The VLDB Journal (2023) 32:1369–1400
https://doi.org/10.1007/s00778-023-00791-3

REGULAR PAPER

An analysis of one-to-one matching algorithms for entity resolution

George Papadakis[1] · Vasilis Efthymiou[2] · Emmanouil Thanos[3] · Oktie Hassanzadeh[4] · Peter Christen[5]

Received: 29 April 2022 / Revised: 3 January 2023 / Accepted: 11 March 2023 / Published online: 18 April 2023

Abstract
Entity resolution (ER) is the task of finding records that refer to the same real-world entities. A common scenario, which we refer to as Clean-Clean ER, is to resolve records across two clean sources (i.e., they are duplicate-free and contain one record per entity). Matching algorithms for Clean-Clean ER yield bipartite graphs, which are further processed by clustering algorithms to produce the end result. In this paper, we perform an extensive empirical evaluation of eight bipartite graph matching algorithms that take as input a bipartite similarity graph and provide as output a set of matched records. 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 these algorithms with respect to accuracy and time efficiency over ten established real-world data sets, from which we generated over 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.

Keywords Bipartite graphs · Graph matching · Clustering · Experimental evaluation · Record linkage

1 Introduction

Entity resolution (ER) is a challenging, yet well-studied problem in data integration [7, 29]. A common scenario is Clean-Clean ER (CCER) [9], where the two data sets to be integrated are both clean (i.e., free of duplicate records), or are cleaned using single-source ER frameworks. Example applications include master data management [43], where a new clean source needs to be integrated into the clean reference data, and knowledge graph matching and completion [23, 55], 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 records across two data sets and provide as output pairs of records along with a confidence or similarity score [9, 13]. This output can then be used to decide which pairs should be matched. The simplest approach is specifying as duplicates all 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 compared records.

Most importantly, for CCER, this approach does not guarantee that each record can be matched with at most one other record. If we view the output as a bipartite *similarity graph*, where nodes are records and edge weights are the similarity scores calculated between records, what we need is finding a *matching* (or independent edge set [36]) so that each record from one data set is matched to at most one record in the other.

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

✉ George Papadakis
gpapadis@di.uoa.gr

Vasilis Efthymiou
vefthym@ics.forth.gr

Emmanouil Thanos
emmanouil.thanos@kuleuven.be

Oktie Hassanzadeh
hassanzadeh@us.ibm.com

Peter Christen
peter.christen@anu.edu.au

1 National and Kapodistrian University of Athens, Athens, Greece

2 Foundation for Research and Technology - Hellas, Heraklion, Greece

3 KU Leuven, Leuven, Belgium

4 IBM Research,Yorktown Heights, NY, USA

5 Australian National University, Canberra, Australia

to primarily focus on bipartite graph matching algorithms, where we examine the relative performance of such algorithms on 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 and which offers the best balance between effectiveness and time efficiency?*
- *Which algorithm is the most robust with respect to configuration parameters and data characteristics?*
- *How well do bipartite graph matching algorithms scale with regard to the size of similarity graphs?*
- *Which characteristics of the input graphs determine the absolute and the relative performance of these algorithms?*

By answering these questions, we intend to facilitate the selection of the best algorithm for a given pair of data sets. In summary, we make the following contributions:

1. In Sect. 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 for CCER in prior work.
2. In Sect. 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.
3. We perform an extensive experimental analysis that involves 739 different similarity graphs from ten established real-world CCER data sets, the sizes of which range from several thousands to hundreds of millions of edges, as described in Sect. 5.
4. In Sect. 6, we assess and discuss the relative performance of the eight matching algorithms with respect to effectiveness and time efficiency.
5. We have publicly released the implementations of all algorithms, our data as well as our experimental results. See https://github.com/gpapadis/BipartiteGraphMatchingAlgorithms for details.

2 Preliminaries

We assume that a *record* is the description of a real-world entity, provided as a set of attribute-value pairs in some *data set* V. The problem of ER is to identify pairs or groups of records (called *matches* or *duplicates*) that correspond to the same entity and place them into a *cluster* c. In other words, the output of ER, ideally, is a set of clusters C, each containing all matching records 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 records coming from two clean (i.e., duplicate-free) data sets V_1 and V_2. This means that the resulting clusters should contain at most two records, one from each data set. *Singular clusters* (also known as singletons) are also possible, indicating records for which no corresponding record has been found in the other data set.

To generate this clustering, a typical CCER pipeline [9] 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) *bipartite graph matching*, which receives the scored candidate pairs and decides which pairs will be placed together in a cluster. In this work, we evaluate how different methods for the last step perform when the previous steps 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 data sets, and $E \subseteq V_1 \times V_2$ is the set of edges with weights in [0,1] which correspond to the similarity scores between records of the two data sets. The output of bipartite graph matching comprises a set of clusters C, each 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 entity.

Figure 1a shows an example of a bipartite similarity graph in which node partitions (data sets) 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 score) of the adjacent nodes. Figure 1b–d shows three different outputs of CCER, in which nodes within the same oval (cluster) correspond to matching records.

2.1 Related work

There is a rich body of literature on ER [3, 8, 9, 46]. Following the seminal Fellegi-Sunter model for record linkage [16], 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 [62], the primary focus of most recent works has been on the effectiveness of the classification task, mainly by leveraging machine [28] and deep learning [5, 35, 40] methods.

Inspired by recent progress and success of prior work on improving the efficiency of ER with blocking and filtering [50], we target ER frameworks where the output of the matching step is used to construct a similarity graph that needs to be partitioned for the final step of ER. Hassanzadeh et al. [25] also target such a framework and perform an evalua-

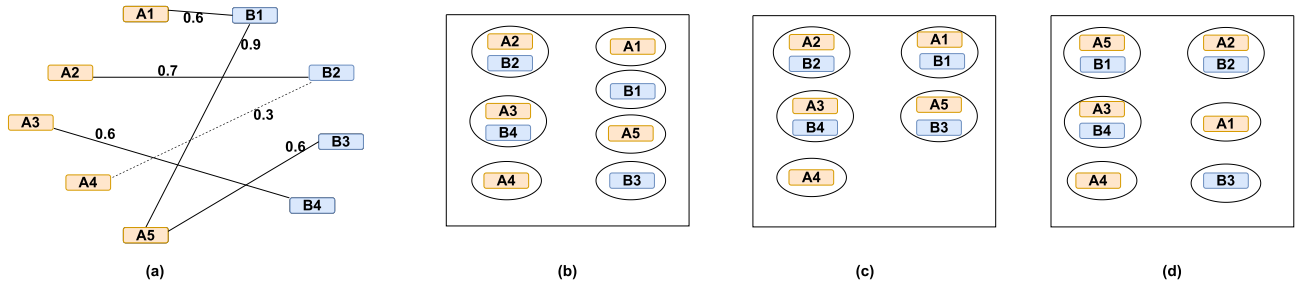

Fig. 1 Example of processing a bipartite similarity graph with a similarity threshold of 0.5 (i.e., edges with a lower score such as A4–B2 are ignored): **a** the similarity graph constructed for a pair of clean data sets (V_1 in orange and V_2 in blue), **b** the resulting clusters after applying CNC, **c** the resulting 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 the UMC, BMC, or EXC algorithms—we describe all these algorithms in Sect. 3 (color figure online)

tion of various graph clustering algorithms for ER. However, they target a scenario where input data sets are not clean, or where more than two clean data sets are merged into a dirty one 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* [9]. Some of the bipartite matching algorithms we use in this paper are adaptations of the graph clustering algorithms used in [25] for Dirty ER.

More recent clustering methods for Dirty ER were proposed in [14]. 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. [19] present two algorithms for CCER as well as more algorithms for different ER settings (such as 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 *Unique Mapping Clustering*. Finally, the *MaxWeight* method [19] utilizes the exact solution of the maximum weight bipartite matching, for which an efficient heuristic approach is considered in our *Best Assignment Heuristic Clustering*.

FAMER [55] 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*, which we describe in the following section.

Wang et al. [58] follow a reinforcement learning approach, based on a Q-learning [60] algorithm, for which a state is represented by the pair $(|L|, |R|)$, where $L \subseteq V_1$, $R \subseteq V_2$ are the nodes/records matched from the two input data sets, and the reward is calculated as the sum of the weights of the selected matches. We leave this algorithm outside the scope of this study, because we only consider learning-free methods. We plan to further explore such methods in the future.

Kriege et al. [30] present a linear approximation to the weighted graph matching problem. However, 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 similarities are in [0,1], as is the case with most matching algorithms.

3 Algorithms

We consider bipartite graph matching algorithms that satisfy 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 elsewhere [14, 25, 55].
2. Their functionality is learning-free in the sense that they do not learn a pruning model over a set of labeled 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

Algorithm 1: Connected Components (CNC)

```
   Input: Similarity graph G = (V1, V2, E), sim. threshold t
   Output: A set of clusters C = {c1, c2, ..., cn}
2  foreach e = (vi, vj, sim) ∈ E do
3      if sim < t then
4          E ← E \ {e}
5  C* ← getConnectedComponentsDFS(G)
6  C ← ∅
   foreach ci ∈ C* do
7      if |ci| = 2 then
8          C ← C ∪ {ci}
9  return C
```

Algorithm 2: Ricochet SR Clustering (RSR)

```
   Input: Similarity graph G = (V1, V2, E), sim. threshold t
   Output: A set of clusters C = {c1, c2, ..., cn}
1  C ← ∅
2  Center ← ∅
3  foreach v ∈ (V1 ∪ V2) do // Initialization
4      simWithCenter(v) ← 0
5      Cluster(v) ← ∅
6      centerOf(v) ← v
7  Q ← G.nodesInDecreasingWeight(v, w(v))
   // w(v) = (∑e∈adj(v) e.sim) / |adj(v)|
8  while Q ≠ ∅ do
9      vi ← Q.pop()          // the vertex with highest weight
10     ToReassign ← ∅
11     foreach e = (vi, vj, sim) ∈ E : sim > t do // for vi's adjacent
       edges
12         if vj ∈ Center then
13             continue
14         if e.sim > simWithCenter(vj) then
15             Cluster(centerOf(vj)).remove(vj)   // remove vj from
               its previous cluster
16             Cluster(vi) ← Cluster(vi) ∪ {vj}
17             ToReassign ← ToReassign ∪ centerOf(vj) // it is now
               a singleton
18             simWithCenter(vj) ← e.sim
19             centerOf(vj) ← vi
20             break
21     if |Cluster(vi)| > 0 then
22         if centerOf(vi) ≠ vi then // if vi was previously in
           another cluster
23             Cluster(centerOf(vi)).remove(vi)
24             ToReassign ← ToReassign ∪ centerOf(vi)
25         Center ← Center ∪ {vi}
26         Cluster(vi) ← Cluster(vi) ∪ {vi} // put vi in its cluster
27         centerOf(vi) ← vi
28         simWithCenter(vi) ← 1
29     foreach vk ∈ ToReassign do
30         maxSim ← 0
31         cMax ← vk
32         foreach e = (vk, vℓ, sim) ∈ E : sim > t do // find singleton
           with the highest similarity with vk to reassign
           it
33             if e.sim > maxSim and |Cluster(vℓ)| < 2 then
34                 cMax ← vℓ
35                 maxSim ← e.sim
36         if maxSim > 0 then
37             Cluster(vk) ← ∅
38             Cluster(cMax) ← Cluster(cMax) ∪ {vk}
39 foreach vi ∈ (V1 ∪ V2) do
40     if |Cluster(vi)| = 2 then
41         C ← C ∪ {Cluster(vi)}
42 return C
```

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* [31], whose time complexity is cubic, $O(n^3)$. For the same reason we exclude the work of Schwartz et al. [57] 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 and Tarjan [17] to provide an approximate solution in $O(n^2 \log n)$.

Note that most of the considered algorithms depend on the number of edges m 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, $O(n) \leq m \ll O(n^2)$.

In the following we describe the selected eight algorithms in detail.

Connected Components (CNC). This is the simplest algorithm. Its functionality is outlined in Algorithm 1. First, it discards all edges with a weight lower than the given similarity threshold (lines 1 to 3). Then, it computes the connected components of the pruned similarity graph (line 4). In the output, it solely retains the connected components (clusters) that contain two records—one from each input data set (lines 6 to 8).

Using a simple depth-first approach, its time complexity is $O(n + m) \sim O(m)$, if $m \gg n$ [10].

Ricochet Sequential Rippling Clustering (RSR). This algorithm, outlined in Algorithm 2, is an adaptation of the homonymous method for Dirty ER in [25] such that it exclusively considers clusters with just one record from each input data set. Initially, RSR sorts all nodes from both input data sets in descending order of the average weight of their adjacent edges (line 7). Whenever a new seed is chosen from the sorted list, we consider all its adjacent edges with a weight higher than t (lines 8 to 11). The first adjacent vertex that is currently unassigned or is closer to the new seed than it is to the seed of its current cluster is re-assigned to the new cluster (lines 14 to 16). If a cluster 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 cluster (lines 29 to 38). The algorithm stops when all nodes have been considered.

In the worst case, the algorithm iterates through n vertices and each time reassigns m vertices to their most similar adjacent vertex, therefore its time complexity is $O(n\,m + n \log n)$

Algorithm 3: Row Column Clustering (RCA)

Input: Similarity graph $G = (V_1, V_2, E)$, sim. threshold t
Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$

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 $Q_i \leftarrow V_2(sim(v_i))$ // a priority queue of V_2's nodes in decreasing sim with v_i
9 **while** $Q_i \neq \emptyset$ **do**
10 $v_j \leftarrow Q_i.pop()$
11 **if** $v_j \notin M_2$ **then** // if v_j is not yet matched
12 $C_1 \leftarrow C_1 \cup \{\{v_i, v_j\}\}$
13 $M_2 \leftarrow M_2 \cup \{v_j\}$
14 $D_1 \leftarrow D_1 + sim(v_i, v_j)$
15 **break**
16 **foreach** $v_j \in V_2$ **do**
17 $Q_j \leftarrow V_1(sim(v_j))$ // a priority queue of V_1's nodes in decreasing sim with v_j
18 **while** $Q_j \neq \emptyset$ **do**
19 $v_i \leftarrow Q_j.pop()$
20 **if** $v_i \notin M_1$ **then** // if v_i is not yet matched
21 $C_2 \leftarrow C_2 \cup \{\{v_i, v_j\}\}$
22 $M_1 \leftarrow M_1 \cup \{v_i\}$
23 $D_2 \leftarrow D_2 + sim(v_i, v_j)$
24 **break**
25 **if** $D_1 > D_2$ **then** // get maximal assignment
26 $C \leftarrow C_1$
27 **else**
28 $C \leftarrow C_2$
29 **foreach** $c = \{v_i, v_j\} \in C$ **do**
30 **if** $sim(v_i, v_j) < t$ **then** // check similarities
31 $C \leftarrow C \setminus \{c\}$ // remove clusters/record pairs with similarity less than t
32 **return** C

[61]. The latter part stems from the sorting of all nodes in line 7.

Row Column Assignment Clustering (RCA). This approach, outlined in Algorithm 3, is based on the Row-Column Scan approximation method in [32] 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 record from data set V_1 is placed in a new cluster with its most similar, currently unassigned record from data set V_2 (lines 7 to 15). In the second pass, the same procedure is applied to the records/nodes of data set V_2 (lines 16 to 24). The value of each solution is the sum of the edge weights (lines 14 and 23) between the nodes assigned to the same (2-node) cluster (lines 12 and 21). The solution with the highest value is returned as output, after discarding the pairs with a similarity less than t (lines 25 to 31).

At each pass the algorithm iterates over all nodes/records of one of the data sets searching for the node/record with maximum similarity from the other data set. Therefore, its time complexity is $O(|V_1||V_2|)$.

Algorithm 4: Best Assignment Heuristic (BAH)

Input: Similarity graph $G = (V_1, V_2, E) : |V_1| > |V_2|$, sim. threshold t, maximum iterations $maxIterations$
Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$

1 $C \leftarrow \emptyset$
2 $iteration \leftarrow 0$
3 **foreach** $(v_i^1, v_j^2) \in (V_1 \times V_2)$ **do**
4 $d(v_i^1, v_j^2) \leftarrow 0$ // initialize pair contributions
5 **foreach** $e = (v_i^1, v_j^2, sim) \in E : sim > t$ **do**
6 $d(v_i^1, v_j^2) \leftarrow sim$ // initialize pair contributions
7 **foreach** $v_i^1 \in V_1, v_i^2 \in V_2 : i \leq |V_2|$ **do**
8 $c_i \leftarrow \{v_i^1, v_i^2\}$ // initialize clusters
9 $p(v_i^1) = v_i^2$
10 **while** $iteration < maxIterations$ **do**
11 $iteration \leftarrow iteration + 1$
12 $i = nextRand(|V_1|)$ // get random i,j in V_1
13 $j = nextRand(|V_1|)$
14 **while** $j = i$ **do**
15 $j = nextRand(|V_1|)$
16 $D \leftarrow 0$
17 **if** $p(v_i^1) \neq null$ **then** // check swaps
18 $D \leftarrow d(v_j^1, p(v_i^1)) - d(v_i^1, p(v_i^1))$
19 **if** $p(v_j^1) \neq null$ **then** // check swaps
20 $D \leftarrow D + d(v_i^1, p(v_j^1)) - d(v_j^1, p(v_j^1))$
21 **if** $D \geq 0$ **then** // if swaps increase assignment value
22 $(p(v_j^1), p(v_i^1)) \leftarrow (p(v_i^1), p(v_j^1))$ // perform swaps
23 **foreach** $v_i \in V_1$ **do**
24 **if** $d(v_i, p(v_i)) > t$ **then** // check similarities
25 $C \leftarrow C \cup \{c_i\}$
26 **return** C

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 clusters. Its functionality is outlined in Algorithm 4.

Initially, each record from the smaller input data set, V_2, is connected to a record from the larger input data set, V_1 (lines 7 to 9). In each iteration of the search process (line 10), two records from V_1 are randomly selected (lines 12 to 15) 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 16 to 20), the swap is accepted (lines 21 and 22). The algorithm stops when a maximum number of search steps is reached or when a maximum run-time has been exceeded.

The time complexity of random search in lines 10–24 is determined by the maximum number of iterations, which is provided as input. Given that this number is a constant, the time complexity of Algorithm 4 is specified by the initialization in lines 3–9, which considers all pairs of nodes, i.e., $O(|V_1||V_2|)$.

Best Match Clustering (BMC). This algorithm is inspired from the Best Match strategy of [38], which solves the Stable Marriage problem [18], as simplified in BigMat [1]. Its functionality is outlined in Algorithm 5. For each record of the source data set V_s, this algorithm creates a new cluster

Algorithm 5: Best Match Clustering (BMC)

Input: Similarity graph $G = (V_1, V_2, E)$, sim. threshold t, source data set selection $s \in \{1, 2\}$
Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$

1 $C \leftarrow \emptyset$
2 $k \leftarrow \{1, 2\} \setminus s$ // the subscript of the target data set V_k
3 $M_k \leftarrow \emptyset$ // matched nodes from V_k
4 **foreach** $v_i \in V_s$ **do**
5 $Q_i \leftarrow v_i.edgesDecOrder(t)$ // edges in desc. sim > t
6 **while** $Q_i \neq \emptyset$ **do**
7 $(v_i, v_j, \_) \leftarrow Q_i.pop()$ // the best match of v_i is v_j; their similarity is ignored
8 **if** $v_j \notin M_k$ **then** // if v_j is not yet matched
9 $C \leftarrow C \cup \{\{v_i, v_j\}\}$ // match v_i with v_j
10 $M_k \leftarrow M_k \cup \{e.v_j\}$
11 **break**
12 **return** C

Algorithm 6: Exact Clustering (EXC)

Input: Similarity graph $G = (V_1, V_2, E)$, sim. threshold t
Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$

1 $C \leftarrow \emptyset$
2 **for** $i \in [1, |V_1|]$ **do**
3 $Q_i^1 \leftarrow \emptyset$ // create an empty PQ in desc. sim
4 **for** $j \in [1, |V_2|]$ **do**
5 $Q_j^2 \leftarrow \emptyset$ // create an empty PQ in desc. sim
6 **foreach** $e = (v_i^1, v_j^2, sim) \in E : sim > t$ **do**
7 $Q_i^1.push(e)$
8 $Q_j^2.push(e)$
9 **foreach** $v_i^1 \in V_1$ **do**
10 $(v_i^1, v_j^2, \_) \leftarrow Q_i^1.pop()$ // the best match for v_i^1 is v_j^2
11 $(v_k^1, v_j^2, \_) \leftarrow Q_j^2.pop()$ // the best match for v_j^2 is v_k^1
12 **if** $v_k^1 = v_i^1$ **then** // if the best match for v_j^2 is v_i^1
13 $C \leftarrow C \cup \{\{v_i^1, v_j^2\}\}$
14 **return** C

(lines 4 to 5), in which the most similar, not-yet-clustered record from the target data set V_k is also placed—provided that the corresponding edge weight is higher than t (lines 6 to 12).

Note that the greedy heuristic for BMC introduced in [38] is the same, in principle, to *Unique Mapping Clustering* discussed below. Note also that BMC involves an additional configuration parameter, apart from the similarity threshold: the source data set that is used as the source for creating clusters can be set to V_1 or V_2. In our experiments, we examine both options and retain the best one.

The algorithm iterates over the nodes of the source data set and in each turn, it searches for the adjacent vertex with maximum similarity. As a result, its time complexity is $O(m)$.

Exact Clustering (EXC). Inspired from the Exact strategy of [38], this algorithm places two records in the same cluster 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 [15].

In more detail, its functionality is outlined in Algorithm 6. Initially, it creates an empty priority queue for every vertex (lines 2 to 5). Then, it populates the queue of every vertex with all its adjacent edges that exceed the given similarity threshold t, sorting them in decreasing weight (lines 6 to 8). Subsequently, EXC places two records in the same cluster (lines 9 to 13) only if they are mutually the best matches, i.e., the most similar candidates of each other (line 12).

Its time complexity is $O(n\,m)$, since the algorithm iterates over each vertex in V_1 searching for its adjacent vertex with maximum similarity and then performs the same search for the latter vertex.

Király's Clustering (KRC). This is an adaptation of the linear time 3/2 approximation to the Maximum Stable Marriage problem, called "New Algorithm" in [27]. Its functionality is outlined in Algorithm 7.

Intuitively, the records of the source data set V_1 ("men" [27]), who are initially single (lines 2 and 7), propose to the records (line 17) from the target data set V_2 ("women" [27]) with an edge weight higher than t to form a cluster ("get engaged" [27]). The records of the target data set accept a proposal under certain conditions (e.g., if it's the first proposal they receive—line 18), and the clusters and preferences are updated accordingly (lines 19, 23 and 25). Records from the source data set (V_1) get a second chance to make proposals (lines 3, 6 and 28 to 31) and the algorithm terminates when all records of V_1 are in a cluster (line 14), or they have already made their second proposals without success (line 28). Note that for brevity, we omit some of the details (e.g., the rare case of "uncertain man") and refer the reader to [27] for more information, such as the acceptance criteria for proposals.

Its time complexity is $O(n + m \log m)$ [27].

Unique Mapping Clustering (UMC). This algorithm sorts edges in decreasing weight and iteratively forms a cluster for the top-weighted edge, as long as none of its nodes has been already matched. This comes from the *unique mapping constraint* of CCER, i.e., the restriction that each record from one data set matches with at most one record from the other. Note that the *CLIP Clustering algorithm*, introduced for the Multi-source ER problem in [56], is equivalent to UMC in the CCER case that we study.

In more detail, its functionality is outlined in Algorithm 8. Initially, it iterates over all edges and those with a weight higher than t are placed in a priority queue that sorts them in decreasing weight/similarity (lines 5 to 6). Subsequently, it iteratively forms a cluster (line 10) for the top-weighted pair (line 8), as long as none of its constituent records has already been matched to some other (line 9).

Its time complexity is $O(m \log m)$, due to the sorting of all edges (lines 5–6).

Algorithm 7: Király's Clustering

Input: Similarity graph $G = (V_1, V_2, E)$, sim. threshold t
Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$

$C \leftarrow \emptyset$
$singlM \leftarrow List()$ // an empty linked list of single men
$lastChance \leftarrow bool[|V_1|]$ // boolean array storing whether it's the last (T) chance or not (F) for each man
foreach $v_i \in V_1$ **do** // V_1 corresponds to men in [27]
  $Q_i^1 \leftarrow \emptyset$ // v_i's edges in desc. sim > t
  $lastChance[i] \leftarrow false$ // v_i's 1st out of 2 chances
  $singlM.addLast(v_i)$ // keeps insertion order
foreach $v_j \in V_2$ **do** // V_2 corresponds to women in [27]
  $Q_j^2 \leftarrow \emptyset$ // v_j's edges in desc. sim > t
  $fiancé[j] \leftarrow null$ // the man currently engaged to v_j
foreach $e = (v_i, v_j, sim) \in E : sim > t$ **do**
  $Q_i^1.push(e)$
  $Q_j^2.push(e)$
while $singlM \neq \emptyset$ **do**
  $v_i \leftarrow singlM.removeFirst()$ // in insertion order
  if $Q_i^1 \neq \emptyset$ **then**
    $v_j \leftarrow Q_i^1.pop()$ // v_i's preference is v_j
    if $fiancé[j] = null$ **then** // v_j is single
      $C \leftarrow C \cup \{\{v_i, v_j\}\}$ // match v_i to v_j
    else
      $v_i' \leftarrow fiancé[j]$ // v_j was engaged to v_i'
      if $acceptsProposal(v_j, v_i)$ **then** // cf. [27]
        $C \leftarrow C \setminus \{\{v_i', v_j\}\}$ // v_i' & v_j break up
        $singlM.addLast(v_i')$ // v_i' is now single
        $C \leftarrow C \cup \{\{v_i, v_j\}\}$ // match v_i to v_j
        $fiancé[j] \leftarrow v_i$ // v_j gets engaged to v_i
  else
    if $lastChance(v_i) = false$ **then**
      $lastChance[i] \leftarrow true$ // 2^{nd} chance for v_i
      $Q_i^1 \leftarrow recoverInitialQueue(v_i)$
      $singlM.addLast(v_i)$
return C

Algorithm 8: Unique Mapping Clustering

Input: Similarity graph $G = (V_1, V_2, E)$, sim. threshold t
Output: A set of clusters $C = \{c_1, c_2, \ldots, c_n\}$

$C \leftarrow \emptyset$
$M_1 \leftarrow \emptyset$ // matched nodes from V_1
$M_2 \leftarrow \emptyset$ // matched nodes from V_2
$Q \leftarrow \emptyset$
foreach $e = (v_i, v_j, sim) \in E : sim > t$ **do**
  $Q.push(e)$ // a PQ of edges in desc. sim > t
while $Q \neq \emptyset$ **do**
  $(v_i, v_j, \_) \leftarrow Q.pop()$ // the most similar pair
  if $v_i \notin M_1$ ***and*** $v_j \notin M_2$ **then** // v_i, v_j not matched
    $C \leftarrow C \cup \{\{v_i, v_j\}\}$
    $M_1 \leftarrow M_1 \cup \{v_i\}$
    $M_2 \leftarrow M_2 \cup \{v_j\}$
return C

Example. Fig. 1 demonstrates an example of applying the above algorithms to the similarity graph in Fig. 1a. For all algorithms, we assume a weight threshold of $t = 0.5$.

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

Algorithms that aim to maximize the total sum of edge weights between the matched records, such as RCA and BAH, will cluster $A1$ with $B1$ and $A5$ with $B3$, as shown in Fig. 1c, 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 Fig. 1d, because their candidates have already been matched to other records. The same output is produced by EXC, as the records in each cluster 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 source data set.

The clusters 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 Fig. 1d is the most likely one for these algorithms, too.

Configuration Parameters. The input of Algorithms 1 to 8 comprises the similarity graph and the similarity threshold t, with the latter constituting the sole configuration parameter in most cases. The only exceptions are BMC, which requires the specification of the source and the target data set, as well as BAH, which receives the maximum number of search steps. Note that for BAH, we set an additional parameter to restrict the maximum run-time per search step.

4 Similarity graphs

Two types of methods can be used for the generation of the similarity graphs that constitute the input to the above eight algorithms [7]:

1. *Learning-free* methods, which produce similarity scores in an unsupervised manner based on the content of the input records, 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 data sets without sacrificing valuable parts for the construction of the training (and perhaps the validation) set. We also avoid fine-tuning numerous configuration parameters, which is especially required in the case of deep learning-based methods [59]. Besides, our goal is not to optimize the performance of the CCER process, but to investigate

		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 SentenceBERT (S-T5)	1) Cosine Similarity 2) Euclidean Similarity 3) Earth Mover's Similarity	fastText and SentenceBERT (S-T5)	1) Cosine Similarity 2) Euclidean Similarity 3) Earth Mover's Similarity

Fig. 2 Taxonomy of the similarity functions we used to weigh 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 [44]. The graph similarities are defined in [20]

how the selected graph matching algorithms perform under a large variety of real settings. For this reason, we produce a large number of similarity graphs per data set, rather than relying on synthetic data.

In this context, we do not apply any blocking method when producing these inputs. Instead, we consider all pairs of records from different data sets with a similarity larger than 0. This allows for experimenting with a large variety of similarity graph sizes, which range from several thousand to hundreds of millions of edges. Besides, the role of blocking, i.e., the pruning of the record 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:

1. the *representation model*, and
2. the *similarity measure*.

For each part, we consider several established approaches from the literature, which are summarized in Fig. 2. We elaborate on them in the following.

4.1 Representation models

A 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:

1. *Schema-agnostic*, which consider all attribute values in a record, and
2. *Schema-based*, which consider only the value of a specific attribute.

Depending on their *form*, we also distinguish the representations into:

1. *Syntactic*, which operate on the original text of the records, and
2. *Semantic*, which operate on vector transformations (embeddings) of the original text. The aim of such representations is 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'}, where underscores represent whitespace characters.

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 [44]:

1. an n-gram vector [37], 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 [20], 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 data set 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.

Generally, a record r_i can be modeled as an n-gram vector with one dimension for every distinct (character or token) n-gram in a data set D: $VM(r_i) = (w_{i1}, \ldots, w_{im})$, where m stands for the *dimensionality* of D (i.e., the number of distinct n-grams in it), while w_{ij} is the weight of the j^{th} dimension that quantifies the importance of the corresponding n-gram for r_i.

The most common weighting schemes are:

1. *Term Frequency* (TF) sets weights in proportion to the number of times the corresponding n-grams appear in the values of record r_i. More formally, $TF(t_j, r_i) = f_j / N_{r_i}$, where f_j stands for the frequency of n-gram t_j in r_i, while N_{r_i} is the number of n-grams in r_i, normalizing TF so as to mitigate the effect of different lengths on the weights.
2. *Term Frequency-Inverse Document Frequency* (TF-IDF) discounts the TF weight for the most common tokens in the entire data set D, as they typically correspond to noise (i.e., stop words). Formally, $TF\text{-}IDF(t_j, r_i) = TF(t_j, r_i) \cdot IDF(t_j)$, where $IDF(t_j)$ is the inverse document frequency of the n-gram t_j, i.e., $IDF(t_j) = \log |D| / (|\{r_k \in D : t_j \in r_k\}| + 1)$. In this way, high weights are given to n-grams with high frequency in r_i, but low frequency in D.

To construct the vector model for a specific record, we aggregate the vectors corresponding to each one of its attribute values. The end result is a weighted vector $(a_i(w_1), ..., a(w_m))$, where $a_i(w_j)$ is the sum of weights, i.e., $a_i(w_j) = \sum_{a_k \in A_i} w_{ij}$, where a_k stands for an individual attribute value in the set of values A_i of record r_i.

Regarding the n-gram graphs, continuing our previous example, the character 3-gram graph corresponding to "Joe Biden" would be a graph with seven nodes, one for each 3-gram listed above. To capture the contextual knowledge, an edge of weight 1 connects the node 'Joe' to the nodes 'oe_' and 'e_B'. Similarly, 'oe_' is connected to 'e_B' and '_Bi' and so on, as shown in Fig. 3.

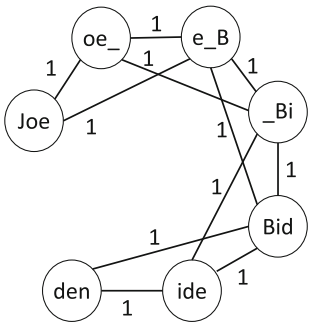

Fig. 3 The 3-gram graph corresponding to the string value "Joe Biden"

To construct the n-gram graph model that represents all attribute values in a record, we merge the individual graph of each value into a larger "record graph" through the update operator, as described in [20, 21]. The resulting graph is basically the union of the individual n-gram graphs with averaged weights.

For both the n-gram vectors and the n-gram graphs, we consider $n \in \{2, 3, 4\}$ for character- and $n \in \{1, 2, 3\}$ for token n-grams in our experiments in Sects. 5 and 6.

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 more similar their vectors are. These representations come in two main forms, which apply uniformly to schema-agnostic and schema-based settings:

1. 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* [39] and *GloVe* [51], suffer from a high portion of out-of-vocabulary tokens—these are words that cannot be transformed into a meaningful vector, because they are not included in the training corpora [40]. This drawback is addressed by the character-level embeddings: *fastText* vectorizes a token by summing the embeddings of all its character n-grams [4]. For this reason, we exclusively consider the 300-dimensional fastText in the following.
2. Transformer-based language models [12] 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 [5, 35]. Among them, we exclusively consider the 768-dimensional S-T5 [52], which is trained over hundreds of gigabytes of web documents in English, the Colossal Clean Crawled Corpus, thus being one of the best performing SentenceBERT models.

4.2 Similarity measures

Every similarity measure receives as input two representation models and produces a score that is proportional to the likelihood that the respective records correspond to the same real world entity: 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, as described below.

Schema-based syntactic representations. We distinguish the similarity measures for this type of representation models into:

1. character-level, which are applied to two strings s_1 and s_2 by treating them as character sequences, and
2. word-level, which are applied to two strings a and b by treating them as sets or multisets (bags) of words.

The former category includes the measures below:

Levenshtein distance. Counts the (minimum) number of insert, delete and substitute operations required to transform one string into the other.

Damerau-Levenshtein distance. The Damerau-Levenshtein distance only differs from the Levenshtein distance by including transpositions among the operations allowed.

Jaro similarity. The Jaro similarity of two strings s_1 and s_2 is defined as:

$$sim(s_1, s_2) = \begin{cases} 0 & \text{, if } m = 0 \\ \frac{1}{3}\left(\frac{m}{|s_1|} + \frac{m}{|s_2|} + \frac{m-t}{m}\right) & \text{, else,} \end{cases}$$

where m is the number of common characters and t the number of transpositions.

Needleman-Wunch. This similarity measure is the result of applying an algorithm that assigns three scores (seen as parameters) to two strings s_1 and s_2, depending on whether the aligned characters are a match, a mismatch, or a gap. A match occurs when the two aligned characters are the same, a mismatch when they are not the same, and a gap when an insert or delete operation is required for the alignment. The match, mismatch and gap scores used in this study are 0, -1 and -2, respectively, as in Simmetrics [6].

Q-grams distance. It applies Block distance (see below) to the 3-gram representation of s_1 and s_2.

Longest Common Substring similarity. It normalizes the size of the longest common substring (lcs_{str}) between two input strings by the size of the longest one: $sim(s_1, s_2) = |lcs_{str}(s_1, s_2)|/max(|s_1|, |s_2|)$.

Longest Common Subsequence similarity. The difference between this measure and the previous is that the common subsequence does not need to consist of consecutive characters.

We also consider the following word-level measures:

Cosine similarity. The similarity is defined as the Cosine of the angle between the multisets (bags) of words a and b, which are expressed as sparse vectors: $sim(a, b) = a \cdot b/(||a||\ ||b||)$.

Euclidean distance. Compares the frequency of occurrence of each word w in two strings a and b: $dist(a, b) = ||a - b|| = \sqrt{\sum_w (freqA(w) - freqB(w))^2}$.

Block distance. It is also known as *L1 distance*, *City Block distance* and *Manhattan distance*. Given two multisets (bags) of words a and b, it amounts to the sum of the absolute differences of the frequency of each word in a versus in b: $dist(a, b) = ||a - b||_1$.

Overlap coefficient. It is estimated as the size of the intersection divided by the smaller size of the two given *sets* of words: $sim(a, b) = |a \cap b|/min(|a|, |b|)$.

Dice similarity. It is defined as twice the shared information (intersection) divided by sum of cardinalities of the two *sets* of words: $sim(a, b) = 2|a \cap b|/(|a| + |b|)$.

Simon White similarity.[1] This similarity is the same as Dice similarity, with the only difference being that it considers a and b as multisets (bags) of words.

Jaccard similarity. It calculates the size of the intersection divided by the size of the union for the two given *sets* of words: $sim(a, b) = |a \cap b|/|a \cup b|$.

Generalized Jaccard similarity. It is the same as the Jaccard similarity, except that it considers multisets (bags) of words instead of sets.

Monge-Elkan similarity. This similarity is the average similarity of the most similar words between two sets of words a and b: $sim(a, b) = \frac{1}{|a|}\sum_{w_i \in a} max_{w_j \in b}\left(sim(w_i, w_j)\right)$, where sim is the optimized Smith-Waterman algorithm [22] that operates as the secondary character-level similarity to compute the similarity of individual words.

Schema-agnostic syntactic representations. As described above, this type of representation models comes in the form of n-gram vectors or n-gram graphs.

[1] http://www.catalysoft.com/articles/StrikeAMatch.html.

To compare two vector models $VM(r_i)$ and $VM(r_j)$, we consider the following similarity measures[2]:

ARCS similarity [47]. It sums the inverse Document Frequency (DF) of the common n-grams in two bag models. The rarer the common n-grams are, the higher gets the overall similarity. Formally: $sim(VM(r_i), VM(r_j)) = \sum_{t_k \in VM(r_i) \cap VM(r_j)} \log 2 / \log(DF_1(t_k) \cdot DF_2(t_k))$, where $t_k \in VM(r_i) \cap VM(r_j)$ indicates the set of common n-grams.

Cosine similarity. It measures the cosine of the angle between the weighted input vectors. Formally, it is equal to their dot product similarity, normalized by the product of their magnitudes: $sim(VM(r_i), VM(r_j)) = \sum_{k=1}^{m} w_{ik} w_{jk} / ||VM(r_i)|| / ||VM(r_j)||$, where m is the dimensionality of the vector models, i.e., $m = |VM(r_i)| = |VM(r_j)|$, while w_{lk} denotes the k^{th} dimension in the vector model $VM(r_l)$.

Jaccard similarity. It defines as similarity the ratio between the sizes of set intersection and union: $sim(VM(r_i), VM(r_j)) = |VM(r_i) \cap VM(r_j)| / |VM(r_i) \cup VM(r_j)|$.

Generalized Jaccard similarity. It extends the above measure so that it takes into account the weights associated with every n-gram: $sim(VM(r_i), VM(r_j)) = \frac{\sum_{k=1}^{m} min(w_{ik}, w_{jk})}{\sum_{k=1}^{m} max(w_{ik}, w_{jk})}$.

Both CS and GJS apply seamlessly to both TF and TF-IDF weights.

To compare two graph models, we consider the following graph similarity measures [20]:

Containment similarity (CoS). It estimates the number of edges shared by two graph models, G_i and G_j, regardless of the corresponding weights (i.e., it merely estimates the portion of common n-grams in the original texts). Formally: $CoS(G_i, G_j) = \sum_{e \in G_i} \mu(e, G_j) / min(|G_i|, |G_j|)$, where $|G|$ is the size of graph G, and $\mu(e, G) = 1$ if $e \in G$, or 0 otherwise.

Value similarity (VS). It extends CoS by considering the weights of common edges. Formally, using w_e^k for the weight of edge e in G_k: $VS(G_i, G_j) = \sum_{e \in (G_i \cap G_j)} \frac{min(w_e^i, w_e^j)}{max(w_e^i, w_e^j) \cdot max(|G_i|, |G_j|)}$.

Normalized Value similarity (NS). It extends VS by mitigating the impact of imbalanced graphs, i.e., the cases where the comparison between a large graph with a much smaller one yields similarities close to 0. Formally: $NS(G_i, G_j) = \sum_{e \in (G_i \cap G_j)} min(w_e^i, w_e^j) / max(w_e^i, w_e^j) / min(|G_i|, |G_j|)$.

Overall similarity (OS). It constitutes the average of the above graph similarity measures, which are all defined [0, 1]. Formally: $OS(G_i, G_j) = (CoS(G_i, G_j) + VS(G_i, G_j) + NS(G_i, G_j))/3$.

Semantic representations. Both the schema-agnostic and the schema-based representations of this type are associated with the three similarity functions. They all receive as input two dense multi-dimensional numeric vectors, $\mathbf{v_i}$ and $\mathbf{v_j}$, which

[2] For the measures that treat the vector models as sets, we assume that dimensions with weights higher than 0 indicate the presence of the corresponding n-gram.

Table 1 Technical characteristics of the real data sets for Clean-Clean ER in increasing computational cost ($|V_1| \times |V_2|$)

	D_{RE}	D_{AB}	D_{AG}	D_{DA}	D_{IM}	D_{IT}	D_{MT}	D_{WA}	D_{DS}	D_{ID}
$Dataset_1$	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	1076	1354	2616	5118	5118	6056	2554	2516	27,615
$\|V_2\|$	2256	1076	3039	2294	6056	7810	7810	22,074	61,353	23,182
NVP_1	1130	2568	5302	10,464	21,294	21,294	23,761	14,143	10,064	1.6×10^5
NVP_2	7519	2308	9110	9162	23,761	20,902	20,902	$1.14 times 10^5$	$1.98 times 10^5$	$8.2 times 10^5$
$\|A_1\|$	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	1076	1104	2224	1968	1072	1095	853	2308	22,863
$\|V_1\| \times \|V_2\|$	$7.65 times 10^5$	$1.16 times 10^6$	$4.11 times 10^6$	$6.00 times 10^6$	$3.10 times 10^7$	$4.00 times 10^7$	$4.73 times 10^7$	$5.64 times 10^7$	$1.54 times 10^8$	$6.40 times 10^8$

are the embedding representations of records r_i and r_j, and produce as output a score in [0, 1] that is proportional to the matching likelihood of r_i and r_j (i.e., 0 corresponds to dissimilarity and 1 to identical representations). More specifically, we consider the following similarity functions:

1. Cosine similarity, which is the dot product between v_i and v_j normalized by the product of their magnitudes: $sim(\mathbf{v_i}, \mathbf{v_j}) = \mathbf{v_i} \cdot \mathbf{v_j}/(||\mathbf{v_i}|| \cdot ||\mathbf{v_j}||)$.
2. Euclidean similarity, which is defined as $sim(\mathbf{v_i}, \mathbf{v_j}) = 1/(1 + euDist(\mathbf{v_i}, \mathbf{v_j}))$, where $euDist(\mathbf{v_i}, \mathbf{v_j}))$ is the Euclidean distance between the embedding vectors, i.e., $euDist(\mathbf{v_i}, \mathbf{v_j}) = \sqrt{\sum_{k=1}^{|v_i|}(v_i^k - v_j^k)^2}$.
3. Earth mover's similarity, which is defined as $sim(\mathbf{v_i}, \mathbf{v_j}) = 1/(1 + emDist(\mathbf{v_i}, \mathbf{v_j}))$, where $emDist$ stands for the Earth mover's distance, also known as Wasserstein distance [54]. In essence, it estimates the minimum distance that the n-grams describing r_i need to "travel" in the semantic space in order to reach the n-grams in r_j [33].

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) and 128 GB of RAM. 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.[3] For the schema-agnostic syntactic similarity functions, we used the implementation provided by the JedAI toolkit [48] (the implementation of n-gram graphs and the corresponding graph similarities is based on the JIinsect toolkit[4]). For the semantic representation models, we employed the Python sister package,[5] which offers the fastText pre-trained embeddings, and Hugging Face,[6] which implements the S-T5 pre-trained embeddings. For the computation of the semantic similarities, we used the Python scipy package.[7]

Data Sets. In our experiments, we use ten real-world, established data sets for ER. Their characteristics are shown in Table 1, where $|V_x|$ stands for the number of input records, $|NVP_x|$ 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 record 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_{RE}, which was introduced in OAEI 2010,[8] contains data about restaurants. D_{AB} matches products from the online retailers Abt.com and Buy.com [29]. D_{AG} interlinks products from Amazon and the Google Base data API (Google Pr.) [29]. D_{DA} contains data about publications from DBLP and ACM [29]. D_{IM}, D_{IT} and D_{MT} contain data about television shows from TheTVDB.com (TVDB) and movies from IMDb and themoviedb.org (TMDb) [42]. D_{WA} contains data about products from Walmart and Amazon [40]. D_{DS} contains data about scientific publications from DBLP and Google Scholar [29]. D_{ID} matches movies from IMDb and DBpedia [48] (note that D_{ID} contains a different snapshot of IMDb movies than D_{IM} and D_{IT}). All these data sets are publicly available through the JedAI data repository.[9]

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 records, while conveying a rich diversity of values, thus yielding high effectiveness. These attributes are "name" and "phone" for D_{RE}, "name" for D_{AB}, "title" for D_{AG}, "title" and "authors" for D_{DA}, "modelno" and "title" for D_{IM}, "title" and "authors" for D_{IT}, "name" and "title" for D_{MT}, "title" and "name" for D_{WA}, "title" and "abstract" for D_{DS}, and "title" for D_{ID}.

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

- *Precision* (Pr) measures the portion of output clusters with two nodes from every partition that are indeed duplicates, i.e., $Pr = |\{c_i \in C | (v_l \in c_i \cap V_1) \wedge (v_k \in c_i \cap V_2) \wedge (v_l \equiv v_k) \wedge (|c_i| = 2)\}|/|\{c_i \in C | |c_i \cap V_1| = 1 \wedge |c_i \cap V_2| = 1\}$.
- *Recall* (Re) measures the portion of matching nodes that co-occur in at least one cluster of the output: $Re = |\{c_i \in C | (v_l \in c_i \cap V_1) \wedge (v_k \in c_i \cap V_2) \wedge (v_l \equiv v_k) \wedge (|c_i| = 2)\}|/|\{(v_l, v_k) \in V_1 \times V_2 | v_l \equiv v_k\}|$.
- *F-Measure* ($F1$) is the harmonic mean of precision and recall: $F1 = 2 \cdot Pr \cdot Re/(Pr + Re)$.

All measures 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 an algorithm requires from receiving the weighted similarity graph as input until it returns the generated partitions as output, over 10 repeated executions.

[3] https://github.com/Simmetrics/simmetrics.

[4] https://github.com/ggianna/JInsect.

[5] https://pypi.org/project/sister.

[6] https://huggingface.co.

[7] https://www.scipy.org.

[8] http://oaei.ontologymatching.org/2010/im.

[9] https://github.com/scify/JedAIToolkit/tree/master/data.

Table 2 The number of similarity graphs $|G|$, their size in terms of the average number of edges $|\bar{E}|$, and their average normalized size ($|\bar{E}|/|V_1| \times |V_2|$), per data set

	Syntactic weights						Semantic weights					
	Schema-based			Schema-agnostic			Schema-based			Schema-agnostic		
	$\|G\|$	$\|\bar{E}\| \cdot 10^6$	$\frac{\|\bar{E}\|}{\|V_1\|\times\|V_2\|}$	$\|G\|$	$\|\bar{E}\| \cdot 10^6$	$\frac{\|\bar{E}\|}{\|V_1\|\times\|V_2\|}$	$\|G\|$	$\|\bar{E}\| \cdot 10^6$	$\frac{\|\bar{E}\|}{\|V_1\|\times\|V_2\|}$	$\|G\|$	$\|\bar{E}\| \cdot 10^6$	$\frac{\|\bar{E}\|}{\|V_1\|\times\|V_2\|}$
D_{RE}	20	0.16	21.2%	46	0.72	93.5%	9	0.22	29.3%	4	0.76	100.0%
D_{AB}	12	1.05	90.5%	47	0.64	55.1%	4	1.16	100.0%	4	1.16	100.0%
D_{AG}	14	2.89	70.5%	53	2.65	64.5%	4	4.11	100.0%	4	4.11	100.0%
D_{DA}	27	4.49	74.8%	24	3.84	64.0%	12	6.00	99.9%	4	6.00	100.0%
D_{IM}	24	5.81	18.7%	48	11.92	38.5%	12	8.29	26.8%	4	30.81	99.4%
D_{IT}	25	8.39	21.0%	45	10.99	27.5%	12	12.40	31.0%	4	39.89	99.8%
D_{MT}	26	2.80	05.9%	42	12.21	25.8%	12	3.71	07.8%	6	47.07	99.35%
D_{WA}	26	28.10	49.8%	47	37.31	66.2%	6	43.93	77.9%	2	56.38	100.0%
D_{DS}	20	119.18	77.2%	46	77.56	50.2%	8	154.36	100.0%	4	154.36	100.0%
D_{ID}	13	250.73	39.2%	43	317.17	49.5%	4	378.56	59.1%	2	640.10	99.99%
Σ	207	–	–	441	–	–	83	–	–	38	–	–

Generation Process. To generate a large variety of input similarity graphs, we apply every similarity function described in Sect. 4 to all data sets in Table 1. We apply all combinations of representation models and similarity measures, thus yielding 60 schema-agnostic syntactic similarity graphs per data set, 16 schema-based similarity graphs per attribute in each data set, and 12 semantic similarity graphs per data set. 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 evaluate the performance of all algorithms, 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.[10] The largest threshold that achieves the highest $F1$ is selected as the optimal one, determining the performance of the algorithm for the particular input. Note that for BAH we set the maximum number of search steps to 10,000, and the maximum run-time per input to 2 min, given that all other algorithms process any similarity graph within seconds.

Next, we took special care to clean the experimental results from noise. We removed all similarity graphs where all matching records had a zero edge weight. We also removed all noisy graphs, where all algorithms achieve an $F1$ lower than 0.25. Finally, we cleaned our data from duplicate inputs. These are similarity graphs that emanate from the same data set but from 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. As such, we consider the cases where the difference in $F1$ and precision or recall is less than 0.002 (i.e., 0.2%). Note that we did not set the difference to 0, so as to accommodate slight differences in the performance of stochastic algorithms like RCA and BAH.

The characteristics of the retained similarity graphs are shown in Table 2. Overall, there are 769 different similarity graphs, most of which rely on syntactic similarity functions and the schema-agnostic settings, in particular. The reason is that much fewer similarity functions are associated with the schema-based settings, especially the semantic ones. Every data set is represented by at least 62 similarity graphs, with every type of weights including at least two graphs. The average number of edges in these graphs ranges from 160K to 379 M. This large set of real-world similarity graphs allows for a rigorous testing of the graph matching algorithms under diverse conditions.

[10] 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, similarity functions and data sets they involve.

6 Experimental analysis

We now discuss our experimental evaluation of bipartite matching algorithms for CCER with regard to a variety of dimensions.

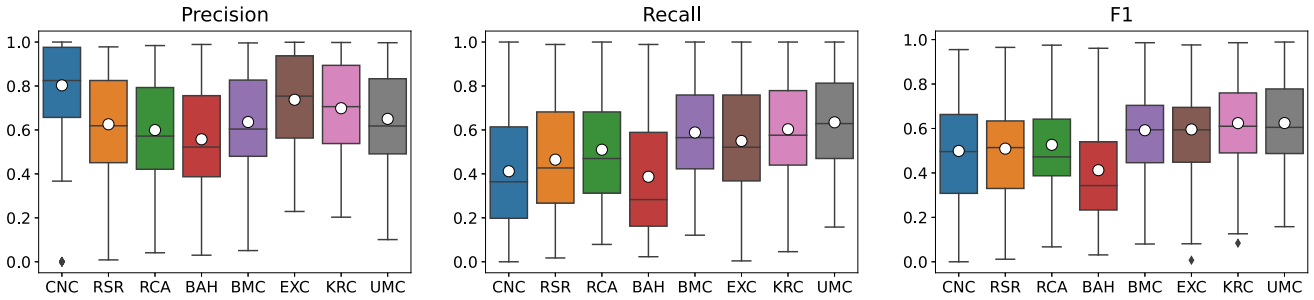

Fig. 4 Precision, recall and $F1$ of all algorithms over all similarity graphs

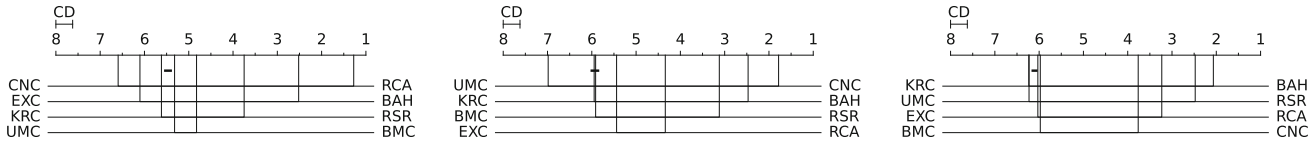

Fig. 5 Nemenyi diagrams based on precision, recall and $F1$ (from left to right)

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. This section examines 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 distribution of all effectiveness measures per algorithm across all input similarity graphs, as reported in Fig. 4. We observe that all algorithms emphasize precision at the cost of lower recall. Based on the average values, the most balanced algorithm is UMC, as it yields the smallest difference between the two measures (just 0.015). In contrast, CNC constitutes the most imbalanced algorithm, as its precision is almost double its recall. The former achieves the best $F1$, followed in close distance by KRC, while the latter achieves the second worst $F1$, surpassing only BAH. Judging from the range of the box plots, BAH is the least robust with respect to all measures, due to its stochastic functionality, while CNC, UMC, KRC and RSR are the most robust with respect to precision, recall and $F1$, respectively. Among the other algorithms, EXC and BMC are closer to UMC and KRC, with the former achieving the third highest $F1$. In contrast, 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 [26] based on their effectiveness measures over the 769 paired samples. In more detail, for each measure, we first perform the non-parametric Friedman test [11] 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 [41] to identify the critical distance (CD = 0.379) between the methods. The resulting Nemenyi diagrams appear in Fig. 5. We observe that for precision and recall, only the difference between RSR and BMC is statistically insignificant. In every diagram, the methods are listed in descending order of performance, with the best performing method appearing in the top left position and the worst one in the bottom right position. Hence, the best algorithm in terms of precision is CNC, followed by EXC, KRC and UMC, while the best recall is achieved by UMC, with KRC in the second place. These results verify the patterns in Figs. 4 and 6, highlighting the excellent balance between precision and recall that is achieved by UMC and KRC. Regarding $F1$, there are no statistically significant differences among the methods with the lowest performance, namely CNC, RCA, RSR, and BAH. All other differences are significant, with KRC, UMC, EXC, and BMC, ranking first (in that order) for both evaluation methods.

Another interesting observation drawn from these patterns is that EXC typically achieves higher precision and lower recall than BMC. This should be expected, given that EXC requires an additional reciprocity check before declaring that two records match. We notice, however, that the gain in precision is greater than the loss in recall and, thus, EXC yields a higher $F1$ than BMC, on average. Note also that in the vast majority of cases, BMC works best when choosing the smallest data set as the basis for creating clusters.

Overall, the top performing algorithms, on average, with respect to all effectiveness measures are KRC, UMC, BMC *and*

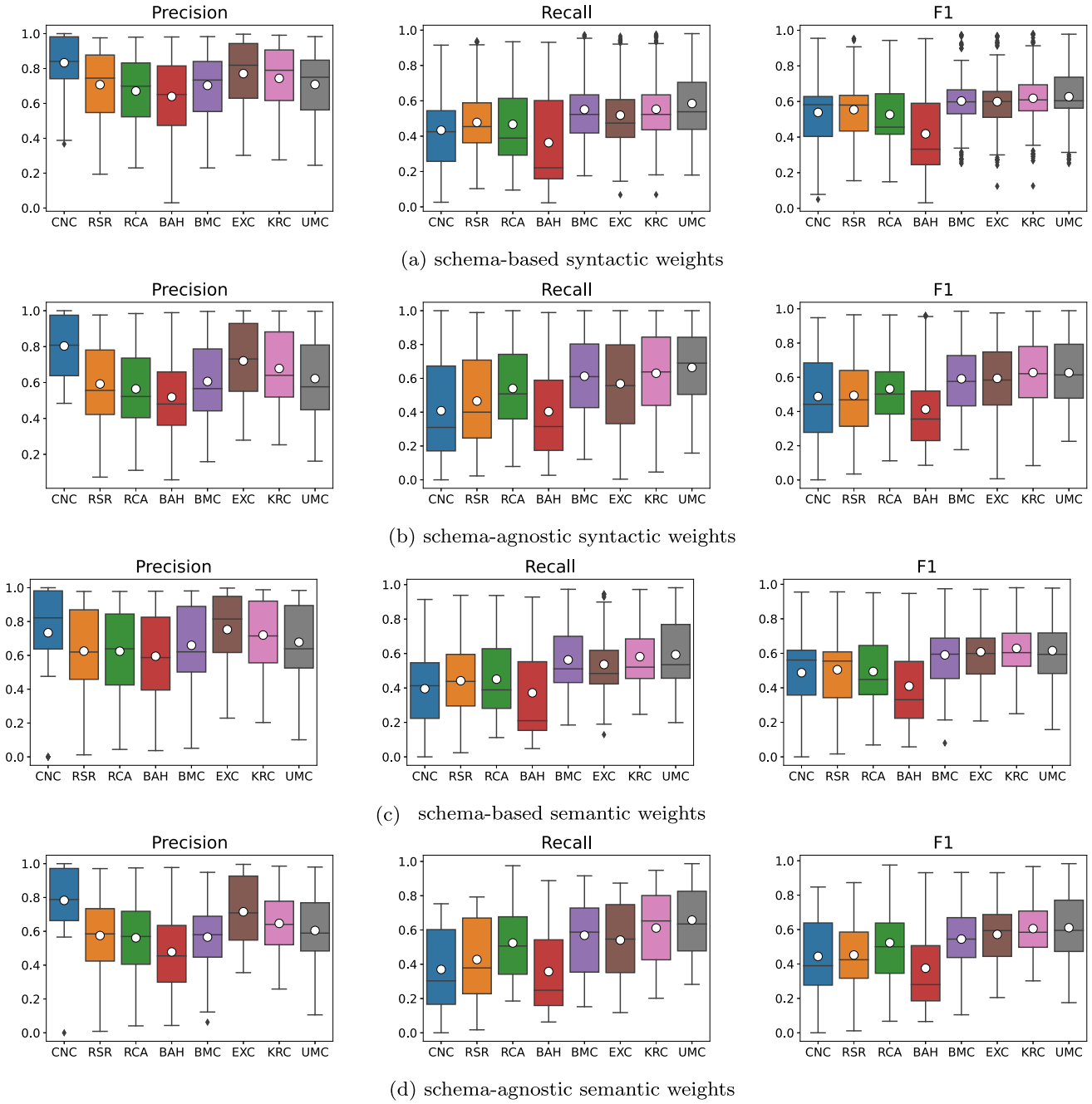

Fig. 6 Precision, recall and $F1$ of all algorithms over all similarity graphs of each type of edge weights

EXC. *Their relative performance depends on the type of edge weights, as explained below.*

To answer **QE(3)**, Fig. 6 presents the distribution of all effectiveness measures per algorithm across the four types of similarity graphs' origin. For the schema-based syntactic weights, we observe in Fig. 6a that the average precision of all algorithms is much higher than the corresponding precision in Fig. 4—from 3.6% (CNC) to 13% (BAH). For CNC and RSR, this is accompanied by an increase in average recall (by 5.1% and 2.7%, respectively), while for all other algorithms, the average recall drops between 6% (EXC) and 9.2% (KRC). This means that the schema-based syntactic similarities reinforce the imbalance between precision and recall in Fig. 4 in favor of the former, for all algorithms except CNC and RSR. The average $F1$ drops only for KRC (by 1%). UMC is the best algorithm with respect to both measures, outperforming the second best, KRC, by ~1.5%. Similarly, BMC exceeds EXC in terms of average $F1$ by ~0.6% in both cases, because the increase in its mean precision is much higher than the decrease in its mean recall, while the opposite is true for

EXC. Finally, it is worth noting that this type of similarities increases significantly the robustness of all algorithms, as the standard deviation of both $F1$ drop by $\geq$13% for all algorithms—the only exception is BAH, due to its stochastic nature.

The opposite patterns are observed for schema-agnostic syntactic weights in Fig. 6b: the imbalance between precision and recall is reduced, as on average, the former drops from 2.2% (EXC) to 7.3% (BAH), while the latter raises from 3.2% (EXC) to 5.6% (RCA). The imbalance is actually reversed for BMC and UMC, whose average recall (0.613 and 0.664, resp.) exceeds the average precision (0.606 and 0.622, resp.). The only exception is CNC, which remains practically stable with respect to both measures, while the same applies to the recall of RSR. Overall, compared to Fig. 4, there are minor changes in the mean $F1$ of most algorithms ($\ll$1% in absolute terms), with KRC and EXC exhibiting almost identical values for both measures with UMC and BMC, respectively.

The schema-based semantic similarity weights in Fig. 6c exhibit similar patterns with their syntactic counterparts, favoring precision over recall for all algorithms. In more detail, most algorithms raise their precision to a minor extent, from 2.2% (EXC) to 6.8% (BAH), on average. The only exceptions are RCA, which practically remains stable, and CNC, whose precision drops by 8.5%. At the same time, all algorithms reduce recall to a significant extent, by 5.2%, on average. The combination of these patterns leads to minor differences ($\ll$ 1%) in the average $F1$ of most algorithms. Changes up to 2% are observed only for CNC, whose performance deteriorates with respect to all measures, EXC, where the increase in precision is higher than the drop in recall, and vice versa for UMC and RCA. In these settings, KRC and EXC outperform UMC and BMC by 2–3% with respect to $F1$.

Finally, the schema-agnostic semantic similarity weights in Fig. 6d give rise to patterns similar to their syntactic counterparts in the sense that the deviation between precision and recall is reduced for practically all algorithms. In fact, UMC reverses the imbalance in favor of recall, while EXC balances both measures. This stems from the deterioration of all evaluation measures. Compared to Fig. 4, the average precision drops by at least 2.6%, with an average of 8.2% across all algorithms. Average recall drops to a lesser extent for most algorithms, with KRC, RCA and UMC actually raising it by 1.3%, 2.7% and 3.5%, respectively. The average $F1$ decreases for all algorithms, by 6.7% and 8.8%, respectively, except for RCA, where it remains practically stable. UMC takes a minor lead over KRC, being the overall top performer, but BMC clearly underperforms EXC.

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 the input data sets, $|V_1|$ and $|V_2|$:

1. *Balanced* (BLC) are the data sets where the vast majority of records in V_i are matched with a record in V_j (i = 1 $\wedge$ j = 2 $\bigvee$ i = 2 $\wedge$ j = 1). This category includes all similarity graphs generated from D_{AB}, D_{DA} and D_{ID}.
2. *One-sided* (OSD) are the data sets where only the vast majority of records in V_1 are matched with a record from V_2, or vice versa. OSD includes all graphs stemming from D_{AG} and D_{DS}.
3. *Scarce* (SCR) are the data sets where a small portion of records in V_i are matched with a record in V_j (i = 1 $\wedge$ j = 2 $\bigvee$ i = 2 $\wedge$ j = 1). This category includes all graphs generated from D_{RE} and D_{IM}-D_{WA}.

We apply this categorization to the four main types of edge weights defined in Sect. 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. 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
3. $\#Top2$ denotes the number of times an algorithm scores the second highest $F1$ for a particular category of similarity graphs.

Note that in the 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 3. 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 data sets. Yet, UMC exhibits consistently higher performance, because it ranks second whenever 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 data sets, KRC takes a minor lead over UMC with respect to $\#Top1$. Even though its Δ is slightly lower, it comes second twice more often than UMC. For scarce data sets, 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 data sets. RSR actually achieves almost the highest Δ, while EXC achieves the second highest $\#Top1$ and $\#Top2$, outperforming UMC. Regarding BAH, we observe that it is more suitable for balanced data sets, where it outperforms all algorithms for 15% of the similarity graphs, comes second for an equal number of inputs

Table 3 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) data sets OVL stands for the overall sums or averages across all similarity graphs per category. Note that there are ties for both $\#Top1$ and $\#Top2$: 47 and 72, resp., over schema-based syntactic weights, 45 and 30, resp., over schema-agnostic syntactic weights, 9 and 13, resp., over schema-based semantic weights, as well as 1 and 2, resp., over schema-agnostic syntactic weights. Note also that– indicates that a method was never among the top performing ones for a particular category

		Syntactic weights								Semantic weights							
		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	$\#Top1$	–	–	21	21	–	–	48	48	–	–	3	3	–	–	2	2
	Δ (%)	–	–	0.52	0.52	–	–	**7.57**	**7.57**	–	–	0.27	0.26	–	–	2.65	2.65
	$\#Top2$	–	–	13	13	–	–	13	13	–	1	4	5	–	–	1	1
RSR	$\#Top1$	–	–	4	4	–	–	1	1	–	–	–	–	–	–	–	–
	Δ (%)	–	–	1.91	1.91	–	–	0.46	0.46	–	–	–	–	–	–	–	–
	$\#Top2$	–	–	13	13	–	–	6	6	–	–	1	1	–	–	1	1
RCA	$\#Top1$	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	Δ (%)	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–	–
	$\#Top2$	–	–	–	–	–	–	1	1	–	–	–	–	2	–	–	2
BAH	$\#Top1$	8	–	1	9	41	–	3	44	4	–	3	7	**4**	–	1	5
	Δ (%)	3.67	–	**1.93**	**3.47**	**5.53**	–	0.65	5.20	**9.81**	–	0.85	**5.97**	**9.48**	–	0.52	**7.69**
	$\#Top2$	8	–	3	11	6	6	5	17	–	–	2	2	–	–	–	–
BMC	$\#Top1$	–	–	12	12	–	–	10	10	–	–	4	4	–	–	–	–
	Δ (%)	–	–	0.54	0.54	–	–	1.82	1.82	–	–	0.16	0.16	–	–	–	–
	$\#Top2$	5	2	32	39	15	6	19	40	–	–	5	5	–	–	1	1
EXC	$\#Top1$	–	5	41	46	–	13	80	93	–	3	19	22	–	–	7	7
	Δ (%)	–	0.78	0.81	0.80	–	0.40	2.10	1.87	–	0.86	1.75	2.39	–	–	2.39	2.39
	$\#Top2$	–	10	36	46	–	20	80	100	–	4	**19**	23	–	**4**	5	9
KRC	$\#Top1$	**23**	**17**	**48**	**88**	18	**52**	**84**	**154**	**15**	**9**	**25**	**49**	3	**8**	**9**	**20**
	Δ (%)	1.47	1.89	0.46	1.00	4.61	2.61	4.12	3.66	1.85	1.38	**2.96**	2.33	3.69	**5.48**	**4.52**	4.78
	$\#Top2$	11	**15**	**61**	**87**	39	**36**	**89**	**164**	1	3	18	22	1	–	**6**	7
UMC	$\#Top1$	**23**	15	36	74	**58**	43	35	136	1	2	4	7	3	–	2	5
	Δ (%)	**4.94**	**2.14**	1.15	2.53	4.88	**3.39**	2.28	3.74	0.37	**3.08**	0.45	1.19	0.65	–	0.62	0.64
	$\#Top2$	**29**	7	34	70	**56**	34	40	130	**19**	**5**	14	**38**	**7**	**6**	**6**	**19**

Bold value indicates the best performance per column

and achieves the second highest Δ. This is in contrast to the poor average performance reported in Fig. 4, but is explained by its stochastic nature, which gives rise to an unstable performance, as indicated by the higher range than all other algorithms for all effectiveness measures.

In the case of schema-agnostic syntactic edge weights, UMC outperforms KRC in the case of balanced data sets with respect to all measures. KRC is also outperformed by BAH, which is the top performer for 36% of the graphs of this type, while exhibiting the highest Δ among all algorithms. For one-sided data sets, 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 data sets, the two competing algorithms are KRC and EXC, with the former taking a minor lead. It exhibits slightly higher $\#Top1$ and $\#Top2$, while its Δ is almost double than that of EXC. Surprisingly, CNC ranks third 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: KRC consistently exhibits the highest $\#Top1$ across all dataset and weight types. There is a single exception, namely the balanced graphs with schema-agnostic weights, where BAH is the top performer. Similarly, UMC exhibits the highest $\#Top2$ in all cases, but the scarce schema-based weights, where it ranks third, behind EXC and KRC. For balanced datasets, BAH achieves by far the highest Δ, regardless of the schema settings. The same applies to KRC for the scarce ones, while for the one-sided ones, the best Δ depends on the origin of weights: the schema-based settings favor UMC and the schema-agnostic ones KRC.

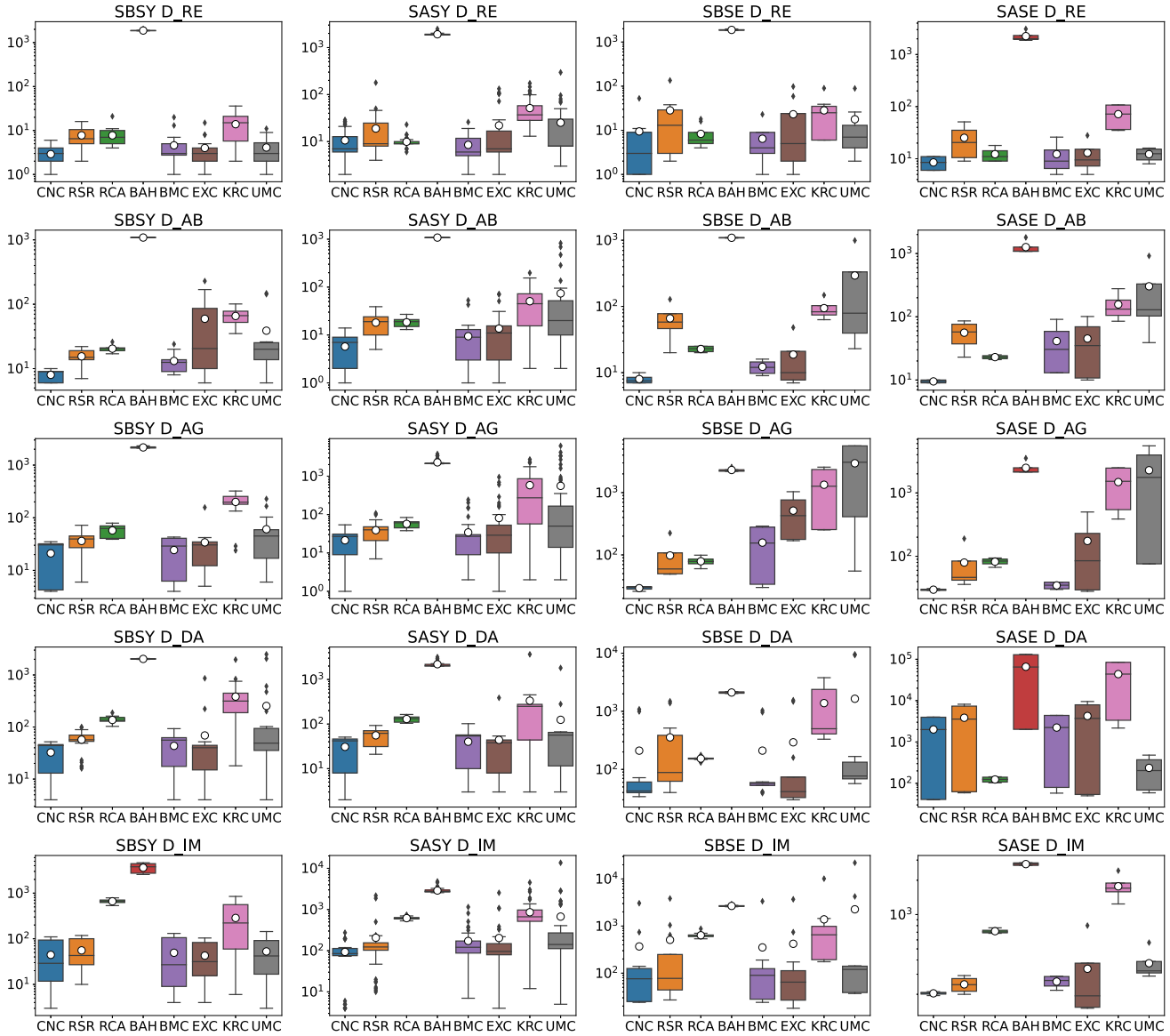

Fig. 7 The distribution of run-time (in ms) per algorithm and type of edge weights across the five smallest data sets. Every row corresponds to a different data set. Note the log-scale of the vertical axes. SBSY: schema-based syntactic weights, SASY: schema-agnostic syntactic weights, SBSE: schema-based semantic weights, SASE: schema-agnostic semantic weights

Overall, we can conclude that on average, the dominant algorithm for the balanced datasets is UMC, with its overall #$Top1$ amounting to 85 out of 196 graphs, leaving KRC in the second place, with an overall #$Top1$ equal to 59. Their relative performance is reversed only for schema-based semantic weights. BAH performs exceptionally well regardless of the weight type, with an overall #$Top1$ equal to 57. No other algorithm excels in this type of datasets, except for BMC and RCA, which rank second in very few cases. For the one-side datasets, which comprise 153 graphs in total, KRC takes the lead, leaving UMC and EXC in the second and third places, respectively. Their overall #$Top1$ amounts to 86, 60 and 21, respectively. Among the rest of the algorithms, BMC, BAH and CNC rank second in very few cases. For the scarce datasets (420 graphs in total), KRC retains its clear lead (overall #$Top1 = 166$), especially in the case of semantic weights. EXC follows in close distance, outperforming UMC to a large extent: the overall #$Top1$ of the former is almost double that of the latter (147 vs. 77). High performance over these datasets is also achieved by CNC and BMC and (less frequently) by RSR and BAH. All algorithms rank second at least once for this type of graphs.

In a nutshell, UMC *and* KRC *are among the most effective algorithms regardless of the type of edge weights and the type*

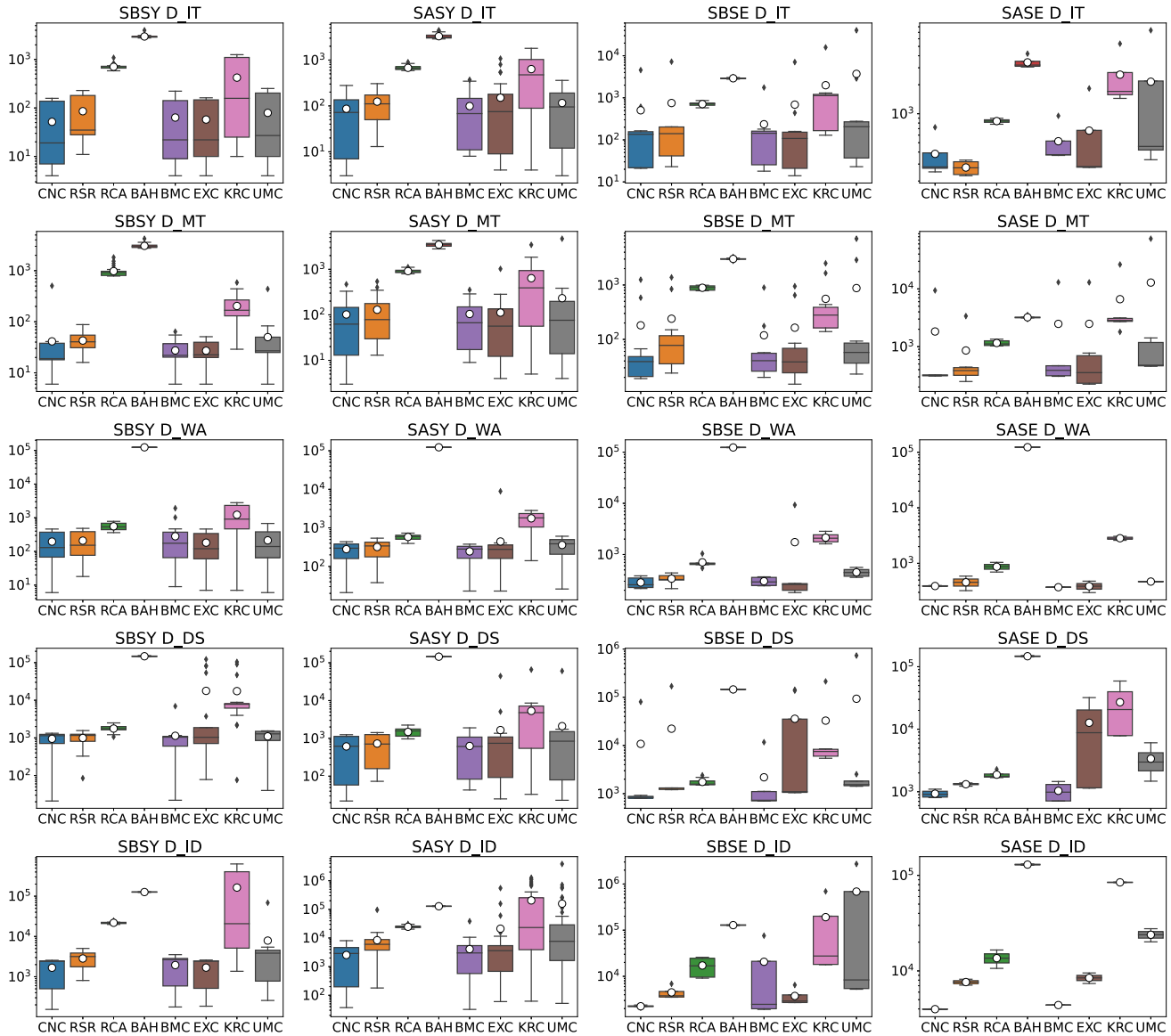

Fig. 8 The distribution of run-time (in ms) per algorithm and type of edge weights across the five largest data sets. Every row corresponds to a different data set. Note the log-scale of the vertical axes. SBSY: schema-based syntactic weights, SASY: schema-agnostic syntactic weights, SBSE: schema-based semantic weights, SASE: schema-agnostic semantic weights

of similarity graphs. BAH *is also quite effective for balanced and (more rarely) for one-sided graphs and* EXC *for the one-sided and scarce ones, while* CNC *and* BMC *yield competitive effectiveness over scarce graphs.*

Note that we looked for similar patterns with respect to additional characteristics of the data sets, such as the distribution of positive and negative weights (i.e., between matching and non-matching records, respectively) and the domain (e-commerce for D_{AB}, D_{AG} and D_{WA}, bibliographic data for D_{DA} and D_{DS} as well as movies for D_{IM}-D_{WA} and D_{ID}). 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 ER, due to the very large similarity graphs, which comprise thousands of records/nodes and (hundreds of) millions of edges/record pairs, as reported in Table 2. Below, we study this aspect along with the scalability of the considered algorithms over the 769 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?

Regarding **QT(1)**, Figs. 7 and 8 show the average run-times over 10 executions of the evaluated algorithms per data set and type of edge weights. To accommodate the much larger scale of the slowest algorithms, all diagrams use logarithmic scale in their vertical axis.

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_{DS} and D_{ID}) within few seconds—the vast majority of runs is below 10^4 milliseconds. Note also that the maximum scale in the last two rows of Fig. 8, which stem from the two largest datasets, corresponds to 10^6 msec $\approx$ 17 min. CNC is the fastest one, on average, in practically all data sets, due to the simplicity of its approach. It is followed in close distance by BMC and EXC, with the former consistently outperforming the latter, due to the additional reciprocity check of EXC. On the other extreme lies BAH, which constitutes by far the slowest method, yielding in many cases two or even three orders of magnitude longer run-times. The reason is the large number of search steps we allow per data set (10,000). For the largest data sets, its maximum run-time actually equals the run-time limit of 2 min. Regarding the remaining algorithms, KRC is usually the second slowest one, on average, while RCA exhibits significantly lower run-times. UMC is usually faster than both of these algorithms, but outperforms RSR only in the case of schema-based syntactic weights. Among the most effective algorithms, EXC is significantly faster than UMC and KRC.

These patterns are verified by Fig. 9, which summarizes the ranking positions per algorithm with respect to run-time across the four types of similarity scores. We observe that CNC consistently ranks first in the majority of graphs, regardless of the type of edge weight. As a result, it achieves the lowest mean ranking position, as denoted by the last column.

There is a strong competition between BMC and EXC for the second fastest approach. For schema-based syntactic weights, EXC ranks first more often than BMC, despite its additional, time-consuming reciprocity check, due to its higher similarity threshold. As a result, its mean ranking position is slightly higher than BMC. This is reversed in all other types of similarity scores, even in the case of the semantic ones, where EXC still ranks first more often that BMC. The reason is that it also appears more often in the lowest ranking positions, unlike BMC, which is more robust, rarely dropping below the fourth place.

On the other extreme lie BAH and KRC. The former ranks last in practically all cases, while the latter dominates the sixth and seventh ranking positions, regardless of the weight type. None of them ranks first in any case.

The relative time efficiency of the remaining algorithms depends largely on the type of similarity scores. UMC is quite fast for schema-based syntactic weights, followed by RSR, while RCA is on par with the second slowest approach, KRC. For the schema-agnostic syntactic weights, RSR takes a minor lead over UMC, while RCA remains slower, but increases its distance from KRC. These patterns are reinforced in the case of schema-based semantic weights. For their schema-agnostic counterparts, RSR remains the fastest among the three algorithms, with RCA outperforming UMC.

Overall, the fastest algorithm typically is CNC, *followed in close distance by* BMC *and* EXC.

Regarding **QT(2)**, there are two main factors that affect the reported run-times: (i) the time complexity of the algorithms, and (ii) the similarity threshold used for pruning the search space. Regarding the first factor, we observe that the run-times in Figs. 7 and 8 generally verify the time complexities described in Sect. 3. With $O(m)$, CNC and BMC are the fastest ones, followed by EXC with $O(n\,m)$, RSR with $O(n\,m + n \log n)$, 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, as we discussed in Sect. 5.

Equally important is the effect of the similarity thresholds: the higher their optimal value (i.e., the one maximizing $F1$), the fewer the edges retained in the similarity graph, reducing the run-time. The optimal similarity threshold depends mostly on the type of the edge weights and the similarity graph at hand, as we explain in the threshold analysis in Sect. 6.4. This means that the relative time efficiency of algorithms with the same theoretical complexity should be

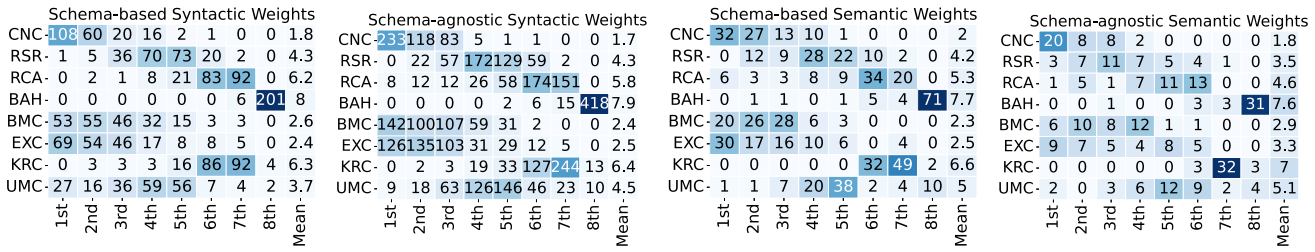

Fig. 9 Heatmaps summarizing the ranking positions per algorithm for each type of edge weights. Lower is better, i.e., the 1st ranking position corresponds to the fastest algorithm

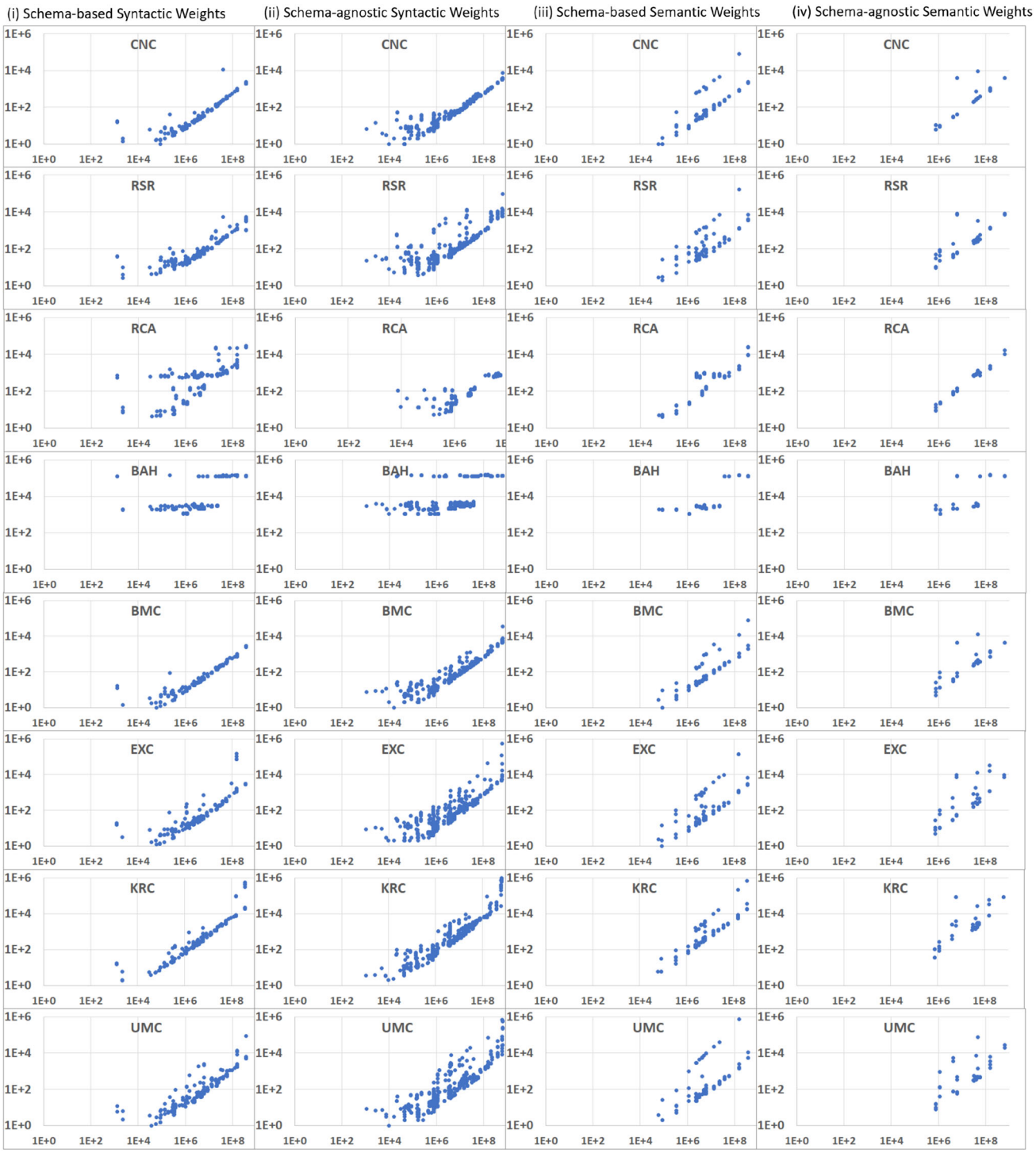

Fig. 10 Scalability analysis of all algorithms over all similarity graphs with (i) schema-based syntactic, (ii) schema-agnostic syntactic, (iii) schema-based semantic and (iv) schema-agnostic semantic edge weights. The horizontal axis corresponds to the number of edges in the similarity graphs and the vertical one to the run-time in milliseconds (maximum value = 16.7 min)

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 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 data sets 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 also accounts for the relative run-times between the same algorithm over different types of edge weights. For example, EXC is 5.5 times slower over the schema-agnostic syntactic weights of D_{ID} than their schema-based counterparts (4 vs. 22 s, on average), even though the former involve just 25% more edges than the latter, as reported in Table 2. 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 6 times when comparing the schema-based with the schema-agnostic syntactic weights (4 vs. 25 s, on average), because its average optimal threshold drops from 0.481 to 0.110.

Overall, the (relative) time efficiency of the algorithms is determined by their time complexity and the similarity threshold they use in every input graph.

To answer **QT(3)**, we examine two aspects of scalability:

1. Edge scalability, where the size of the input graphs (i.e., the number of their edges) increases by orders of magnitude, while the portion of matching edges remains practically stable, and
2. Node scalability, where the order of input graphs (i.e., the number of their nodes) increases by orders of magnitude, while the portion of matching nodes remains practically stable.

We delve into each aspect of scalability below.

6.2.1 Edge scalability

To examine the edge scalability, we use the similarity graphs of Table 2, which stem from the datasets in Table 1. Their size increases by four orders of magnitude, from 10^4 to 10^8, while the number of matching edges remains practically stable to 10^3 across most datasets. The only exceptions are the smallest dataset, D_{RE}, where the matching edges drop to 10^3, and the largest one, D_{ID}, where they raise to 10^4.

Figure 10 presents the edge scalability analysis of every algorithm over these 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, 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, EXC and UMC 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, namely 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 records in the input data sets, i.e., $O(|V_1|\ |V_2|)$. As a result, different similarity graphs from the same data set yield similar run-times that coincide in the diagrams of Fig. 10. 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.

On the whole, these experiments suggest that with the exception of BAH, *all algorithms typically scale sublinearly as the size of similarity graphs increases by orders of magnitude.*

6.2.2 Node scalability

The goal of this analysis is to examine how the run-time of all algorithms evolves as the order of similarity graphs increases from a few thousand to a few million records. To this end, we employ a large Clean-Clean ER dataset with a complete ground truth, i.e., where the relations between all pairs of records are known: the DBpedia dataset, which matches records from two versions of English DBpedia Infoboxes,[11] namely DBpedia 3.0rc, which dates back to October 2007 and contains 1.2 million records, and DBpedia 3.4, which was published in October 2009 and contains 2.2 million records. This dataset has been extensively used in the prior work [45, 48, 49].

Applying all similarity functions of Fig. 2 to the DBpedia dataset is quite challenging, given the time and space requirements for computing and storing the weight for the 2.6 trillion edges of the complete similarity graph. For this reason, we first converted every record into a pre-trained, schema-agnostic fastText embedding vector (we did not use S-T5, because it is an order of magnitude slower). Then, we indexed all record vectors of the largest constituent dataset, i.e., DBpedia 3.4, using FAISS, one of the fastest algorithms for approximate nearest neighbor search [2]. Next, every record of the smallest dataset, i.e., DBpedia 3.0rc, was used as a query to FAISS, retrieving the 10 record vectors from DBpedia 3.4 that have the highest cosine similarity with the query vector. This was repeated 10 times with an increasing number of queries in order to create 10 subsets with

[11] https://www.dbpedia.org.

Table 4 The similarity graphs used in the node scalability analysis, with $|V_1|$ (resp. $|V_2|$) denoting the number of entities from DBPedia 3.0rc (resp. DBpedia 3.4)

	$\|V_1\|$	$\|V_2\|$	Total nodes	$\|E\|$	Matching edges
S_1	58,194	32,903	91,097	77,273	4071
S_2	178,468	97,032	275,500	310,581	16,545
S_3	315,437	176,300	491,737	702,075	37,059
S_4	451,794	265,351	717,145	1,244,226	66,376
S_5	581,217	362,282	943,499	1,941,693	103,418
S_6	706,526	463,321	1,169,847	2,789,577	149,200
S_7	828,989	569,933	1,398,922	3,779,819	203,076
S_8	949,013	679,773	1,628,786	4,942,050	265,784
S_9	1,068,543	792,680	1,861,223	6,257,856	335,710
S_{10}	1,190,733	1,401,747	2,592,480	11,907,330	640,076

increasing number of nodes, but the same portion of matching records. The technical characteristics of the resulting subsets appear in Table 4.

We observe that in every subset, ∼5.3% of all edges connect matching records, while the total number of nodes increases by ∼220,000 from subset to subset. The only exception is the final subset, S_{10}, which contains ∼730,000 more records/nodes than S_9, because its new query records from DBpedia 3.0rc share very few nearest neighbors from DBpedia 3.4 with the rest of the queries. The overlap of nearest neighbors between the existing and the new queries in every subset also accounts for the increase in the number of edges per subset, which consistently diminishes: from 4× between S_2 and S_1 to 1.9× between S_{10} and S_9.

Given that it is quite challenging to optimize the similarity threshold of every algorithm on every subset, we performed the node scalability experiments as follows: First, we fine-tuned every algorithm on the smallest subset, S_1. The similarity threshold that maximizes F1 is 0.85 for RCA and KRC, 0.90 for EXC and UMC and 0.95 for CNC, RSR, BAH and BMC. Using these thresholds, we applied each algorithm on all subsets 10 times and took the average run-time. We also measured effectiveness in terms of precision, recall and F-measure. The results are reported in Fig. 11. Note the logarithmic scale in the vertical axis of the leftmost diagram.

Regarding run-time, we observe that RCA is consistently the slowest algorithm by far, requiring ∼3 and ∼21 h for S_1 and S_9, respectively (we terminated its execution on S_{10}, because it took more than 24 h). This poor time efficiency should be expected, as it is the only algorithm, whose time complexity is determined by the Cartesian product of the number of nodes in each partition, as explained above, in response to QT(2). Nevertheless, RCA scales sublinearly with respect to both the size and the order of the input graphs: the number of nodes (resp. edges) increases by 20 (resp. 81) times from S_1 and S_9, but RCA's run-time raises by less than 7 times.

The second slowest algorithm over the largest subsets is RSR. The reason is that in graphs of large order, its time complexity is dominated by the sorting of nodes in decreasing weight (see line 7 in Algorithm 2). Note that the cost of this operation is negligible for the graphs in Table 2, where the number of edges is much higher than the number of nodes. Its run-time increases in proportion to $(|V_1| + |V_2|)|E|$, which is in line with its theoretical time complexity (see Sect. 3).

BAH exhibits the same run-time across all subsets, which is equal to time limit of 2 min. Even though this time is much higher than all other algorithms across all subsets, BAH exhibits very low effectiveness, as demonstrated by its precision, which consistently remains lower than 0.12, and its poor recall, which fluctuates around 0.01. Inevitably, its $F1$ is close to zero in all cases.

The next slowest algorithm is KRC, whose time complexity is dominated by the iteration over all nodes and the sorting

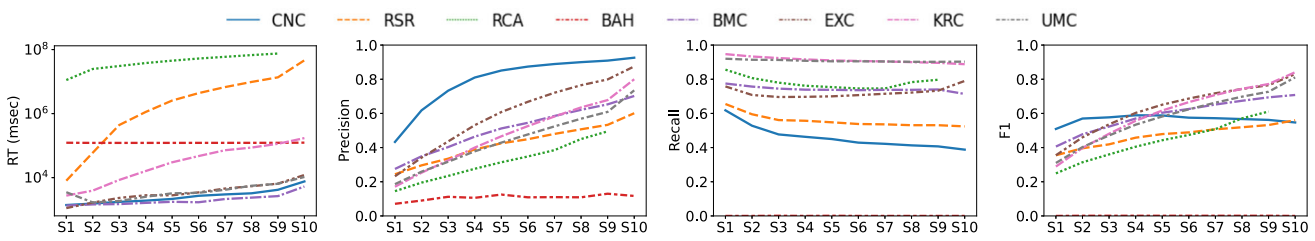

Fig. 11 Node scalability of all algorithms over the subsets in Table 4 with respect to run-time, precision, recall and F-Measure

of all edges, as explained in Sect. 3. This means that its execution time is determined both by the size and the order of the input similarity graphs, $|V_1|+|V_2|$ and $|E|$, respectively. Given that KRC's run-time increases by 60 times from S_1 to S_{10}, it scales superlinearly with respect to $|V_1|+|V_2|$, which increases by 28 times, and sublinearly with respect to $|E|$, which increases by 154 times. These patterns should be attributed to the similarity threshold used by KRC, which prunes most of the edges, but does not affect the processed nodes (i.e., KRC goes through all of them).

The rest of the algorithms exhibit sublinear node scalability, processing even the largest subset within 12 s. More specifically, the run-time of CNC, BMC, EXC and UMC raise by just 5.5, 4, 11 and 3 times, respectively, from S_1 to S_{10}. The fastest algorithm is consistently BMC, followed closely by CNC. Both use a higher similarity threshold than EXC and UMC (0.95 vs. 0.90), while having a linear time complexity with respect to $|E|$ (and $|V_1|+|V_2|$ for CNC). EXC and UMC exhibit practically identical run-times in all cases, despite their slightly different time complexities, because they process very few edges, after pruning the similarity graph with their high threshold.

Regarding effectiveness, we observe that precision raises significantly as the size and order of the input similarity graphs increases. Comparing S_1 with S_{10}, precision actually increases by 2× (CNC) to 5× (KRC), except for BAH, where it raises by 2/3. As expected, the highest precision corresponds to CNC, with EXC and BMC ranking second and third respectively—except for the three largest subsets, where KRC ranks third. The distance between CNC and EXC is fairly high for the smallest datasets, but decreases substantially (to just 5%) over the smallest one.

In terms of recall, most algorithms exhibit quite stable performance. The difference between the highest and the lowest recall across all subsets is less than 13% for all algorithms (just 2% for UMC). The only exceptions are CNC and RCA, whose recall gradually decreases from S_1 to S_{10} by 37% and 20%, respectively. The highest recall is consistently achieved by KRC, with UMC in the second place; both remain consistently above 0.9 across all subsets.

The combined patterns of recall and precision suggest that all algorithms detect a relatively stable and high portion of the matching edges across all subsets, while the portion of false positives diminishes as the size and order of the similarity graphs increase. In other words, the accuracy of all algorithms increases as more connections between the input records are taken into account. As a result, the $F1$ of all algorithms increases substantially, when comparing the smallest with the largest subsets. For CNC, the increase is just 7.5%, but raises to 57.8% and 74.4% for RSR and BMC, respectively, and to ∼2.5× for the rest of the algorithms. The highest $F1$ across all algorithms is achieved by CNC over the three smallest subsets, EXC over the next four subsets (S_4–S_7), and KRC of the three largest ones.

Overall, only RSR, RCA *and* BAH *exhibit poor time efficiency and/or poor effectiveness in the node scalability settings. The rest of the algorithms scale quite well, with* EXC *and* UMC *offering the best balance between* $F1$ *and run-time in most cases.*

6.3 Trade-off between *F*1 and run-time

We now examine the best trade-off that is achieved on average by all combinations of algorithms and types of edge weights across all data sets. To this end, Fig. 12 contains one diagram per data set, excluding the combinations including BAH, as their average performance consistently underperforms with respect to $F1$ and/or run-time. 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.

Starting with D_{RE}, 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 high effectiveness should be attributed to the relatively clean values of names and phones for the duplicate records and the high time efficiency to the lack of attribute values for most non-matching records. 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 2. 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 an average run-time of 4 ms). This combination practically dominates all others. Only CNC is significantly faster (run-time of 3 ms), but its $F1$ (0.756) is significantly lower.

In D_{AB}, we observe that the best performance clearly corresponds to the combination of UMC with schema-agnostic syntactic weights, which achieves the highest macro-averaged $F1$ (0.738). The next best combinations involve KRC or UMC with other types of edge weights, but reduce the average $F1$ by 10% for a similar run-time. The high performance of UMC (and KRC) should be attributed to the balanced nature of D_{AB}, as there is a 1–1 match between the records of its constituent datasets. The high performance of the schema-agnostic syntactic similarities stems from the long textual values of D_{AB}, which abound in terminologies (e.g., product names), thus hampering the performance of the semantic similarities. Note that BAH exhibits high effectiveness, too, but its run-time is ∼15 times higher than UMC.

In D_{AG}, the best $F1$ is achieved by the schema-based weights, as the title suffices for identifying the matching products. The semantic weights achieve significantly higher

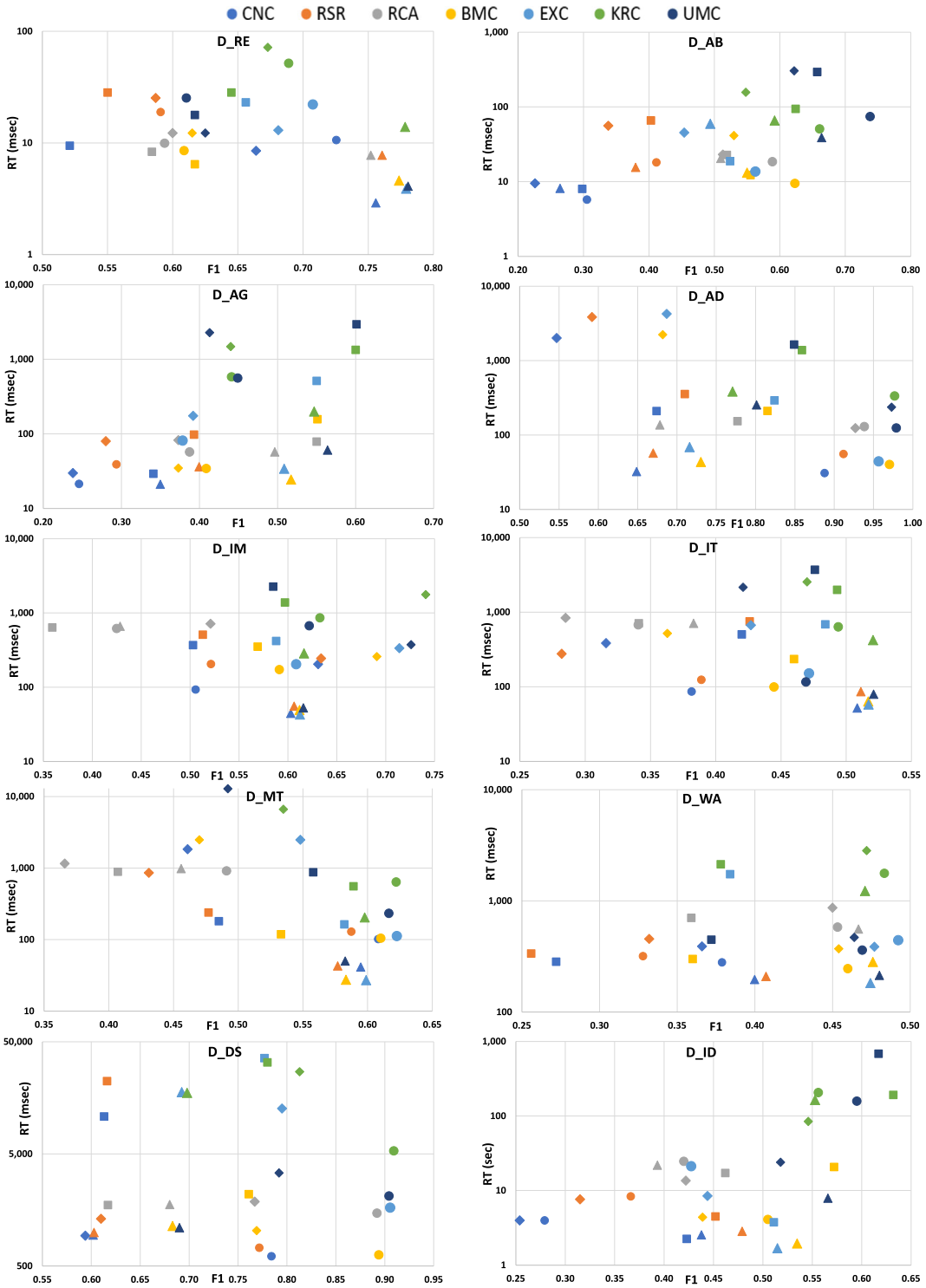

Fig. 12 Scatter plots of the average $F1$ (on the horizontal axis) and the average run-time (on the vertical axis) per algorithm and type of edge weights across all data sets. Every algorithm corresponds to a different color, as explained in the legend, and every type of input to a different shape: circle 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. Note the logarithmic scale in the vertical axis of all diagrams and that D_{ID} reports seconds instead of milliseconds, unlike all other cases (color figure online)

F1 than the syntactic ones, at the cost of much higher run-time. This applies to almost all algorithms, except for CNC and RSR, where the two weight types exhibit practically equivalent F1. The reason is that the semantic similarity functions assign high or just non-zero scores to most pairs of records, as suggested by the large average size of the resulting graphs in Table 2. Hence, they suffer from poor discriminativeness, requiring low similarity thresholds, i.e., they process many more edges, in order to achieve high effectiveness. In contrast, the syntactic similarity functions sacrifice some matching records, at the cost of lower effectiveness, but excel in time efficiency, due to the lower graph sizes. Despite being an one-sided dataset, the best F1 overall is achieved by KRC and UMC in combination with schema-based semantic weights. Yet, UMC with schema-based syntactic weights has lower F1 by 6.1%, while reducing the run-time by 22× and 49×, respectively. BMC and EXC further reduce the run-time almost by 50%, though at the cost of another significant decrease in $F1$ by <10%.

D_{DA} is dominated by the schema-agnostic inputs, due to the noise in the form of misplaced attribute values (e.g., the author of a publication is added in its title). This type of error cannot be addressed by schema-based weights, thus reducing their effectiveness significantly. In contrast, the schema-agnostic weights tackle this noise inherently, as they take into account the entire textual information per record. Given that D_{DA} is a balanced dataset, it is no surprise that UMC and KRC exhibit the highest average $F1$ (0.98), though at the cost of high run-time, due to the low similarity thresholds they employ, which result in large similarity graphs. Both algorithms are combined with schema-agnostic syntactic weights. UMC, though, achieves equally high performance with schema-agnostic semantic weights, too. Despite the higher threshold, though, these weights double the run-time, because they yield much larger similarity graphs. As shown in Table 2, they actually connect every pair of records with positive weights, unlike their syntactic counterparts. The best trade-off is achieved by BMC, which significantly reduces the run-time by at least 2/3 (due to its higher similarity threshold) for a negligible reduction to $F1$ (1%). BMC also dominates EXC, which is slightly slower for slightly lower F1.

D_{IM} constitutes a highly noisy data set, with many missing values in all attributes. As a result, all schema-based weights exhibit low effectiveness. Yet, the schema-agnostic ones consider all attribute values per record, with the semantic weights leveraging the contextual knowledge offered by fastText and SentenceBERT pre-trained embeddings. As a result, the schema-agnostic semantic weights achieve the highest $F1$, outperforming their syntactic counterparts by at least 17% across all algorithms. KRC is actually the top performing one in terms of effectiveness ($F1$=0.74), but UMC and EXC offer a better trade-off: $F1$ drops to ∼0.72, while the run-time is reduced by ∼5 times. The high time efficiency of the latter algorithms stems from the higher similarity threshold they use. For this reason, UMC is also the only algorithm that is faster with the semantic than the syntactic schema-agnostic weights.

The data sets D_{IT} and D_{MT} share similar levels and forms of noise with D_{IM}. However, the performance of schema-agnostic weights, especially the semantic ones, is much lower in both cases, due to the TVDB records that are included in both data sets, but not in D_{IM}. As a result, in D_{IT} the schema-based syntactic weights dominate all others, with the $F1$ of all algorithms (except RCA) confined in 0.51 and 0.52. The best trade-off is thus achieved by the fastest ones, namely CNC ($F1$ = 0.51, run-time of 52 ms) and EXC ($F1$ = 0.52, run-time of 58 ms). Note that the very low run-time should be attributed to the very small graph sizes (see Table 2) and the very high similarity thresholds.

Similarly, in D_{MT}, the schema-based syntactic weights achieve relatively high $F1$ (∼0.6) for very low run-time (≪100 msec), due to very small input graphs and the relatively high similarity thresholds. The schema-agnostic syntactic weights offer slightly higher $F1$ (∼0.62) for significantly higher run-time (∼100 ms), due to the significantly larger graphs and the smaller similarity thresholds. Among their combinations, the optimal choice is EXC, which achieves the highest macro-average $F1$ ($F1$ = 0.623) for a reasonable run-time (112 ms). KRC exhibits the same $F1$, but its run-time is almost 6 times higher.

D_{WA} constitutes a highly noisy data set, which restricts the $F1$ of all combinations below 0.5. The two types of syntactic weights are competing for the best performance with the schema-agnostic semantic types. Being a scarce dataset, the top performing algorithms are BMC, EXC, KRC and UMC. For these algorithms, the schema-based syntactic weights trade slightly lower effectiveness than both types of schema-agnostic weights for significantly lower run-times. For each type of weights, the best balance is offered by EXC: with the schema-based syntactic weights it achieves $F1$ = 0.474 and run-time of 182 ms, dominating the schema-agnostic semantic ones (almost the same $F1$, for double run-time), while the schema-agnostic syntactic ones achieve $F1$ = 0.492 with run-time of 443 ms. The final choice depends on the requirements of the application at hand.

D_{DS} is another bibliographic data set with noise in the form of misplaced values, similar to D_{DA}. As a result, the best performance is achieved by schema-agnostic syntactic weights, which consistently outperform the schema-based ones. In most cases, the schema-agnostic semantic weights outperform their schema-based counterparts, too, but lag behind the corresponding syntactic weights, due to the author names and the terminology that abounds in the publication titles. The combination of BMC with schema-agnostic syntactic weights dominates all other combinations, as its $F1$ is 1.5% lower than the maximum one (0.894 vs. 0.909 for KRC

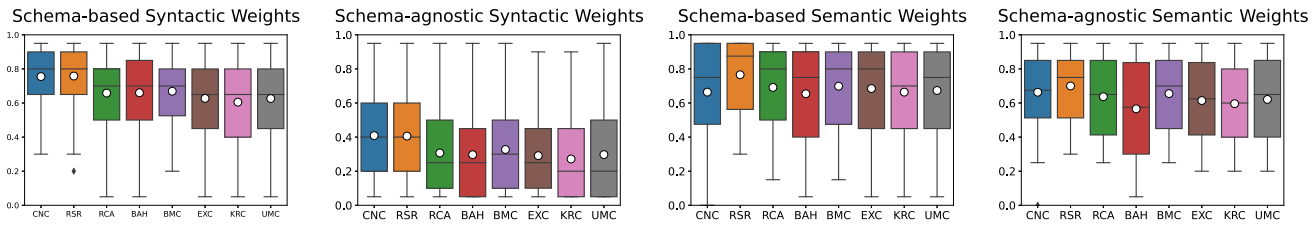

Fig. 13 The distribution of similarity thresholds per type of weights. White circles indicate the mean threshold in each case

with the same weights), while achieving the second lowest run-time (627 vs. 611 msec for CNC with the same weights). This is because the schema-agnostic syntactic graphs are by far the smallest ones for this data set (see Table 2), while BMC uses the highest average similarity threshold among all top-performing algorithms.

Finally, D_{ID} constitutes a rather noisy data set with one of the highest portion of missing values. $F1$ remains below 0.65 in all cases, with the top performance corresponding to KRC and UMC, because of the relatively balanced number of matching records in V_1 and V_2. Both algorithms perform better in combination with schema-based semantic weights, because the contextual information they provide is crucial for going beyond the mere syntactic similarity, allowing for distinguishing between the matching and the non-matching pairs of records. Among the two algorithms, KRC clearly outperforms UMC in all respects: its $F1$ is higher by 2.4%, while being faster by 3.6 times.

Overall, the best combination of bipartite graph matching algorithms and type of edge weights depends on the data sets at hand and the type of noise it incorporates. On average, though, UMC *in combination with syntactic similarities (mostly schema-agnostic ones) is consistently close to the best trade-off between $F1$ and run-time, if not the best one.*

6.4 Similarity threshold analysis

As explained above, in the response to **QT(2)**, the similarity threshold constitutes the most important configuration parameter for the effectiveness and time efficiency of all bipartite graph matching algorithms. It is crucial, therefore, to understand how easily this parameter can be fine-tuned a-priori, examining the main factors that determine its optimal value. To this end, Fig. 13 presents the distribution of similarity threshold per algorithm and type of edge weights.

We observe different patterns for each type of edge weights. More specifically, the schema-based syntactic inputs yield relatively high thresholds: the average values fluctuate between 0.61 and 0.67 for all algorithms—except for CNC and RSR, whose mean thresholds amount to 0.76. The median thresholds are slightly higher, between 0.65 and 0.8. Yet, the variance is also high, as the standard deviation is consistently higher than 0.16. The reason is that the similarity thresholds of all algorithms cover the entire space in [0.05, 0.95], as suggested by the minimum and maximum values. For most algorithms, though, the first and third quartiles significantly restrict the optimal values to the range 0.4/0.5−0.8 or 0.65−0.90.

Regarding the schema-agnostic syntactic weights, we observe that the variance is higher than their schema-based counterparts, even though the average and median values are much lower, even by 50%. The reason is that the similarity thresholds still cover the entire space in [0.05, 0.95], with the interquartile range increasing from 0.25 or 0.3 to 0.4 for practically all algorithms. This means that this type of weights yields a larger diversity of graphs, which hinders the configuration of the similarity thresholds.

Among the semantic inputs, the schema-based weights share similar patterns with their syntactic counterparts. The average thresholds are quite high for all algorithms, fluctuating between 0.65 and 0.77, while the median ones are consistently higher, ranging from 0.70 to 87. The variance is slightly higher than the respective syntactic weights, while the interquartile range is significantly higher, fluctuating between 0.4 and 0.5; the reason is that the first quartile is lower, while the third one is higher. These patterns suggest that fine-tuning any algorithm on the schema-based semantic weights is more challenging than the syntactic weights (of any type).

Finally, the schema-agnostic semantic graphs exhibit patterns that bear little resemblance to those arising from their syntactic counterparts. The average thresholds are much higher for all algorithms, but lower than the schema-based weights of any type, fluctuating between 0.57 and 0.70. The same applies to the first and third quartiles, which fluctuate in [0.30, 0.51] and [0.80, 0.85], respectively. The only common characteristics with the respective syntactic weights are the interquartile range, which ranges from 0.35 to 0.45 for most algorithms, and the standard deviation, which is around ∼0.22 in both cases. Overall, the configuration of these weights is easier than the schema-based semantic ones, but harder than the schema-based syntactic ones.

Note that the relative order of the algorithms with respect to the average thresholds remains the same in both types of syntactic weights: CNC and RSR exhibit the highest thresholds, with KRC lying at the other extreme, followed in close

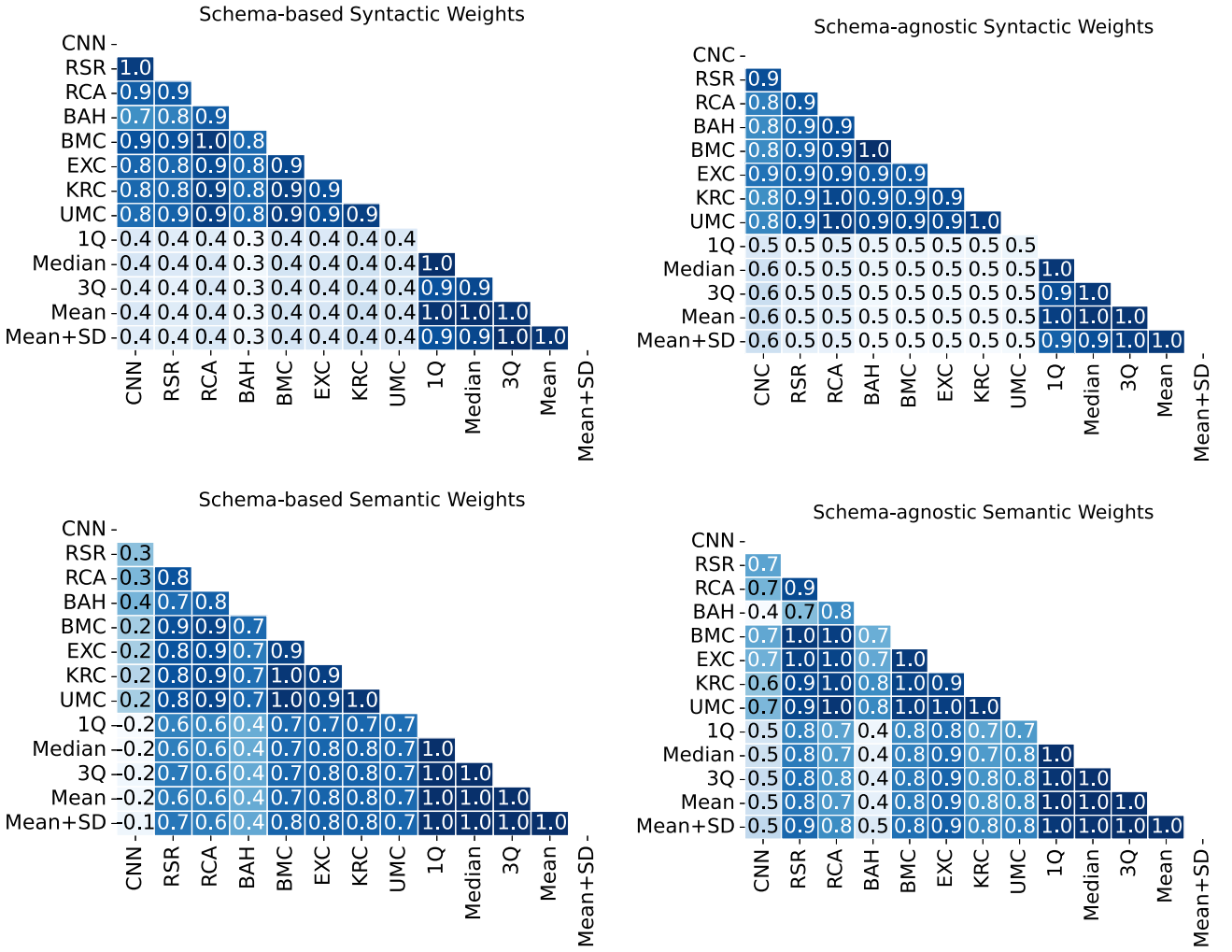

Fig. 14 Pearson correlation between the similarity thresholds of every algorithm as well as between the similarity thresholds and the descriptive statistics of the edge weights distribution

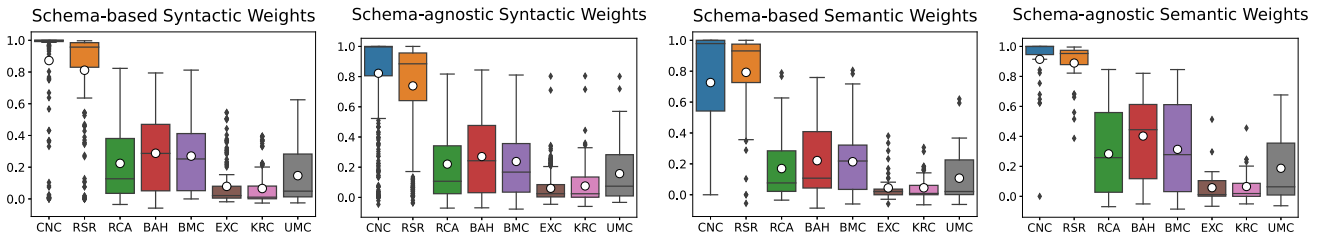

Fig. 15 The deviation in $F1$ between the fine-tuned threshold and the automatically configured one, i.e., Mean + SD, across all algorithms and types of edge weights

distance by EXC and UMC. In the case of semantic inputs, there are no clear patterns, except for RSR and BAH, which consistently yield the highest and the lowest average threshold, respectively. The difference between the rest of the algorithms is much lower than 0.05 in most cases.

On the whole, these experiments suggest that the optimal similarity threshold per algorithm depends mostly on the type of edge weights as well as on the distribution of edge weights in the given similarity graph.

6.4.1 Automatic threshold configuration

The goal of this section is to examine how easy it is to fine-tune the threshold of these algorithms in practice. To this end, Fig. 14 presents the Pearson correlations for each pair of

algorithms across the four types of edge weights. We observe that this correlation is highly positive, taking values well above 0.8 in the vast majority of cases, especially for the syntactic weights. There are few exceptions in the case of semantic weights: BAH's correlations fluctuate between 0.7 and 0.8, while CNC typically requires much higher thresholds than the rest of the algorithms. Consequently, its Pearson correlation remains positive, but significantly lower, in the range [0.2, 0.7].

These patterns mean that knowing the optimal threshold for a particular algorithm over a specific data set provides strong indications for fine-tuning the rest of the algorithms over the same data set. In other words, the optimal threshold of each algorithm typically depends on the similarity graph at hand and the distribution of its edge weights.

To verify this statement, Fig. 14 also presents the Pearson correlation between the optimal threshold per algorithm and the descriptive statistics of the edge weight distribution in the similarity graphs: the main quartiles, i.e., the first, second (i.e., median) and third one, the mean edge weight as well as the mean plus the standard deviation (Mean + SD). We observe highly positive correlations between these statistics and the similarity thresholds in all cases.

In more detail, the lowest correlation scores, on average, correspond to the schema-based syntactic weights, where they fluctuate between 0.35 and 0.44 for all algorithms, but BAH, which restricts them to [0.29, 0.33]. For schema-agnostic syntactic weights, they range from 0.47 to 0.59. For the semantic weights, the correlations are much higher, at least 0.59 (0.72) for the schema-based (schema-agnostic) ones. This applies to all algorithms, but: (i) BAH, whose stochastic nature restricts the correlations to [0.40, 0.47], and (ii) CNC, which consistently requires much higher thresholds than all algorithms, because the semantic similarity functions yield larger graphs, on average, associating almost all record/node pairs with positive edge weights. As a result, the correlations are slightly negative, in [−0.20, −0.15], for CNC over the schema-based semantic weights, while for the schema-agnostic ones, they are confined in [0.50, 0.55], which is quite high, but significantly lower than the rest of the algorithms.

Among the five descriptive statistics, Mean + SD yields the highest correlation, albeit to a minor extent, for most combinations of algorithm and type of edge weights. This should be expected, as the first quartile, the median and the mean similarity scores are usually much lower than the optimal threshold, while the third quartile is much higher. Instead, Mean + SD exhibits the lowest deviation from the absolute value of fine-tuned similarity thresholds. Hence, Mean + SD seems a natural choice for automatically configuring each algorithm in any given similarity graph.

We tested the performance of Mean + SD in all 769 graphs that are considered in our experimental analysis. The outcomes are presented in Fig. 15 with respect to *F1 deviation*, which is defined as $1 - F1_{\text{auto}}/F1_{\text{max}}$, where $F1_{\text{auto}}$ is the F-measure achieved when using Mean + SD as the similarity threshold, and $F1_{\text{max}}$ is the F-measure corresponding to the fine-tuned similarity threshold.

Based on the distribution of F1 deviation across the four types of edge weights, we can distinguish the eight algorithms into three groups. The first one includes CNC and RSR, both of which are quite sensitive to the similarity threshold, exhibiting extremely poor performance when configured with Mean + SD. The reason is that both algorithms ignore clusters that include more than two nodes/records. Hence, retaining few false positive edges leads to a drastic reduction in precision, which also affects $F1$ to a significant extent. More specifically, the transitive closure of CNC amplifies the error introduced by a few false positive edges, yielding an output that is dominated by clusters with multiple entities. All these clusters are ignored, minimizing precision and recall and, thus, f-measure. The same applies to RSR. Of course, there are similarity graphs where the performance of both algorithms remains intact (i.e., F1 deviation is close to zero), but in the vast majority of graphs, the reduction in $F1$ exceeds 80%.

The second group of algorithms includes EXC and KRC, which are quite robust with respect to the similarity threshold. Their median, mean and third quartile F1 deviation are well below 0.1 in every type of edge weights. This means that replacing their optimal similarity threshold with Mean + SD reduces the $F1$ to a minor extent, which is less than 10% in the vast majority of cases. This should be attributed to the fact that both algorithms rely heavily on the reciprocity of edges with the highest weight per node. As a result, a false positive edge merely reduces precision, by adding to the end result matches that are discarded by the fine-tuned similarity threshold.

The third group includes the rest of the algorithms, namely RCA, BAH, BMC and UMC, whose performance fluctuates between the two extremes of the other two groups. The best performance among them is achieved by UMC, as its median F1 deviation remains below 0.75 across all edge types. However, its third quartile exceeds 0.20 in all cases, while its mean F1 deviation fluctuates between 0.11 and 0.19 for the schema-based and the schema-agnostic semantic weights, respectively. These patterns can be explained by the type of similarity graph: in balanced graphs, UMC is quite robust to the similarity threshold, because it operates similarly to EXC, associating every node with its most similar one from the other partition; in scarce and one-side graphs, though, a fine-tuned threshold is necessary, otherwise UMC provides matches for most nodes, even the non-matching ones—in contrast, EXC leaves unmatched the nodes that are not reciprocally most similar.

Table 5 Comparison to state-of-the-art unsupervised matching methods with respect to F-measure

	ZeroER	UMC
D_{AB}	0.52	0.95 (character bi-grams, $t = 0.35$)
D_{AG}	0.48	0.60 (token bi-grams, $t = 0.05$)
D_{DA}	0.96	0.99 (token uni-grams, $t = 0.40$)
D_{DS}	0.86	0.94 (character four-grams, $t = 0.35$)

UMC is exclusively combined with schema-agnostic TF-IDF weights and cosine similarity

The remaining algorithms exhibit higher sensitivity to the configuration of the similarity threshold, as their mean F1 deviation exceeds 0.2 in practically all cases. The highest interquartile range in F1 deviation typically corresponds to BAH, due to its stochastic nature, which yields an unstable performance. BMC typically performs better than BAH, but worse than RCA, which is very close to UMC. Similar to UMC, BMC needs a fine-tuned similarity threshold in order to avoid associating records in scarce and one-sided graphs with false positives. RCA, on the other hand, needs a fine-tuned threshold in order to clean its output from clusters with very low similarity (see lines 29–31 in Algorithm 3).

On the whole, we can conclude that EXC *and* KRC *are quite robust to the threshold that prunes the input similarity graph, due to the reciprocity they require for their matching records.* UMC *is also robust in the case of balanced similarity graphs, but quite sensitive in scarce and one-sided cases.*

6.5 Comparison with the best matching methods

We now compare the performance achieved by bipartite graph matching algorithms with the top performing matching method that leverages unsupervised learning: ZeroER [63]. We consider the four common data sets: D_{AB}, D_{AG}, D_{DA} and D_{DS} (D_{RE} is a larger and noisier version of FZ in [34, 63], and thus not directly comparable). Table 5 reports the relative performance in terms of maximum $F1$ for ZeroER from [63].

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 data set. These settings do not necessarily correspond to the highest $F1$ across all algorithms and edge weights we have considered, but demonstrate the capabilities of bipartite graph matching when varying just two configuration parameters. The results are shown in Table 5.

We observe that UMC consistently achieves higher performance than ZeroER: its $F1$ is higher by 3%, 9%, 25%, and 83% over D_{DA}, D_{DS}, D_{AG} and D_{AB}, respectively. We can conclude, therefore, that *bipartite graph matching is capable of outperforming the most effective unsupervised matching algorithm in the literature.*

7 Conclusions

We draw the following important conclusions from our experiments:

1. The most effective 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/records.
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 data sets with syntactic weights, especially the schema-agnostic ones.
3. RSR is a fast algorithm that rarely achieves high effectiveness in the case of scarce data sets.
4. RCA is an efficient method that 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 data sets (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 data sets, 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 highest $F1$ over scarce and one-sided data sets, 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 efficiency/scalability.
8. KRC achieves very high or the highest effectiveness in most cases, especially over one-sided and scarce data sets. 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 data sets, 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.
10. Regarding the balance between effectiveness and time efficiency, the best combination of graph matching algorithm and type of edge weights is data set-specific. Yet,

coupling UMC with schema-agnostic syntactic similarities usually approximates (or even achieves) the best possible trade-off between $F1$ and run-time.

11. For all algorithms, the fine-tuning of their similarity threshold does not depend on their functionality, but on the type of edge weights and their distribution. As a result, for each similarity graph, similar thresholds are used by all algorithms. Relatively high thresholds are used in most cases, except for schema-agnostic syntactic weights. Most importantly, EXC and KRC can be automatically fine-tuned by setting as their similarity threshold the average edge weight plus its standard deviation. The same applies to UMC in similarity graphs with 1–1 matching between the nodes of the two partitions.
12. With proper configuration, the top-performing bipartite graph matching algorithms are competitive to the state-of-the-art matching algorithm that leverages unsupervised learning.

Our focus in this work has been on ER in an unsupervised setting where large enough training data is not available. Following the approach in [53], an interesting avenue for future work is a supervised learning system incorporating a variety of bipartite graph matching algorithms in a unified framework and effectively learning the right parameters for a given data set. We will also explore adaptive solutions that decide on the best algorithm based on the analysis of initial results. Finally, we will use F^* [24] as an additional effectiveness measure in order to identify more interesting patterns in the relative performance of the considered algorithms.

Acknowledgements 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. This work was partially funded by the EU project STELAR (Horizon Europe - Grant No. 101070122)

Funding Open access funding provided by HEAL-Link Greece.

References

1. Assi, A., Mcheick, H., Dhifli, W.: BIGMAT: a distributed affinity-preserving random walk strategy for instance matching on knowledge graphs. In: IEEE Big Data, pp. 1028–1033 (2019)
2. Aumüller, M., Bernhardsson, E., Faithfull, A.J.: Ann-benchmarks: a benchmarking tool for approximate nearest neighbor algorithms. Inf. Syst. **87**, 101374 (2020)
3. Binette, O., Steorts, R.C.: (Almost) all of entity resolution. Sci. Adv. **8**(12), eabi8021 (2022)
4. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Trans. Assoc. Comput. Linguist. **5**, 135–146 (2017)
5. Brunner, U., Stockinger, K.: Entity matching with transformer architectures—a step forward in data integration. In: EDBT, pp. 463–473 (2020)
6. Chapman, S.: Simmetrics: open source similarity measure library. http://sourceforge.net/projects/simmetrics/ (2007)
7. Christen, P.: Data Matching—Concepts and Techniques for Record Linkage, Entity Resolution, and Duplicate Detection. Data-Centric Systems and Applications. Springer, Berlin (2012)
8. Christen, P., Ranbaduge, T., Schnell, R.: Linking Sensitive Data. Springer, Heidelberg (2020)
9. Christophides, V., Efthymiou, V., Palpanas, T., Papadakis, G., Stefanidis, K.: An overview of end-to-end entity resolution for big data. ACM Comput. Surv. **53**(6), 127:1-127:42 (2021)
10. Dasgupta, S., Papadimitriou, C.H., Vazirani, U.V.: Algorithms. McGraw-Hill, New York (2008)
11. Demsar, J.: Statistical comparisons of classifiers over multiple data sets. J. Mach. Learn. Res. **7**, 1–30 (2006)
12. Devlin, J., Chang, M., Lee, K., Toutanova, K.: BERT: pre-training of deep bidirectional transformers for language understanding. In: NAACL-HLT, pp. 4171–4186 (2019)
13. Dong, X.L., Srivastava, D.: Big Data Integration. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, San Rafael (2015)
14. Draisbach, U., Christen, P., Naumann, F.: Transforming pairwise duplicates to entity clusters for high-quality duplicate detection. ACM J. Data Inf. Qual. **12**(1), 3:1-3:30 (2020)
15. Efthymiou, V., Papadakis, G., Stefanidis, K., Christophides, V.: Minoaner: schema-agnostic, non-iterative, massively parallel resolution of web entities. In: EDBT, pp. 373–384. OpenProceedings.org (2019)
16. Fellegi, I.P., Sunter, A.B.: A theory for record linkage. J. Am. Stat. Assoc. **64**(328), 1183–1210 (1969)
17. Fredman, M.L., Tarjan, R.E.: Fibonacci heaps and their uses in improved network optimization algorithms. J. ACM **34**(3), 596–615 (1987)
18. Gale, D., Shapley, L.S.: College admissions and the stability of marriage. Am. Math. Mon. **69**(1), 9–15 (1962)
19. Gemmell, J., Rubinstein, B.I.P., Chandra, A.K.: Improving entity resolution with global constraints. CoRR. arXiv:1108.6016 (2011)
20. Giannakopoulos, G., Karkaletsis, V., Vouros, G.A.: Summarization system evaluation revisited: N-gram graphs. ACM Trans. Speech Lang. Process. **5**(3), 5:1-5:39 (2008)
21. Giannakopoulos, G., Palpanas, T.: Content and type as orthogonal modeling features: a study on user interest awareness in entity subscription services. Int. J. Adv. Netw. Serv. **3**(2) (2010)
22. Gotoh, O.: An improved algorithm for matching biological sequences. J. Mol. Biol. **162**(3), 705–708 (1982)
23. Gutierrez, C., Sequeda, J.F.: Knowledge Graphs: A Tutorial on the History of Knowledge Graph's Main Ideas, pp. 3509–3510. Association for Computing Machinery (2020). https://doi.org/10.1145/3340531.3412176

24. Hand, D.J., Christen, P., Kirielle, N.: F*: an interpretable transformation of the f-measure. Mach. Learn. **110**(3), 451–456 (2021)
25. Hassanzadeh, O., Chiang, F., Miller, R.J., Lee, H.C.: Framework for evaluating clustering algorithms in duplicate detection. Proc. VLDB Endow. **2**(1), 1282–1293 (2009)
26. Herbold, S.: Autorank: a python package for automated ranking of classifiers. J. Open Source Softw. **5**(48), 2173 (2020)
27. Király, Z.: Linear time local approximation algorithm for maximum stable marriage. Algorithms **6**(3), 471–484 (2013)
28. Konda, P., Das, S., Doan, A., Ardalan, A., Ballard, J.R., Li, H., Panahi, F., Zhang, H., Naughton, J.F., Prasad, S., Krishnan, G., Deep, R., Raghavendra, V.: Magellan: toward building entity matching management systems. Proc. VLDB Endow. **9**(12), 1197–1208 (2016)
29. Köpcke, H., Thor, A., Rahm, E.: Evaluation of entity resolution approaches on real-world match problems. Proc. VLDB Endow. **3**(1), 484–493 (2010)
30. Kriege, N.M., Giscard, P., Bause, F., Wilson, R.C.: Computing optimal assignments in linear time for approximate graph matching. In: ICDM, pp. 349–358 (2019)
31. Kuhn, H.W., Yaw, B.: The Hungarian method for the assignment problem. Naval Res. Logist. Q. **2**, 83–97 (1955)
32. Kurtzberg, J.M.: On approximation methods for the assignment problem. J. ACM (JACM) **9**(4), 419–439 (1962)
33. Kusner, M.J., Sun, Y., Kolkin, N.I., Weinberger, K.Q.: From word embeddings to document distances. In: ICML, vol. 37, pp. 957–966 (2015)
34. Li, Y., Li, J., Suhara, Y., Doan, A., Tan, W.: Deep entity matching with pre-trained language models. Proc. VLDB Endow. **14**(1), 50–60 (2020)
35. Li, Y., Li, J., Suhara, Y., Wang, J., Hirota, W.: Deep entity matching: challenges and opportunities. ACM J. Data Inf. Qual. **13**(1), 1:1-1:17 (2021)
36. Lovasz, L., Plummer, M.D.: Matching theory. vol 367, American Mathematical Soc (2009)
37. Manning, C.D., Raghavan, P., Schütze, H.: Introduction to Information Retrieval. Cambridge University Press, Cambridge (2008)
38. Melnik, S., Garcia-Molina, H., Rahm, E.: Similarity flooding: a versatile graph matching algorithm and its application to schema matching. In: ICDE, pp. 117–128 (2002)
39. Mikolov, T., Sutskever, I., Chen, K., Corrado, G.S., Dean, J.: Distributed representations of words and phrases and their compositionality. In: NIPS, pp. 3111–3119 (2013)
40. Mudgal, S., Li, H., Rekatsinas, T., Doan, A., Park, Y., Krishnan, G., Deep, R., Arcaute, E., Raghavendra, V.: Deep learning for entity matching: a design space exploration. In: SIGMOD, pp. 19–34 (2018)
41. Nemenyi, P.: Distribution-Free Multiple Comparisons. Princeton University, Princeton (1963)
42. Obraczka, D., Schuchart, J., Rahm, E.: EAGER: embedding-assisted entity resolution for knowledge graphs. CoRR. arXiv:2101.06126 (2021)
43. Otto, B., Reichert, A.: Organizing master data management: findings from an expert survey. In: Proceedings of the 2010 ACM Symposium on Applied Computing (SAC), pp. 106–110 (2010). https://doi.org/10.1145/1774088.1774111
44. Papadakis, G., Giannakopoulos, G., Paliouras, G.: Graph vs. bag representation models for the topic classification of web documents. World Wide Web **19**(5), 887–920 (2016)
45. Papadakis, G., Ioannou, E., Niederée, C., Fankhauser, P.: Efficient entity resolution for large heterogeneous information spaces. In: WSDM, pp. 535–544 (2011)
46. Papadakis, G., Ioannou, E., Thanos, E., Palpanas, T.: The Four Generations of Entity Resolution. Synthesis Lectures on Data Management. Morgan and Claypool Publishers, San Rafael (2021)
47. Papadakis, G., Koutrika, G., Palpanas, T., Nejdl, W.: Meta-blocking: taking entity resolution to the next level. IEEE Trans. Knowl. Data Eng. **26**(8), 1946–1960 (2014). https://doi.org/10.1109/TKDE.2013.54
48. Papadakis, G., Mandilaras, G.M., Gagliardelli, L., Simonini, G., Thanos, E., Giannakopoulos, G., Bergamaschi, S., Palpanas, T., Koubarakis, M.: Three-dimensional entity resolution with JedAI. Inf. Syst. **93**, 101–565 (2020)
49. Papadakis, G., Papastefanatos, G., Koutrika, G.: Supervised meta-blocking. Proc. VLDB Endow. **7**(14), 1929–1940 (2014)
50. Papadakis, G., Skoutas, D., Thanos, E.: Blocking and filtering techniques for entity resolution: a survey. ACM Comput. Surv. **53**(2), 31:1-31:42 (2020). https://doi.org/10.1145/3377455
51. Pennington, J., Socher, R., Manning, C.D.: Glove: global vectors for word representation. In: EMNLP, pp. 1532–1543 (2014)
52. Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. J. Mach. Learn. Res. **21**(140), 1–67 (2020)
53. Reas, R., Ash, S., Barton, R., Borthwick, A.: Superpart: supervised graph partitioning for record linkage. In: IEEE International Conference on Data Mining, ICDM 2018, Singapore, November 17–20, 2018, pp. 387–396. IEEE Computer Society (2018). https://doi.org/10.1109/ICDM.2018.00054
54. Rüschendorf, L.: The Wasserstein distance and approximation theorems. Probab. Theory Relat. Fields **70**(1), 117–129 (1985)
55. Saeedi, A., Nentwig, M., Peukert, E., Rahm, E.: Scalable matching and clustering of entities with FAMER. Complex Syst. Inform. Model. Q. **16**, 61–83 (2018)
56. Saeedi, A., Peukert, E., Rahm, E.: Using link features for entity clustering in knowledge graphs. In: ESWC, Lecture Notes in Computer Science, vol. 10843, pp. 576–592. Springer (2018)
57. Schwartz, J., Steger, A., Weißl, A.: Fast algorithms for weighted bipartite matching. In: WEA, Lecture Notes in Computer Science, vol. 3503, pp. 476–487 (2005)
58. Wang, Y., Tong, Y., Long, C., Xu, P., Xu, K., Lv, W.: Adaptive dynamic bipartite graph matching: a reinforcement learning approach. In: ICDE, pp. 1478–1489 (2019)
59. Wang, Z., Sisman, B., Wei, H., Dong, X.L., Ji, S.: Cordel: a contrastive deep learning approach for entity linkage. In: ICDM (2020)
60. Watkins, C.J.C.H., Dayan, P.: Technical note q-learning. Mach. Learn. **8**, 279–292 (1992)
61. Wijaya, D.T., Bressan, S.: Ricochet: a family of unconstrained algorithms for graph clustering. In: International Conference on Database Systems for Advanced Applications, pp. 153–167. Springer (2009)
62. Winkler, W.E.: Overview of record linkage and current research directions. Technical Report, Bureau of the Census (2006)
63. Wu, R., Chaba, S., Sawlani, S., Chu, X., Thirumuruganathan, S.: Zeroer: entity resolution using zero labeled examples. In: SIGMOD, pp. 1149–1164 (2020)