Sorted Neighborhood for Schema-Free RDF Data

Mayank Kejriwal(✉) and Daniel P. Miranker

University of Texas at Austin, Austin, USA
{kejriwal,miranker}@cs.utexas.edu

Abstract. Entity Resolution (ER) concerns identifying pairs of entities that refer to the same underlying entity. To avoid $O(n^2)$ pairwise comparison of n entities, blocking methods are used. Sorted Neighborhood is an established blocking method for Relational Databases. It has not been applied to schema-free Resource Description Framework (RDF) data sources widely prevalent in the Linked Data ecosystem. This paper presents a Sorted Neighborhood workflow that may be applied to schema-free RDF data. The workflow is modular and makes minimal assumptions about its inputs. Empirical evaluations of the proposed algorithm on five real-world benchmarks demonstrate its utility compared to two state-of-the-art blocking baselines.

Keywords: Entity resolution · Sorted neighborhood · Schema-free RDF

1 Introduction

Entity Resolution (ER) is the problem of identifying pairs of entities across databases that refer to the same *underlying* entity. An example is illustrated in Fig. 1. The problem goes by many names in the data mining and knowledge discovery communities, examples being *deduplication* [5], *record linkage* [4], *link discovery* [15] and *coreference resolution* [13], to name just a few. Instances of the problem have been documented for structured [5], semistructured [17], as well as unstructured data models [13].

Given n entities and a boolean *link specification* function that determines whether two entities are equivalent, a naïve ER application would run in time $O(n^2)$. Scalability indicates a two-step approach [5]. The first step is called *blocking*. Blocking uses a many-many clustering function called a *blocking key* to assign entities in near-linear time into one or more *blocks*, where each block represents a cluster [4]. In the second *similarity* step, only entities sharing a block are paired and evaluated by the (expensive) link specification function [8,15].

In the Relational Database (RDB) community, a state-of-the-art blocking algorithm is *Sorted Neighborhood* (SN) [6]. The classic SN version accepts a rigidly structured tabular database as input, and sorts the table by using the blocking key as a *sorting key*. A window of constant size is then slid over the records, and records sharing a window are paired and become candidates for

F. Gandon et al. (Eds.): ESWC 2015, LNCS 9341, pp. 217–229, 2015.
DOI: 10.1007/978-3-319-25639-9_38

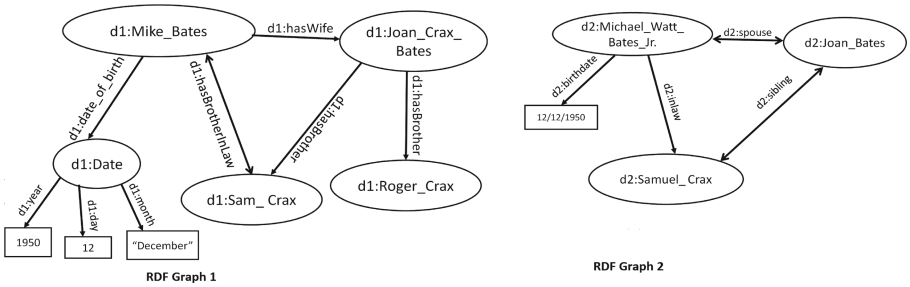

Fig. 1. An instance of the Entity Resolution (ER) problem on two RDF graphs. Various entities between the two graphs need to be interlinked using *owl:sameAs* edges

the second ER step. In several studies, the RDB version of SN was verified to have excellent theoretical run-time guarantees and good empirical performance [4,6]. To the best of our knowledge, an SN algorithm has never been proposed for *schema-free* RDF data sources despite its success. Such data sources are common on Linked Open Data (LOD), which has continued to grow since its inception in 2007 with just 12 datasets [20]. According to a recently concluded study, LOD currently contains over 1000 datasets and tens of billions of triples[1].

Typical blocking systems such as Silk and Limes in the Semantic Web community assume that the link specification function is known *a priori* [8,15]. Using the properties of the link specification function, these systems partition the space of entities into blocks. In contrast, Sorted Neighborhood does not assume any knowledge of the link specification function [6]. In practice, employing an *agnostic* blocking method such as Sorted Neighborhood enables an ER practitioner to *decouple* the two ER steps in terms of implementation and execution.

This paper presents a Sorted Neighborhood workflow that processes schema-free RDF data and accommodates a range of practical options (Sect. 3). The described algorithm is evaluated on five real-world test cases against two established baselines (Sect. 4). Preliminary results show that the method is a promising blocking procedure for schema-free RDF datasets.

2 Related Work

Entity Resolution is an old problem [14], and is methodologically surveyed by Elmagarmid et al. [5]. The blocking step is surveyed separately by Christen [4], who has also written a book synthesizing recent trends in ER research [3]. We note that traditionally, blocking has not been given as much importance as the second ER step, but this changed in the mid-90's, with large databases increasingly accessible over the Web [4]. Hernàndez and Stolfo published the Sorted Neighborhood work [6], and extended it to include parallel implementations [7]. Recent work has adapted the method to XML data, but under the assumption

[1] http://linkeddata.org.

that the XML data sources possess the *same schema*, possibly after a *schema matching* step [17]. To the best of our knowledge, the method has not been adapted yet to *schema-free* semi-structured data.

The advent of Linked Open Data has brought renewed focus on the RDF-based version of Entity Resolution, which is commonly denoted as *instance matching* or *link discovery* in the Linked Data community [15,18]. The fourth Linked Data principle states that data must not exist in silos, but must be interlinked to maximize value [2]. Linked Open Data (LOD) is currently known to contain over eight million entities [20], many of which are believed to refer to the same underlying entity. Silk and Limes are two popular ER frameworks that have sought to address the issue of discovering these entity pairs [8,15]. Another example of a Semantic Web-based ER system is RDF-AI, which offers customizable packages, similar to the proposed workflow [19]. The survey by Scharffe et al. provides more details on recently developed ER systems in the Linked Data community [18]. While these systems represent significant advances, the blocking methods that they employ are typically not agnostic of the link specification function. In contrast, the Sorted Neighborhood workflow proposed in this paper is completely independent of the second ER step, and operates on schema-free RDF data. Additionally, the proposed workflow offers options for both batch and online data access settings.

3 The Workflow

Figure 2 illustrates a Sorted Neighborhood workflow for schema-free RDF data. At a high level, the workflow accepts two inputs, namely the pair of RDF graphs G_1 and G_2 containing entities that need to be interlinked, as well as corresponding blocking keys B_1 and B_2. The final output is a set of entity-entity pairs (denoted as the *candidate set*), which is piped to the second ER step and evaluated by an unknown link specification function. As earlier noted, an advantage of Sorted Neighborhood is that it is agnostic of the link specification function.

3.1 Blocking Keys

The quality of the workflow in Fig. 2 depends directly on the quality of blocking keys B_1 and B_2. There are several methods in the literature on both defining and learning appropriate blocking keys for schema-free data [9,16]. One of the earliest solutions for this problem was to use a simple token-based distance measure (e.g. cosine similarity) to cluster entities, and to ignore all structural information [12]. This token-based algorithm, called *Canopy Clustering* [12], was found to have good performance on many test cases, but has been outperformed by more sophisticated learning algorithms in recent years [1,9].

Two classes of viable learning algorithms (for blocking keys) have emerged as state-of-the-art. The first is the Attribute Clustering (AC) algorithm [16]. AC is an unsupervised procedure that groups properties (or *attributes*) after

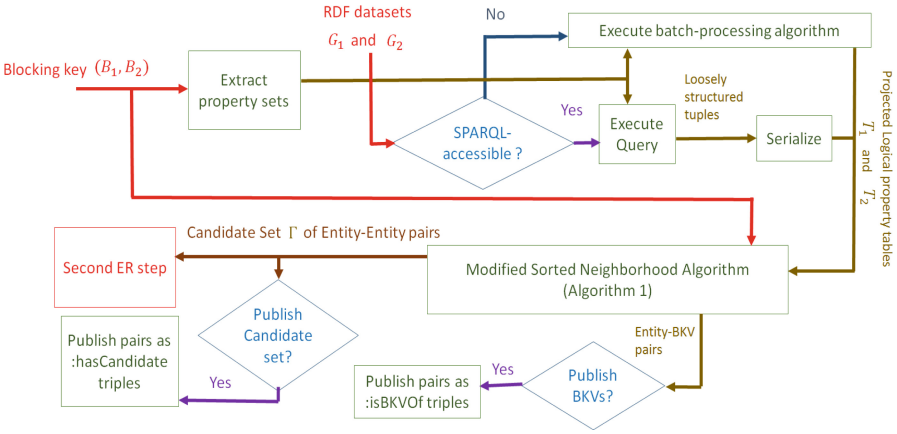

Fig. 2. A modular Sorted Neighborhood workflow for schema-free RDF graphs

computing the overlap between the properties' object value-sets using a trigram-based similarity score. Two entities share a block if they share tokens in any two properties that were grouped together. The main advantage of AC is that it does not require *property alignments* to be computed between the input RDF graphs, and is simple and effective to implement. Empirically, the learned blocking keys were found to be competitive when evaluated by a variety of blocking methods [16]. Note that the Sorted Neighborhood method was not proposed or considered in that paper. In Sect. 4, we use a high-performing blocking method called *block purging* from the original AC paper as a baseline for evaluating the proposed Sorted Neighborhood workflow.

The second class of learnable blocking keys, called the Disjunctive Normal Form (DNF) blocking keys, are currently considered state-of-the-art in the Relational Database community [1,9]. The learning procedure for these keys is adaptive, in that training samples are used by a machine learning algorithm to learn the key. In recent work, we extended DNF blocking keys to operate on schema-free RDF data [10,11]. A potential disadvantage of this method is that tractably learning these keys requires some form of property alignment between the schema-free datasets. Recent research has proposed some reliable methods for automatic property alignment, making the class of DNF blocking keys a promising unsupervised alternative to the AC class.

In summary, a practitioner has several viable options for acquiring blocking keys. In this paper, the only requirement that is imposed on the blocking key is that it must not rely on the existence of a schema or type information. In practice, this implies that the blocking key will be defined either in terms of the properties in the input RDF graphs or the subject URIs in the triples (or both). An even simpler alternative is to use the Canopy Clustering algorithm and block entities based on their degree of token overlap [12]. We consider this option as a baseline in Sect. 4.

3.2 Projected Logical Property Table

As mentioned in Sect. 3.1, the blocking keys $B_{1,2}$ will be defined[2] in terms of the properties in the RDF graphs $G_{1,2}$, since schema information cannot be assumed. Let the set of properties used in the construction of B_i be denoted as the *property set* P_i of the blocking key B_i $(i = 1, 2)$.

Example 1. Consider Fig. 1, and let the blocking keys B_1 and B_2 be given by the respective expressions[3] *Tokens(subject)* $\cup$ *Tokens(d1:hasBrother)* and *Tokens(subject)* $\cup$ *Tokens(d2:sibling)*, where *subject* indicates the subject URI of the entity to which the blocking key is applied. The property sets P_1 and P_2 are respectively {`d1:hasBrother`} and {`d2:sibling`}. *subject* is not included in the sets since it is not technically a property.

Given B_1 and B_2, the property sets P_1 and P_2 are first constructed. The goal of extracting the property sets is to construct a *projected logical property table*. A logical property table representation of an RDF graph G_i is defined by the schema $\{subject\} \cup \mathcal{P}_i$, where $\mathcal{P}_i$ is the set of *all* property URIs occurring in G_i [11]. The *subject* column essentially serves as the key for the table. Two special cases need to be borne in mind when populating a property table. First, there may be subjects that do not have object values for a given property. For example, in graph G_1 in Fig. 1, `d1:Sam_Crax` does not have an object value for the property `d1:hasBrother`. For such cases, a reserved keyword (e.g. *null*) is inserted in the corresponding table cell[4] The second special case is that an entity may have *mutliple* object values for a given property. Again, an example can be found in graph G_1 in Fig. 1, with `d1:Joan_Crax_Bates` having two object values for the property `d1:hasBrother`. For such cases, a reserved delimiter (e.g. ;) is used to collect multiple object values in a single table cell.

A projected logical property table with respect to a property set P is a table that contains only the *subject* column and the properties in P. In other words, the columns $\mathcal{P} - P$ are projected out from the schema of the logical property table. As with the logical property table, the projected table also always has the *subject* column as its key.

The advantages of this tabular serialization (of an RDF graph) are subsequently described. At present, we note that, if an RDF graph G_i is accessible as a *batch* file (e.g. in N-triples format), devising a linear-time batch-processing algorithm for scanning the triples and populating the table in two passes is straightforward [11]. In a first pass, the triples would be scanned and the property set P_i would be populated, along with an index with the subject URIs as keys. In the second pass, the rows of the initialized table would be populated. The index ensures that updates and inserts into the table are near-constant.

---

[2] $B_{1,2}$ is shorthand for the phrase 'B_1 and B_2'; similarly for other quantities.

[3] *Tokens* is a function that tokenizes a string into a set of words, based on a given set of delimiters. The set of tokens generated by the blocking key is precisely the set of blocking key values (BKVs) assigned to that entity.

[4] Located in the row of subject `d1:Sam_Crax` and column `d1:hasBrother`.

If the graph is accessible through a SPARQL endpoint, an efficient procedure is less straightforward. A naïve option would be to use a SELECT * style query to download the entire graph as a batch dump and then run the linear-time algorithm mentioned above. In an online setting, bandwidth is a precious resource. If $|P| << |\mathcal{P}|$, obtaining a full batch dump is clearly not efficient.

Instead, we deploy the following query on the SPARQL endpoint to obtain a table of *loosely structured tuples*:

SELECT ?entity $?o_1 \ldots ?o_d$
WHERE{
{SELECT DISTINCT ?entity WHERE ?entity ?p ?o.
FILTER (?p = $< p_1 >$ || ... || ?p = $< p_d >$}}
OPTIONAL {?entity $< p_1 >$ $?o_1$}
...
OPTIONAL {?entity $< p_n >$ $?o_d$}
}

Subject	d1:hasBrother
d1:Mike_Bates	
d1:Joan_Crax_Bates	d1:Sam_Crax
d1:Joan_Crax_Bates	d1:Roger_Crax
d1:Date	
d1:Sam_Crax	

(a)

Subject	d1:hasBrother
d1:Mike_Bates	*null*
d1:Joan_Crax_Bates	d1:Sam_Crax; d1:Roger_Crax
d1:Date	*null*
d1:Sam_Crax	*null*

(b)

Fig. 3. (a) is the table of loosely structured tuples obtained when the given SPARQL query is executed on graph G_1 in Fig. 1, assuming property set $P_1 = \{d1 : hasBrother\}$. (b) is the projected logical property table serialization of G_1 w.r.t P_1

Here, $< p_i >$ (for $i \in \{1 \ldots d\}$) is a placeholder for a property URI in P. In the nested query, the triples are filtered by the properties in P. As earlier mentioned, an RDF entity is not guaranteed to have an object value for a property in P. The *Optional* clause (for each property) provides a safeguard for this possibility.

Example 2. Consider again the first graph in Fig. 1. Assuming the blocking key *Tokens(subject)* $\cup$ *Tokens(d1:hasBrother)* defined in the example earlier, the loosely structured tuples generated by executing the query are shown in Fig. 3(a). Figure 3(b) shows the projected logical property table serialization of graph G_1 given the property set $P_1 = \{d1 : hasBrother\}$.

These tuples are denoted as *loosely-structured* for two reasons. First, the table is not guaranteed to contain a column that serves as a key. Secondly, there may be empty table cells. In order to impose structure on this table, therefore, the table is serialized into a projected logical property table.

We note that an advantage of employing the property table, instead of inventing a serialization, is that it already has a *physical* implementation in triple

Algorithm 1. Modified Sorted Neighborhood

Input: Blocking keys $B_{1,2}$, projected logical property tables $T_{1,2}$, windowing parameter w

Output: Candidate set Γ of entity-entity pairs

1. Initialize Γ to be an empty set
2. Apply B_1 (B_2) to each tuple in T_1 (T_2) to obtain *multimaps* $\Pi_1 = \{< bkv, R >\}$ ($\Pi_2 = \{< bkv, S >\}$), with bkv in each key-value set pair referring to a blocking key value string and R (S), denoted as a *block*, being a set of subject URIs in T_1 (T_2)
3. Join Π_1 and Π_2 on their keys to obtain joined map $\Pi_{map} = \{< bkv, < R, S >\}$, where $keySet[\Pi_{map}] = keySet[\Pi_1] \cap keySet[\Pi_2]$, and the larger of R and S in each pair in Π_{map} is truncated so that $|R| = |S|$
4. Sort Π_{map} into a list L with columns BKV, $subject_1$ and $subject_2$, using the keys in Π_{map} (or the BKV column) as sorting keys
5. For each tuple $< bkv, s_1, s_2 >$ in L, emit pairs $< s_1, bkv >$ and $< s_2, bkv >$ as entity-BKV pairs (Fig. 2)
6. Slide a window of size w over tuples in L (Fig. 4)
7. Add entity-entity pair $< e_1, e_2 >$ to Γ, where e_1 (e_2) is a subject URI from column $subject_1$ ($subject_2$) and e_1 and e_2 are in (not necessarily the same) tuples that fall within a common window
8. **return** Γ

stores such as Jena [21]. The primary advantage of the physical table is that it allows storing RDF files in back-end Relational Database infrastructure and significantly speeds up self-joins on properties for certain SPARQL queries. This paper proposes yet another use-case for this table; namely, realizing the Sorted Neighborhood algorithm for schema-free RDF data.

3.3 Candidate Set Generation

Given two projected logical property tables $T_{1,2}$ derived from graphs $G_{1,2}$, the blocking keys $B_{1,2}$, and the windowing parameter w, Algorithm 1 performs Sorted Neighborhood (SN) on $T_{1,2}$. Note that the original Sorted Neighborhood cannot be used, since it assumes either a single dataset (adhering to a single schema) or two datasets that have been individually cleansed of duplicates and then merged into a single dataset. Also, the original SN method assumes that each entity has at most one blocking key value [6]. In the heterogeneous space of RDF datasets, these assumptions are too restrictive [16]. Algorithm 1 performs the SN procedure without making these assumptions.

First, the algorithm applies the blocking keys to each tuple in the projected property tables and obtains two *multimaps* $\Pi_{1,2}$ of *blocks* indexed by a blocking key value (BKV), with each block essentially being a set of subject URIs. Considering the blocking key B_1 = *Tokens(subject)* $\cup$ *Tokens(d1:hasBrother)* in Example 1, one example of a generated block (on the first dataset in Fig. 1) would be $\{d1 : Mike\_Bates, d1 : Joan\_Crax\_Bates\}$, indexed by the BKV *Bates*, since

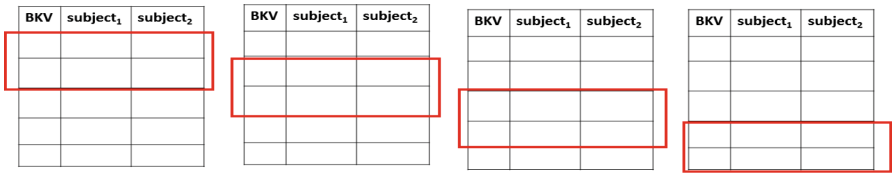

Fig. 4. The window sliding procedure on the sorted three-column list L, with $w = 2$

the two entities share a common token in their subject URIs. Note that an entity may have multiple BKVs and may occur in several blocks. Next, the algorithm joins Π_1 and Π_2 into a map Π_{map} by using the keysets (the BKVs) as the join keys. Thus, blocks in Π_1 that do not have a corresponding[5] block in Π_2 are discarded; similarly for blocks in Π_2. Also, for any two blocks R and S (from Π_1 and Π_2 respectively), we truncate the larger block by randomly removing elements till $|R| = |S|$. In line 4, Algorithm 1 sorts Π_{map} into a three-column sorted list by using the BKVs as sorting keys. This is similar to the sorting step in the original SN algorithm [6]. Finally, a window of size w is slid over the tuples in L and two entities (from columns $subject_1$ and $subject_2$ respectively) sharing a window are added to Γ (Fig. 4).

Algorithm 1 (and also, the last part of the workflow) allows the user to make operational decisions on whether the candidate set or the BKVs of entities (or both) should be *published* as sets of triples using two specially defined properties *:hasCandidate* and *:IsBKVOf* with *equivalence class* and *inverse function* semantics respectively. These triples can be published by third-party sources on dedicated servers and be made accessible as business services, similar to Relational Database cloud-based data matching[6] services.

4 Experimental Analysis

4.1 Test Cases, Metrics and Setup

The system is evaluated on five publicly available benchmarks, four of which (*Persons 1*, *Persons 2*, *Film* and *Restaurants*) were published as part of the 2010 *instance matching* track of OAEI[7], which is an annual Semantic Web initiative, and one of which (*IM-Similarity*) is a multilingual, crowdsourced dataset describing books and was released as part of OAEI 2014. These datasets, summarized in Table 1, span several domains and offer a variety of qualitative challenges.

[5] That is, indexed by the same blocking key value (BKV).

[6] An example is the D&B Business Insight service: https://datamarket.azure.com/dataset/dnb/businessinsight.

[7] Ontology Alignment Evaluation Initiative: http://oaei.ontologymatching.org/2010/#instance.

Table 1. Details of evaluation benchmarks. Each benchmark is a *pair* of RDF files with the notation (where applicable) being (first dataset) /× (second dataset)

ID	Name	Triples	Entity pairs	Matching entities
1	Persons 1	9000/7000	2000 × 1000 = 2 million	500
2	Persons 2	10,800/5600	2400 × 800 ≈ 1.92 million	400
3	Restaurants	1130/7520	339 × 2256 = 764,784	89
4	IM-Similarity	2204/2184	181 × 180 = 32,580	496
5	Film	9995/8979	1549 × 519 = 803,931	412

The blocking step[8] is typically evaluated by computing *efficiency* and *effectiveness* metrics for the candidate set Γ of entity-entity pairs generated by the blocking method. Let the number of *matching* (or ground-truth) entity pairs in Γ be denoted by the symbol Γ_m. Similarly, let Ω denote the full set (the Cartesian product) of entity pairs, and Ω_m denote the ground-truth set of matching entity pairs. With this terminology, the efficiency metric, *Reduction Ratio* (RR), and the effectiveness metric, *Pairs Completeness* (PC), are defined below:

$$RR = \frac{1 - |\Gamma|}{|\Omega|} \tag{1}$$

$$PC = \frac{|\Gamma_m|}{|\Omega_m|} \tag{2}$$

We capture the tradeoff between RR and PC by computing their harmonic mean or their F-score[9]:

$$F\text{-}score = \frac{2 \times RR \times PC}{(RR + PC)} \tag{3}$$

Note that, for all three metrics (PC, RR and the F-score), higher values indicate better performance.

We discovered the blocking key for each dataset in the five test cases by employing the trigrams-based Attribute Clustering (AC) algorithm that was described briefly in Sect. 3.1; complete details are provided in the original paper by Papadakis et al. [16]. In general, the learning algorithm leads to a blocking key that considers tokens of multiple properties (or 'attributes') and the name of the entity (its subject URI) as blocking key values. Brief examples were earlier provided. Once learned, Algorithm 1 is executed with w varied from 2 to 50. For each value of w, PC, RR and F-score values are recorded, with values corresponding to the highest-achieved F-score reported.

[8] This includes both the blocking key learning and the subsequent blocking method (such as the Sorted Neighborhood method presented in this paper).

[9] This is different from the F-score employed in Information Retrieval, where it is typically the harmonic mean of *precision* and *recall*.

4.2 Baselines and Implementation

Two baseline blocking methods are used to gauge the performance of Algorithm 1 on the five test cases. The first baseline is the *block purging* method that was used with the AC blocking key in the original paper along with a variety of other blocking methods, many of which made assumptions that are not applicable to this work [16]. Block purging was the least restrictive in terms of assumptions, simple to implement and execute (but with excellent empirical performance), and similar to the proposed method, involved a single parameter $maxPairs$. For these reasons, we employ it as a baseline to test the proposed system. Similar to lines 1–3 of Algorithm 1, block purging first generates a map of joined blocks Π_{map} (but without truncating larger blocks). Rather than employing a sliding window, the method discards an entry $< bkv, < R, S >>$ from Π_{map} if $|R||S| > maxPairs$. Among surviving entries, all entity-entity pairs in $R \times S$ are added to the candidate set Γ.

We also employed the classic *Canopy Clustering* (CC) blocking method as a second baseline [12]. CC is a generic clustering method that relies on an inexpensive similarity function such as cosine similarity, and is known to perform well on tabular datasets [4]. Given a single threshold parameter $thresh = [0, 1]$, the algorithm operates by locating for every tuple t in the *first* dataset, all tuples s in the *second* dataset such that the tuple pair $< t, s >$ has similarity at least $thresh$, and adds $< t, s >$ to the candidate set Γ. Note that CC does not rely on a blocking key, but uses all tokens in the tuples to compute the cosine similarity. We execute CC on the full[10] logical property table representations of the input RDF datasets. Similar to w in Algorithm 1 and $maxPairs$ in the block purging method, $thresh$ is varied and only the highest-achieved F-score values are reported. For a *fixed* value of the thresholds, note that the algorithms are deterministic and only need to be run once. Finally, we restricted parameter tuning (per test case) to a maximum of fifty iterations. In terms of run-time, all algorithms (for a given setting of the parameters) ran within a minute per dataset, making them roughly equal in that respect.

Finally, all programs were implemented serially in Java on a 32-bit Ubuntu virtual machine with 3385 MB of RAM and a 2.40 GHz Intel 4700MQ i7 processor. We have released datasets and ground-truth files on a project website[11].

4.3 Results and Discussion

Table 2 shows the highest F-scores obtained by the proposed method and the block purging baseline, along with corresponding PC and RR values. The results show that, on three of the five test cases, the modified SN procedure outperforms the block purging algorithm. On the RR metric, SN outperforms block purging by over 6 % and on the F-score metric by over 3.5 %. On the PC metric, block purging does better by about 1.7 %. Overall, the results show that the modified

[10] Since there is no blocking key (and no property set), projection does not take place.
[11] https://sites.google.com/a/utexas.edu/mayank-kejriwal/projects/sorted-neighborhood.

Table 2. Comparative results of Algorithm 1 and the block purging baseline. Both methods used the same blocking key, learned through Attribute Clustering [16]. *Count* tabulates number of test cases on which a method performs *equal/better/worse*

	Test Case	Sorted Neighborhood			Block Purging		
		PC	RR	F-score	PC	RR	F-score
1	Persons 1	100.00 %	99.26 %	**99.63 %**	100.00 %	98.86 %	99.43 %
2	Persons 2	91.25 %	89.77 %	90.50 %	99.75 %	99.02 %	**99.38 %**
3	Restaurants	100.00 %	99.41 %	99.70 %	100.00 %	99.57 %	**99.79 %**
4	IM-Similarity	100.00 %	84.73 %	**91.73 %**	100.00 %	62.79 %	77.14 %
5	Film	97.33 %	92.10 %	**94.64 %**	97.33 %	73.09 %	83.49 %
	Average	*97.72 %*	**93.05 %**	**95.24 %**	**99.42 %**	*86.67 %*	*91.85 %*
	Count	*4/0/1*	*0/3/2*	*0/3/2*	*4/1/0*	*0/2/3*	*0/2/3*

SN algorithm is a promising blocking candidate. We note that, on the RR metric in particular, the SN algorithm is more *stable*, with block purging achieving less than 75 % on at least two datasets. As earlier authors have noted [4], even small variations in RR can have a large impact on the run-time of a full ER workflow, since RR is quadratic in the number of input entities (Eq. 1). Thus, SN may be a more reliable method than block purging under resource-constrained settings.

Although not tabulated here due to space constraints, the average (on the five test cases) highest F-score and corresponding PC and RR results obtained by the Canopy Clustering (CC) baseline are 84.74 %, 80.39 % and 96.58 % respectively. Both the block purging and proposed method outperform CC on both PC and F-score by a large margin, demonstrating that on schema-free RDF data, simple token-based blocking techniques are rarely sufficient.

5 Conclusion and Future Work

In this paper, we proposed a Sorted Neighborhood workflow for schema-free RDF data, implemented as a sequence of relatively independent modules. All workflow steps can be operationalized using existing Semantic Web technology. Evaluations on five test cases show that the proposed algorithm is competitive with established blocking techniques for schema-free RDF data.

Future work will aim to deploy the system as a Linked Data service in a distributed paradigm, and to evaluate it on large-scale datasets. Given certain assumptions about the input datasets, a promising theoretical avenue is to automatically determine the optimal value of w for the inputs.

References

1. Bilenko, M., Kamath, B., Mooney, R.J.: Adaptive blocking: learning to scale up record linkage. In: Sixth International Conference on Data Mining, 2006. ICDM 2006, pp. 87–96. IEEE (2006)
2. Bizer, C., Heath, T., Berners-Lee, T.: Linked data-the story so far. Int. J. semant. Web Inf. Syst. **5**(3), 1–22 (2009)
3. Christen, P.: Further topics and research directions. In: Christen, P. (ed.) Data Matching, pp. 209–228. Springer, Heidelberg (2012)
4. Christen, P.: A survey of indexing techniques for scalable record linkage and deduplication. IEEE Trans. Knowl. Data Eng. **24**(9), 1537–1555 (2012)
5. Elmagarmid, A.K., Ipeirotis, P.G., Verykios, V.S.: Duplicate record detection: a survey. IEEE Trans. Knowl. Data Eng. **19**(1), 1–16 (2007)
6. Hernández, M.A., Stolfo, S.J.: The merge/purge problem for large databases. In: ACM SIGMOD Record, vol. 24, pp. 127–138. ACM (1995)
7. Hernández, M.A., Stolfo, S.J.: Real-world data is dirty: data cleansing and the merge/purge problem. Data Min. Knowl. Disc. **2**(1), 9–37 (1998)
8. Isele, R., Jentzsch, A., Bizer, C.: Efficient multidimensional blocking for link discovery without losing recall. In: WebDB (2011)
9. Kejriwal, M., Miranker, D.P.: An unsupervised algorithm for learning blocking schemes. In: Thirteenth International Conference on Data Mining, ICDM 2013. IEEE (2013)
10. Kejriwal, M., Miranker, D.P.: A two-step blocking scheme learner for scalable link discovery. In: Thirteenth International Semantic Web Conference on Ontology Matching Workshop, ISWC 2014 (2014)
11. Kejriwal, M., Miranker, D.P.: A dnf blocking scheme learner for heterogeneous datasets (2015). arXiv preprint arXiv:1501.01694
12. McCallum, A., Nigam, K., Ungar, L.H.: Efficient clustering of high-dimensional data sets with application to reference matching. In: Proceedings of the Sixth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, pp. 169–178. ACM (2000)
13. McCarthy, J.F., Lehnert, W.G.: Using decision trees for coreference resolution (1995). arXiv preprint cmp-lg/9505043
14. Newcombe, H., Kennedy, J., Axford, S., James, A.: Automatic linkage of vital records (1959)
15. Ngomo, A.-C.N.: A time-efficient hybrid approach to link discovery. In: Ontology Matching, p. 1 (2011)
16. Papadakis, G., Ioannou, E., Niederée, C., Fankhauser, P.: Efficient entity resolution for large heterogeneous information spaces. In: Proceedings of the Fourth ACM International Conference on Web Search and Data Mining, pp. 535–544. ACM (2011)
17. Puhlmann, S., Weis, M., Naumann, F.: XML duplicate detection using sorted neighborhoods. In: Ioannidis, Y., Scholl, M.H., Schmidt, J.W., Matthes, F., Hatzopoulos, M., Böhm, K., Kemper, A., Grust, T., Böhm, C. (eds.) EDBT 2006. LNCS, vol. 3896, pp. 773–791. Springer, Heidelberg (2006)
18. Scharffe, F., Ferrara, A., Nikolov, A., et al.: Data linking for the semantic web. Int. J. Seman. Web Inf. Syst. **7**(3), 46–76 (2011)
19. Scharffe, F., Liu, Y., Zhou, C.: RDF-AI: an architecture for RDF datasets matching, fusion and interlink. In: Proceedings of the IJCAI 2009 workshop on Identity, reference, and knowledge representation (IR-KR), Pasadena (CA US) (2009)

20. Schmachtenberg, M., Bizer, C., Paulheim, H.: Adoption of the linked data best practices in different topical domains. In: Mika, P., et al. (eds.) ISWC 2014, Part I. LNCS, vol. 8796, pp. 245–260. Springer, Heidelberg (2014)
21. Wilkinson, K., Sayers, C., Kuno, H.A., Reynolds, D., et al.: Efficient RDF storage and retrieval in Jena2. SWDB **3**, 131–150 (2003)