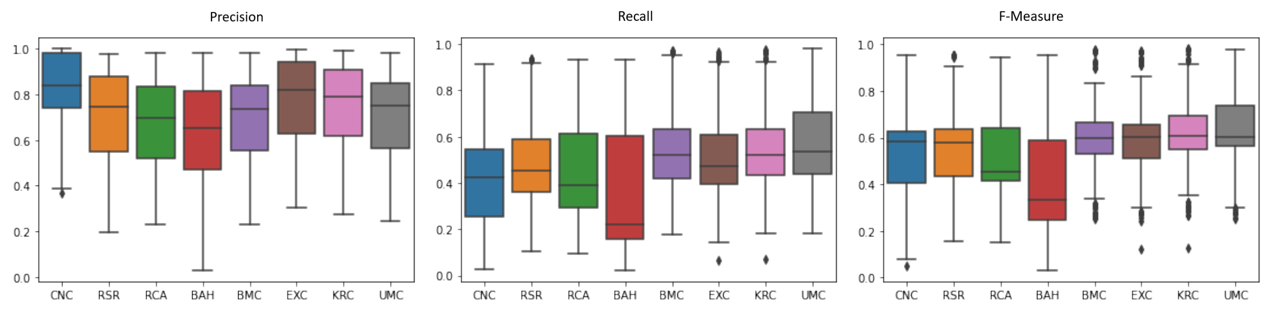
(1) *Balanced* (BLC) are the entity collections where the vast majority of entities in V_i are matched with an entity in V_j (i=1 ∧ j=2 or i=2 ∧ j=1). This category includes all similarity graphs generated from D_2, D_4 and D_{10}.
(2) *One-sided* (OSD) are the entity collections, where only the vast 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.
(3) *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.

We apply this categorization to the four main types of similarity graphs defined in Section 4 and for each subcategory, we consider three new effectiveness measures:

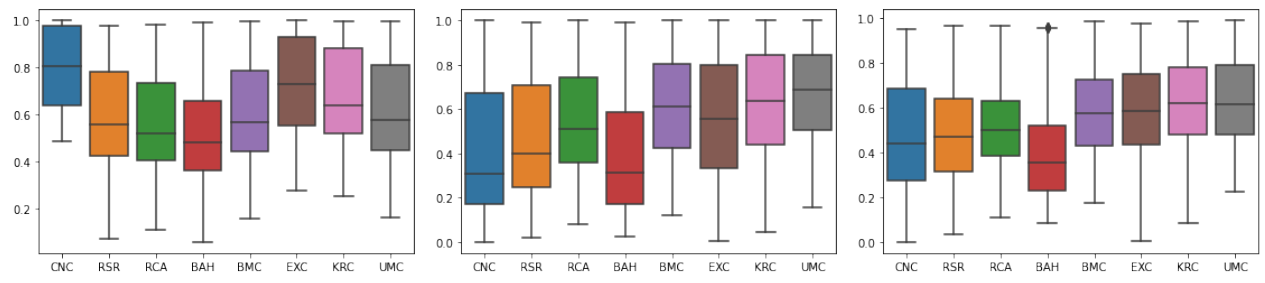
(1) $\#Top1$ denotes the number of times an algorithm achieves the maximum F1 for a particular category of similarity graphs.
(2) $\#Top2$ denotes the number of times an algorithm scores the second highest F1 for a particular category of similarity graphs.
(3) Δ (%) stands for the average difference (expressed as a percentage) between the highest and the second highest F1 across all similarity graphs of the same category.

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.

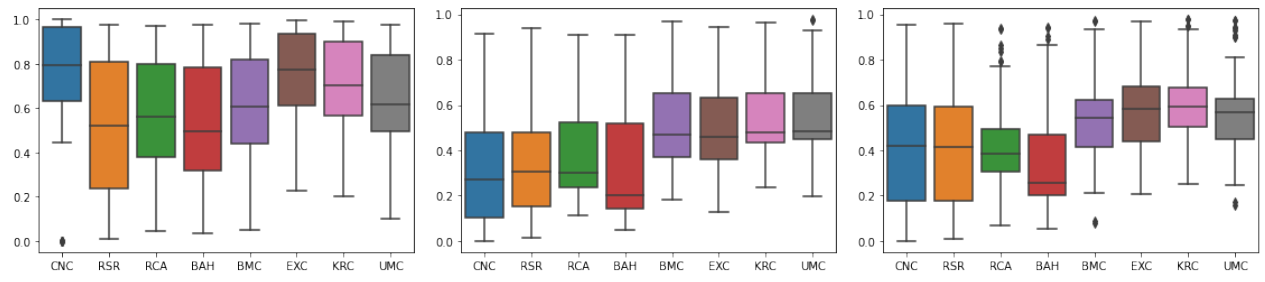
The results for these measures are reported in Table 5. For schema-based syntactic weights, there is a strong competition between KRC and UMC for the highest effectiveness. Both algorithms achieve the maximum F1 for the same number of similarity graphs in the case of balanced and one-sided entity collections.



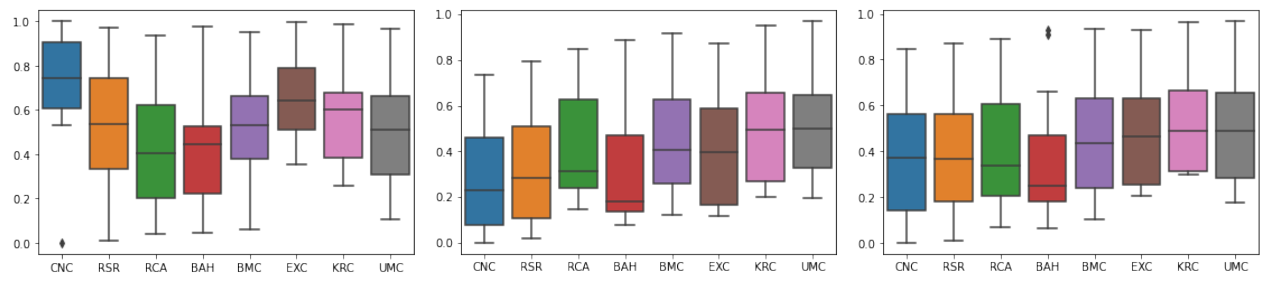
(a) Schema-based Syntactic Weights



(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.

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 4, 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 the top F1 2.5 times more often, 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

Table 5: The number of times each algorithm achieves the highest and second highest F1 for a particular similarity graph, $\#Top1$ and $\#Top2$, respectively, 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, respectively, over schema-based syntactic weights, 17 and 11, respectively, over schema-agnostic syntactic weights, 9 and 2, respectively, 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**

competing algorithms are KRC and EXC, as they share the highest $\#Top1$. Yet, the former achieves three times higher Δ and slightly higher $\#Top2$. Surprisingly, 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 the schema-agnostic weights and in $\#Top2$ for the 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, movies for D_5-D_8 and D_{10}). Yet, no clear patterns emerged in these cases.

Time Efficiency. The (relative) run-time of the evaluated algorithms is a crucial aspect for EER, due to the very large similarity graphs, which comprise thousands of entities/nodes and (hundreds of) millions of edges/entity pairs, as reported in Table 3. Below, we study this aspect along with the scalability of the considered algorithms over the 739 different similarity graphs. More specifically, we examine the following questions:

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): Which algorithms offer the best trade-off between F-Measure and run-time?

Table 6 reports the average run-times over 10 executions of the evaluated algorithms per dataset and type of edge weights.

Regarding QT(1), we observe that all algorithms are quite fast, as they are all able to process even the largest similarity graphs (i.e., those of D_9 and D_{10}) within minutes or even seconds. CNC is the fastest one 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 efficient, but as expected, it is usually 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 2 or even 3 orders of magnitude longer run-times. The reason is the large number (10,000) of search steps we allow per dataset. 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 time-out. 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. Among the

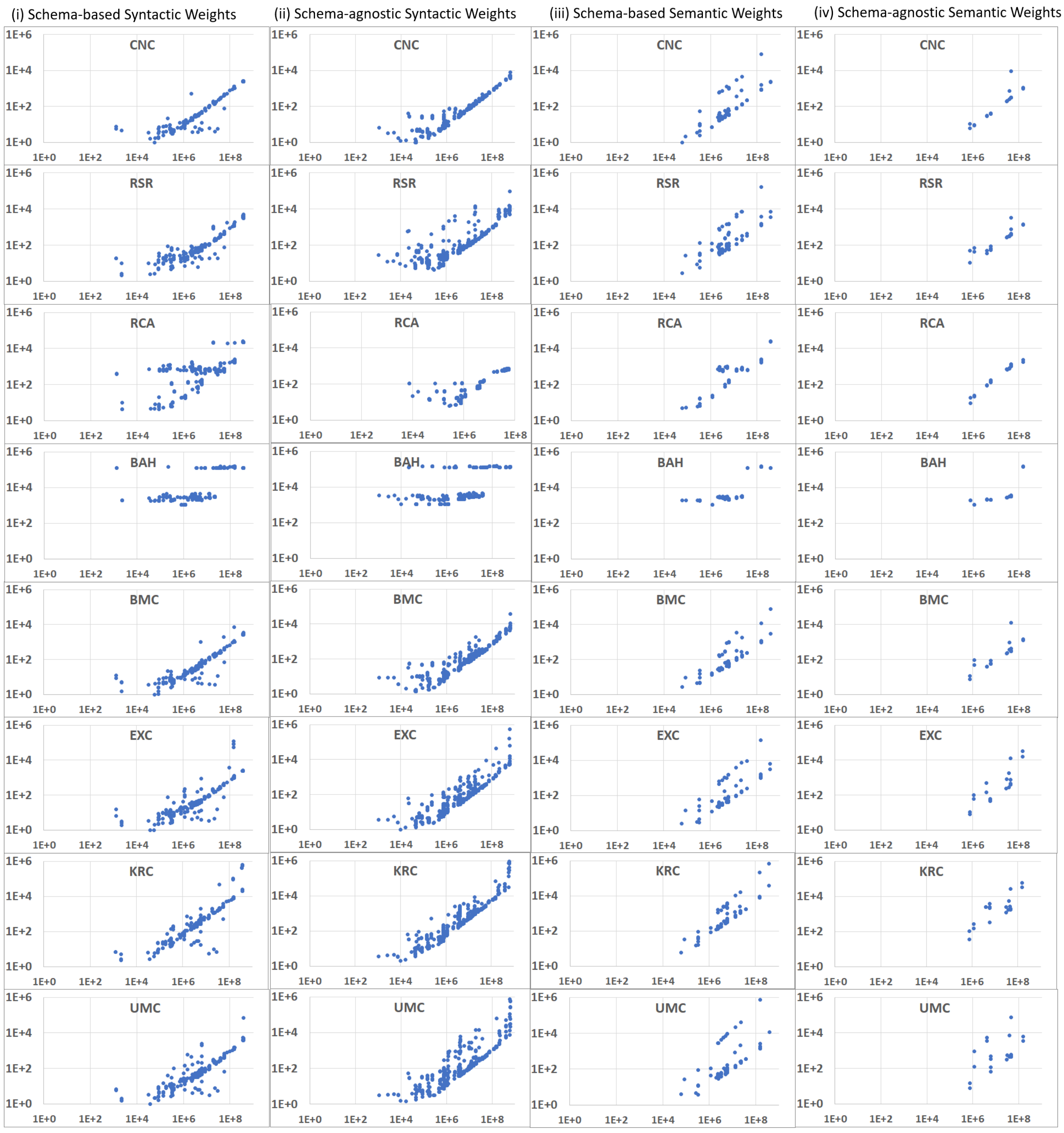


Figure 4: Scalability analysis of all algorithms over all similarity graphs with (i) schema-based syntactic, (ii) schema-agnostic syntactic, (iii) schema-based semantic and (iv) schema-agnostic semantic edge weights. The horizontal axis corresponds to the number of edges in the similarity graphs and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

most effective algorithms, EXC is significantly faster than UMC, which is significantly faster than KRC.

Regarding QT(2), there are two main factors that affect the reported run-times:

(1) the time complexity of the algorithms, and
(2) the similarity threshold used for pruning the search space.

Regarding the first factor, we observe that the run-times in Table 6 verify the time complexities described in Section 3. With $O(m)$, CNC and BMC are the fastest ones, followed by RSR and EXC with $O(n\ m)$, 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 determined by the number of search steps and the run-time limit.

Equally important is the effect of the similarity thresholds: the higher their optimal value (i.e., the one maximizing F1) is, the fewer edges are retained in the similarity graph and the faster is its processing. The optimal similarity threshold depends on the type of edge weights and the size of the similarity graph, as we explain in the threshold analysis in [38]. This means that the relative time efficiency of algorithms with the same theoretical complexity should be attributed to their different similarity threshold. For example, the average optimal thresholds for CNC and BMC over all schema-based syntactic weights are 0.755 and

Table 6: Mean run-time per algorithm, dataset and type of input. Milliseconds are reported, except for BAH, D_{10} and cases followed by s, which are measured in seconds.

	CNC	RSR	RCA	BAH (sec)	BMC	EXC	KRC	UMC
D1	3±1	8±5	8±4	1.9±.0	4±5	4±3	14±9	4±3
D2	8±1	15±4	20±2	1.1±.0	13±5	59±73	66±19	39±50
D3	21±13	36±16	57±15	2.2±.1	24±16	34±37	200±91	61±64
D4	32±18	57±23	136±20	2.0±.0	43±25	69±164	384±381	254±601
D5	44±41	56±38	662±66	3.6±.8	49±49	43±35	286±254	53±45
D6	52±60	86±79	706±97	3.0±.2	63±75	58±64	422±508	79±93
D7	41±95	43±17	980±251	3.1±.3	28±16	27±13	204±138	50±81
D8	196±168	209±167	557±136	123.5±.6	282±402	182±153	1.2s±1.0s	213±194
D9	946±421	994±408	1.8s±.3s	147.5±2.7	1.1s±1.4s	18s±36s	17s±29s	1.1s±.5s
D10	1.6±1.1	2.8±1.4	21.9±1.2	127±2	1.9±1.2	1.6±1.1	164±243	7.9±18.4

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

	CNC	RSR	RCA	BAH (sec)	BMC	EXC	KRC	UMC
D1	11±8	19±27	10±2	1.9±.1	9±5	22±34	52±0.2	25±47
D2	6±3	18±8	18±4	1.1±.0	10±10	14±16	51±40	74±166
D3	21±13	39±21	57±13	2.3±.4	34±48	81±177	582±686	599±1.2s
D4	31±19	56±22	130±17	2.2±.3	40±26	44±74	334±705	125±356
D5	93±58	206±393	616±55	2.9±.4	172±201	203±404	858±821	672±2.0s
D6	86±83	124±84	676±70	3.3±.4	99±92	151±216	635±578	116±107
D7	102±109	129±127	912±91	3.5±.4	104±96	112±168	637±713	232±716
D8	280±121	319±132	581±91	122.8±.4	245±103	443±1,264	1.8s±747	360±155
D9	611±492	728±515	1.5s±340	144.8±2	627±499	1.7s±6.5s	5.3s±9.7s	2.1s±8.8s
D10	2.5±2.1s	8.4±14.3	24.6±2.8	128.4±2.1	4.1±6.0	21.1±86.4	205±354	157±600

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

	CNC	RSR	RCA	BAH (sec)	BMC	EXC	KRC	UMC
D1	11±18	32±43	9±5	1.9±.0	9±7	16±19	33±26	20±29
D2	7±0	91±51	23±3	1.0±.0	15±2	30±26	118±42	79±48
D3	28±4	136±124	88±16	2.2±.0	282±11	602±612	2.4s±.2	5.6s±.0s
D4	210±388	392±514	152±14	2.1±.1	217±366	295±569	1.5s±1.4s	1.6±3.6s
D5	384±869	883±1.7s	629±49	2.8±.2	373±947	445±1s	1.4s±2.8s	2.3s±6.3
D6	550±1.3s	1.4s±2.8s	730±70	3.0±.2	250±482	754±2s	2.1s±4.4s	4s±12s
D7	181±372	264±415	917±77	3.0±.1	124±247	168±301	547±741	882±2.2s
D8	216±0	381±78	654±31	123.6±.3	245±2	4.8s±6.4s	1.8s±.0s	364±10
D9	1s±317	1.8s±1.1s	2.2s±.3s	147.8±2.7	1.1s±.1s	1.2s±.2s	8.2s±.6s	1.7s±.5s
D10	2.3±.1	5.2±2.4	24.9±1.3	128.8±1.2	39.4±51.6	4.8±2.4	365±463	137±191

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

	CNC	RSR	RCA	BAH (sec)	BMC	EXC	KRC	UMC
D1	8±3	31±28	14±6	1.9±.1	9±3	9±2	72±50	11±5
D2	9±0	58±22	23±1	1.1±.0	70±30	80±30	215±89	521±561
D3	30±2	43±9	92±3	2.1±.0	39±0	320±255	2.5s±.0s	4.5s±1.5s
D4	40±1	65±13	147±16	2.0±.0	65±14	54±7	2.2s±1.4s	250±194
D5	199±5	280±16	700±22	2.8±.0	230±19	517±401	1.8s±.8s	441±180
D6	486±336	372±12	816±54	3.2±.2	660±412	1.1s±1.1s	3.6s±2.5s	3.9s±4.9s
D7	2.1s±4.1s	1.1s±1.3s	1.2s±.1s	3.3±.2	2.8s±5.6	3.0s±5.6s	7.0s±10.9s	15s±33s
D8	-	-	-	-	-	-	-	-
D9	1.1s±.1	1.3s±.1s	2.0s±.3s	148.6±3.3	1.4s±.1	24s±11s	46s±18s	4.8s±1.8s
D10	-	-	-	-	-	-	-	-

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

0.669, respectively, while over schema-agnostic syntactic weights they are 0.409 and 0.327, respectively. These large differences account for the significantly lower run-time of CNC in almost all datasets for both cases. The larger the difference in the similarity threshold, the larger is the difference in the run-times.

Note that the similarity threshold typically accounts for the relative run-times that conflict with the relative time complexities, too: in case an algorithm runs faster than another one with lower time complexity, this is typically caused by the higher similarity threshold it employs. For example, KRC exhibits a lower average run-time than EXC over D_9 for schema-based syntactic weights, because their mean optimal similarity thresholds amount to 0.550 and 0.490, respectively. Similarly, UMC runs much faster than EXC over D_8 with schema-agnostic syntactic weights, because their mean optimal similarity thresholds amount to 0.427 and 0.387, respectively.

Finally, the similarity threshold accounts for the relative run-times between the same algorithm over different types of edge weights. For example, EXC is 13 times slower over the schema-agnostic syntactic weights of D_{10} than their schema-based counterparts, even though the former involve just 25% more edges than the latter, as reported in Table 3. This significant difference should be attributed to the large deviation in the mean optimal thresholds: 0.153 for the former weights and 0.535 for the latter ones. The same applies to UMC, whose average run-time increases by 20 times when comparing the schema-based with the schema-agnostic weights, because its average optimal threshold drops from 0.481 to 0.110.

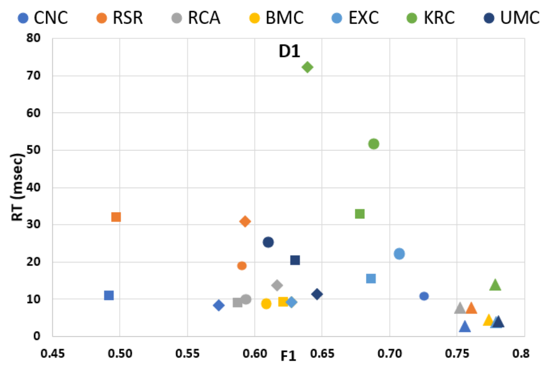


Figure 5: F1 vs run-time diagram for all algorithms, except for BAH, over D_1.

To answer QT(3), Figure 4 presents the scalability analysis of every algorithm over all similarity graphs for each type of edge weights. In each diagram, every point corresponds to the run-time of a different similarity graph. We observe that for all algorithms, the run-time increases linearly with the size of the similarity graphs: as the number of edges increases by four orders of magnitude, from 10^4 to 10^8, the run-times increase to a similar extent in most cases. For all algorithms, though, there are outlier points that deviate from the "central" curve. The larger the number of outliers is, the less robust is the time efficiency of the corresponding algorithm, due to its sensitivity to the size of the graph and the similarity threshold. In this respect, the least robust algorithms are RSR over schema-based syntactic weights along with UMC and EXC over schema-agnostic syntactic weights. These patterns seem to apply to the semantic weights, too, despite the limited number of the similarity graphs, especially in the case of schema-agnostic settings.

Note that there are two exceptions to these patterns: RCA and BAH. The diagrams of the former algorithm seem to involve a much lower number of points, as its time complexity depends exclusively on the number of entities in the input entity collections, i.e., $O(|V_1|\ |V_2|)$. As a result, different similarity graphs from the same dataset yield similar run-times that coincide in the diagrams of Figure 4. Regarding BAH, it exhibits a step-resembling scalability graph, because its processing terminates after a pre-defined timeout or a fixed number of iterations (whichever comes first), independently of the size of the similarity graph.

To answer QT(4), Figure 5 depicts the trade-off between macro-averaged F1 and run-time (RT) per algorithm and type of edge weights in D_1. Note that every type corresponds to a different shape: circle stands for the schema-agnostic syntactic weights, triangle for the schema-based syntactic ones, rhombus for the schema-agnostic semantic ones and rectangle for the schema-based semantic ones. We observe that the schema-based syntactic similarity graphs dominate the other types of input, as they exhibit very high F1 in combination with the lowest run-times. The former should be attributed to the relatively clean values of names and phones for the duplicate entities and the latter to the lack of attribute values for most non-matching entities. As a result, the average size of these graphs is significantly lower than the other types of graphs, especially the schema-agnostic ones, as shown in Table 3. Among the schema-based syntactic inputs, the differences in F1 are lower than 4%, with UMC achieving the best trade-off between the two measures (average F1=0.781 for average run-time=4 msec). This combination practically dominates all others. Only CNC is significantly faster (RT=3 msec), but its F1

Table 7: Comparison to state-of-the-art matching methods.

	ZeroER	DITTO	UMC (schema-agnostic TF-IDF weights, cosine sim.)
D_2	0.52	0.89	0.95 (character bi-grams, $t = 0.35$)
D_3	0.48	0.76	0.60 (token bi-grams, $t = 0.05$)
D_4	0.96	0.99	0.99 (token uni-grams, $t = 0.40$)
D_5	0.86	0.96	0.94 (character four-grams, $t = 0.35$)

(0.756) is also significantly lower. For the patterns pertaining to rest of the datasets, please refer to the extended version in [38].

Comparison with the best matching methods. We now compare the performance achieved by bipartite graph matching algorithms with the recent state-of-the-art matching methods: ZeroER [52], which leverages unsupervised learning, and DITTO [27], which is based on deep learning. We consider the four common datasets, namely D_2-D_5 (D_1 is a larger and noisier version of FZ in [27, 52] and, thus, not directly comparable). Table 7 reports the relative performance in terms of maximum F1 for ZeroER and DITTO, as it was reported reported in [52] and [27], respectively.

Bipartite matching is represented by UMC in combination with cosine similarity over schema-agnostic vector models with TF-IDF weights; the best representation model and the corresponding similarity threshold depend on the dataset. These settings do not necessarily correspond to the highest F1 across all algorithms and similarity graphs we have considered, but demonstrate the capabilities of bipartite matching when varying just two configuration parameters. The results appear in Table 7.

Compared to ZeroER, we observe that UMC consistently achieves higher performance: its F1 is higher by 3%, 9%, 25% and 83% over D_4, D_5, D_3 and D_2, respectively. Compared to DITTO, UMC achieves identical performance over D_4 and 2% lower F1 over D_5. The difference is much larger over D_3, where UMC underperforms by 21%, due to the contextual evidence captured by the training set and the RoBERTa language model [29] that lies at the core of DITTO. Note, though, that the difference is reduced to 12%, when considering the best schema-based configuration of UMC (the Overlap Coefficient of the tokens of the "Title" attribute with a similarity threshold of 0.3). For D_2, UMC outperforms DITTO by 7%.

Overall, we can conclude that bipartite graph matching underperforms the best (deep learning-based) performance in the literature in just one out of four benchmark datasets.

7 CONCLUSIONS

We draw the following important patterns from our experiments:

(1) The best performing algorithm for a particular similarity graph mainly depends on the type of edge weights and the portion of duplicates with respect to the total number of nodes/entities.
(2) CNC constitutes the fastest algorithm, due to its simplicity and the high similarity thresholds it employs, achieving the highest precision at the cost of low recall. It frequently outperforms all other algorithms with respect to F1 in the case of scarce entity collections with syntactic weights, especially the schema-agnostic ones.
(3) RSR is a fast algorithm that rarely achieves high effectiveness, in the case of scarce entity collections.
(4) RCA is efficient, but never excels in effectiveness.
(5) BAH constitutes a slow, stochastic approach that is capable of the best and the worst. It frequently achieves, by far, the highest F1 over balanced entity collections (and rarely over scarce ones), but in most cases, it yields the lowest scores with respect to all effectiveness measures.
(6) BMC is the second fastest algorithm that tries to balance precision and recall, being particularly effective in the case of scarce entity collections, especially in combination with syntactic weights.
(7) EXC improves BMC by boosting precision at the cost of lower recall and higher run-time. It consistently achieves (close to) the maximum F1 over scarce and one-sided entity collections, losing only to KRC and (rarely) to UMC. Given, though, that it outperforms both algorithms to a significant extent with respect to run-time, it constitutes the best choice for applications requiring both high effectiveness and high efficiency/scalability.
(8) KRC achieves very high or the highest effectiveness in most cases, especially over one-sided and scarce entity collections. This comes, though, at the cost of higher (yet stable) run-times than its top-performing counterparts.
(9) UMC is the best choice for balanced entity collections, especially when coupled with syntactic weights, exhibiting a much more robust performance than BAH. It achieves very high (and frequently the highest) effectiveness in the rest of the cases, too. Its run-time, though, is rather unstable, depending largely on the optimal similarity threshold.

Following the approach in [43], an interesting avenue for future work is a supervised learning system that incorporates various bipartite graph matching algorithms in a unified framework and effectively learns the right parameters for a given data set.

ACKNOWLEDGMENTS

This project has received funding from the Hellenic Foundation for Research and Innovation (HFRI) and the General Secretariat for Research and Technology (GSRT), under grant agreement No 969.

REFERENCES

[1] 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.
[2] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomás Mikolov. 2017. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguistics* 5 (2017), 135–146.
[3] Ursin Brunner and Kurt Stockinger. 2020. Entity Matching with Transformer Architectures - A Step Forward in Data Integration. In *EDBT*. 463–473.
[4] Peter Christen. 2012. *Data Matching - Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection.* Springer.
[5] 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.
[6] Sanjoy Dasgupta, Christos H. Papadimitriou, and Umesh V. Vazirani. 2008. *Algorithms.* McGraw-Hill.
[7] Janez Demsar. 2006. Statistical Comparisons of Classifiers over Multiple Data Sets. *J. Mach. Learn. Res.* 7 (2006), 1–30.
[8] 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.
[9] Xin Luna Dong and Divesh Srivastava. 2015. *Big Data Integration.* Morgan & Claypool Publishers.
[10] Uwe Draisbach, Peter Christen, and Felix Naumann. 2020. Transforming Pairwise Duplicates to Entity Clusters for High-quality Duplicate Detection. *ACM J. Data Inf. Qual.* 12, 1 (2020), 3:1–3:30.
[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] Steffen Herbold. 2020. Autorank: A Python package for automated ranking of classifiers. *Journal of Open Source Software* 5, 48 (2020), 2173.

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

[21] 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.

[22] 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.

[23] 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.

[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, AnHai Doan, and Wang-Chiew Tan. 2020. Deep Entity Matching with Pre-Trained Language Models. *Proc. VLDB Endow.* 14, 1 (2020), 50–60.

[28] 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.

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. *CoRR* abs/1907.11692 (2019).

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

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

[32] 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.

[33] 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.

[34] 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.

[35] P. Nemenyi. 1963. *Distribution-free Multiple Comparisons.* Princeton University.

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

[37] 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

[38] George Papadakis, Vasilis Efthymiou, Emanouil Thanos, and Oktie Hassanzadeh. 2021. Bipartite Graph Matching Algorithms for Entity Resolution: An Empirical Evaluation. arXiv:2112.14030 https://arxiv.org/abs/2112.14030

[39] 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.

[40] 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.

[41] 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

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

[43] 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

[44] 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.

[45] 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.

[46] 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.

[47] 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.

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

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

[50] 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.

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

[52] Renzhi Wu, Sanya Chaba, Saurabh Sawlani, Xu Chu, and Saravanan Thirumuruganathan. 2020. ZeroER: Entity Resolution using Zero Labeled Examples. In *SIGMOD.* 1149–1164.