Efficient Multidimensional Blocking for Link Discovery without losing Recall

Robert Isele, Anja Jentzsch, and Christian Bizer
Freie Universität Berlin, Web-based Systems Group
Garystr. 21, 14195 Berlin, Germany
mail@robertisele.com, mail@anjajentzsch.de, chris@bizer.de

ABSTRACT

Over the last three years, an increasing number of data providers have started to publish structured data according to the Linked Data principles on the Web. The resulting Web of Data currently consists of over 28 billion RDF triples. As the Web of Data grows, there is an increasing need for link discovery tools which scale to very large datasets. In record linkage, many partitioning methods have been proposed which substantially reduce the number of required entity comparisons. Unfortunately, most of these methods either lead to a decrease in recall or only work on metric spaces. We propose a novel blocking method called MultiBlock which uses a multidimensional index in which similar objects are located near each other. In each dimension the entities are indexed by a different property increasing the efficiency of the index significantly. In addition, it guarantees that no false dismissals can occur. Our approach works on complex link specifications which aggregate several different similarity measures. MultiBlock has been implemented as part of the Silk Link Discovery Framework. The evaluation shows a speedup factor of several 100 for large datasets compared to the full evaluation without losing recall.

Keywords

Blocking, Link Discovery, Identity Resolution, Duplicate Detection, Record Linkage, Linked Data

1. INTRODUCTION

The Web of Data forms a single global data space for the very reason that its data sources are connected by RDF links [2]. While there are some fully automatic tools for link discovery [6], most tools generate links semi-automatically based on *link specifications* [20, 16, 11]. Link specifications specify the conditions which must hold true for a pair of entities for the link discovery tool to generate a RDF link between them. Based on a link specification, the link discovery tool compares entities and concludes to set links if the aggregated similarity is above a given threshold. The naive approach to compare all entities with each other does not scale due to the computation of the Cartesian product of all the entities.

As the Web of Data is growing fast there is an increasing need for link discovery tools which scale to very large datasets. A number of methods have been proposed to improve the efficiency of link discovery by dismissing definitive non-matches prior to comparison. The most well-known method to achieve this is known as *blocking* [5]. Unfortunately, standard blocking methods in general lead to a decrease of recall due to false dismissals [4].

We propose a novel blocking approach which maps entities to a multidimensional index, called MultiBlock. The basic idea of the mapping function is that it preserves the distances of the entities i.e. similar entities will be located near to each other in the index space. While standard blocking techniques block in one dimension, MultiBlock concurrently blocks by multiple properties using multiple dimensions increasing its efficiency significantly. MultiBlock has been implemented and evaluated within the Silk Link Discovery Framework. It works on complex link specifications which aggregate multiple different similarity measures such as string, geographic or date similarity and does not need any additional configuration. MultiBlock is organized in three phases:

1. In the **index generation** phase, an index is built for each similarity measure. The basic idea of the indexing method is that it preserves the distances of the entities i.e. similar entities will be located near each other in the index. The specific indexing method depends on the employed similarity measure.
2. In the **index aggregation** phase, all indexes are aggregated into one multidimensional index, preserving the property of the indexing that the indexes of two entities within a given distance share the same index.
3. Finally, the **comparison pair generation** employs the index to generate the set of entity pairs which are potential links. These pairs are then evaluated using the link specification to compute the exact similarity and determine the actual links.

We illustrate the indexing by looking at the simple example of interlinking geographical entities based on their label and geographic coordinates: In that case the index generation phase would generate 2 indices: A 1-dimensional index of the labels and a 2-dimensional index of the coordinates. The index aggregation phase would than aggregate both indexes into a single 3-dimensional index. Figure 1 visualises the index generation and aggregation in this example. Note that each similarity measure may create multiple indexes for a single entity, which for simplicity is not considered in the Figure.

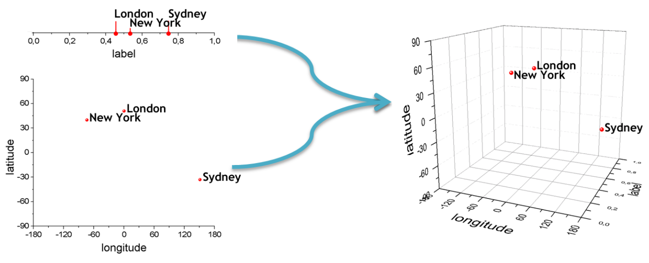

Figure 1: Aggregating a geographic and a string similarity

Figure 1 shows the aggregated index for 1,000 cities in DBpedia.

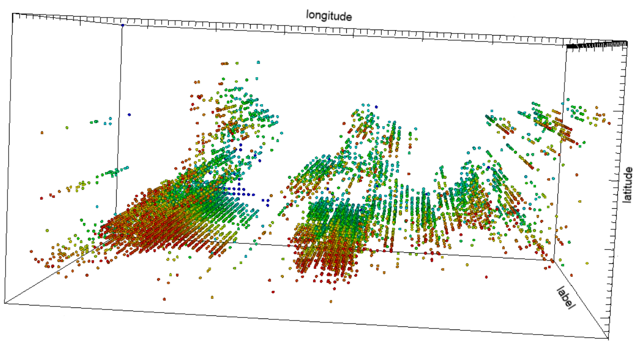

Figure 2: Index of 1,000 cities in DBpedia

This paper is structured as follows: In the next Section we discuss related work. In Section 3, we outline how our work contributes to the current state of art. Section 4, introduces some preliminaries on the process of link discovery. Based on these foundations, Section 5 explains the general framework of our approach which is independent of a specific similarity measure or similarity aggregation. At the same time, it specifies which properties a similarity measure or aggregation must adhere to in order to be used in our approach. Subsequently, Section 6 specifies various similarity measures and aggregations which can be plugged into the framework in order to provide a complete blocking method. Our implementation as part of the *Silk Link Discovery Framework* [16] is discussed in Section 7. Finally, Section 8 reports on the results of the evaluation.

2. RELATED WORK

The problem of link discovery is very similar to record linkage. In record linkage [5] a number of methods to improve the efficiency by reducing the number of required comparisons are often applied:

Traditional **blocking** methods work by partitioning the entities into blocks based on the value of a specific property [1]. Then only entities from the same block are compared reducing the number of comparisons significantly at the cost of a loss in accuracy. Especially in cases where the data is noisy as it is often the case in Linked Data, similar entities might be assigned to different blocks and thus not compared in the subsequent comparison phase. In order to reduce the number of false dismissals, a multi-pass approach has been proposed [13]. In a multi-pass approach the blocking is run several times, each time with a different blocking key.

The **Sorted-Neighborhood** method has been proposed in order to improve the handling of fuzzy data [12]. The Sorted-Neighborhood method works on the list of entities which has been sorted according to a user-defined key. The entities which are selected for comparison are determined by sliding a fixed-size window along the sorted list. Only entities inside the window are selected for comparison. The biggest problem of the Sorted-Neighborhood Method lies in the choice of the window size. A small size may miss entities if many similar entities share the same key. A big size will lead to a decrease in efficiency. A solution for this is to adapt the windows size while sliding through the list [24].

The **Sorted Blocks** [4] method generalizes Blocking and Sorted-Neighborhood in order to overcome some of their individual disadvantages. It uses overlapping blocks and is both easy to parallize and more stable in the presence of noise.

While Blocking and Sorted-Neighborhood methods usually map the property key to a single dimensional index, some methods have been developed which map the similarity space to a multidimensional Euclidean space. The fundamental idea of these methods is to preserve the distance of the entities i.e. after mapping, similar entities are located close to each other in the Euclidean space. Techniques which use this approach include **FastMap** [8], **MetricMap** [21], **SparseMap** [15] and **StringMap** [17]. Unfortunately, in general, these methods do not guarantee that no false dismissals will occur [14]. The only exception is SparseMap for which variants have been proposed which guarantee no false dismissals [14]. All of these approaches require the similarity space to form a metric space i.e. the similarity measure must respect the triangle inequality. This implies that they can not be used with non-metric similarity measures such as Jaro-Winkler [22].

Another approach which uses the characteristics of metric spaces, in particular the triangle inequality, to reduce the number of similarity computations, has been implemented in LIMES[20]. To the best of our knowledge no other tool in link discovery, besides Silk and LIMES, makes use of blocking techniques.

3. CONTRIBUTIONS

Our main contribution to the current state of art is that, while significantly reducing the number of comparisons, MultiBlock guarantees that no false dismissals and thus no loss of recall can occur and does not require the similarity space to form a metric space. In addition, it uses a multidimensional index increasing its efficiency significantly. Another advantage of MultiBlock is that it can work on a stream of entities as it does not require to preprocess the whole dataset. Table 1 summarizes how MultiBlock compares to existing methods.

Method	Lossless	Non-Metrics	Stream.
Traditional Blocking	no	yes	yes
Sorted-Neighborhood	no	yes	no
Sorted Blocks	no	yes	no
FastMap	no	no	no
MetricMap	no	no	no
SparseMap	no	no	no
StringMap	no	no	no
Modified SparseMap	yes	no	no
MultiBlock	**yes**	**yes**	**yes**

Table 1: Comparison of different blocking methods

4. PRELIMINARIES

The general problem of link discovery can be formalized as follows [9]:
Given two datasets A and B, find the subset of all matching pairs:

$$M = \{(a, b); sim_s(a, b) \geq \theta, a \in A, b \in B\} \tag{1}$$

Where sim_s assigns a similarity value to each pair of entities:

$$sim_s : \; A \times B \to [0, 1] \tag{2}$$

If the similarity between two entities exceeds a threshold θ, a link between these two entities is generated. sim is computed by evaluating a link specification s (in record linkage typically called linkage decision rule [23]) which specifies the conditions two entities must fulfill in order to be interlinked.

A link specification typically applies one or more similarity measures to the values of a set of property paths. While in the case of database records this is typically achieved by selecting a subset of the record fields, in Linked Data the features are selected by specifying a set of property paths through the graph. If the data sources use different data types, the property values may be normalized by applying a transformation prior to the comparison. Due to the reason that in most cases the similarity of two entities can not be determined by evaluating a single property, a typical link specification aggregates multiple similarity measures into one compound similarity value. Finally, a threshold is applied to determine if the entities should be interlinked.

In order to compute the set of all matching pairs M, the naive approach is to evaluate the link specification for each entity of the Cartesian product $A \times B$ which results in $|A| \times |B|$ comparisons. Because this is infeasible for all but very small datasets, we define a blocking function which assigns, based on the link specification, a block to each entity:

$$block_s : \; A \cup B \to \mathbb{N} \tag{3}$$

The fundamental idea behind blocking is to assign the same block to similar entities so that comparisons only have to be made between the entities in each block. This decreases the number of required comparisons considerably by reducing the set of pairs which have to be evaluated to:

$$R = \{(a, b); block_s(a) = block_s(b), a \in A, b \in B\} \tag{4}$$

5. APPROACH

In this section, we lay down the general framework of our approach which is independent of a specific similarity measure or aggregation. At the same time, we define which properties a similarity measure or aggregation must adhere to in order to be used in our approach. The subsequent Section will specify various similarity measures and aggregations which can be plugged into the framework in order to provide a complete blocking method.

Our approach is organized in three phases: At first, for each similarity measure, an index is generated. Afterwards, all similarity values are aggregated into a single multidimensional index. Finally, the comparison pairs are generated from the index.

5.1 Index Generation

For each similarity measure in the link specification, an index is built which consists of a set of vectors which define locations in the Euclidean space. The basic idea of the indexing method is that it preserves the distances of the entities i.e. similar entities will be located near each other in the index. The index generation is not restricted to a specific similarity measure. In order to be used for MultiBlock, each similarity measure must define the following functions:

1. A function which computes a similarity value for a pair of entities:
$$sim_s : \; A \times B \to [0, 1] \tag{5}$$
2. A blocking function which generates the index for a single entity:
$$index_s : \; (A \cup B) \times [0, 1] \to \mathcal{P}(\mathbb{N}^n) \tag{6}$$
where $\mathcal{P}$ denotes the power set (i.e. it might generate multiple blocks for a single entity) and n is the dimension of the index. The first argument denotes the entity to be blocked, which may be either in the source or target set. The second argument denotes the similarity threshold. $index_s$ includes two modifications of the standard $block$ function presented in the preliminaries. Firstly, it does not map each entity to a one-dimensional block, but to a multi-dimensional block. This way increases the efficiency as the entities are distributed in multiple dimensions. Secondly, it does not map each entity to a single block, but to multiple blocks at once, similarily to multi-pass blocking. This avoids losing recall if an entity cannot be mapped to a definite block such as in string similarity measures (see Section 6.1).

The $index_s$ function must adhere to the property that two entities whose similarity according to sim_s is below the threshold must share a block. More formally, given two entities e_1, e_2 and a threshold θ, sim_s and $index_s$ must be related by the following equivalence:

$$sim_s(e_1, e_2) \leq \theta \iff index_s(e_1) \cap index_s(e_2) \neq \emptyset \tag{7}$$

Section 6.1 gives an overview over the most common similarity measures which can be used in MultiBlock.

5.2 Index Aggregation

In the index aggregation phase, all indexes which have been built in the index generation phase are aggregated into one compound index. The aggregation function preserves the property of the index that two entities within a given distance share the same index vector. Generally, aggregating the indexes of multiple similarity measures will lead to an increase in dimensionality, but the concrete aggregation function depends on the specific aggregation type. Section 6.2 outlines the concrete aggregation functions for the most common aggregation types.

In order to be used for MultiBlock, each aggregation must define the following functions:

1. A function which aggregates multiple similarity values:
$$aggSim_a: \ [0,1]^n \times [0,1]^n \rightarrow [0,1] \tag{8}$$
where n is the number of operators to be aggregated. The first argument denotes the similarity values to be aggregated. As an aggregation may weight the results of the underlying operators e.g. a weighted average, the second argument denotes the weight of the specific operator. Each weight is a number between 0 and 1, while all weights total to 1.
2. A function which aggregates multiple blocks:
$$aggIndex_a: \ \mathcal{P}(\mathbb{N}^n) \times \mathcal{P}(\mathbb{N}^n) \rightarrow \mathcal{P}(\mathbb{N}^n) \tag{9}$$
where $\mathcal{P}$ denotes the power set and n is the dimension of the index. Note that while $aggIndex_a$ only aggregates two sets of blocks at once, it can also be used to aggregate multiple sets by calling it repeatedly.
3. A function which updates the threshold of the underlying operators in order to retain the condition that two entities within the threshold share a block.
$$threshold_a: \ [0,1] \times [0,1] \rightarrow [0,1] \tag{10}$$
The first argument denotes the threshold on the aggregation. As in the similarity aggregation function, the second argument denotes the weight of the specific operator.

5.3 Comparison Pair Generation

Finally, the comparison pair generation employs the index to generate the set of entity pairs which are potential links. For each two entities which share a block, a comparison pair is generated. These pairs are then evaluated using the link specification to compute the exact similarity and determine the actual links.

6. EMPLOYED SIMILARITY MEASURES AND AGGREGATIONS

This section provides an overview of various similarity measures and aggregations which can be used in conjunction with our approach. For each similarity measure, we define the required similarity function and the corresponding blocking function. Likewise, for each aggregation, we define the required similarity aggregation function as well as the blocking aggregation function.

6.1 Similarity Measures

As RDF datasets typically make use of a variety of different data types, many similarity measures have been proposed to match their values [7]. As the most common data types are plain string literals, most of these techniques handle approximate string matching. Apart from that, similar techniques have been proposed for numeric data or special purpose data types such as dates or geographic coordinates. In this section, we show how similarity measures for various data types can be integrated into our proposed approach.

String Similarity

A number of string similarity measures have been developed in literature [19, 3]. We use the Levenshtein distance [18] for approximate string comparisons in this paper.

Given a finite alphabet Σ and two strings $\sigma_1 \in \Sigma^*$ and $\sigma_2 \in \Sigma^*$, we define the similarity function to compute the normalized Levenshtein distance as:
$$sim_s(\sigma_1, \sigma_2) := \frac{levenshtein(\sigma_1, \sigma_2)}{max(|\sigma_1|, |\sigma_2|)} \tag{11}$$

The basic problem of blocking string values under the presence of typographical errors is the potential loss of recall. We define a blocking function which avoids false dismissals by indexing multiple q-Grams of the given input string. For this purpose, we first define a function which assigns a single block to a given q-Gram $\sigma_q \in \Sigma^q$:
$$block_q(\sigma_q) := \sum_{i=0}^{q} |\Sigma|^i \cdot \sigma_q(i) \tag{12}$$
$block_q$ assigns each possible letter combination of the q-Gram to a different block.

In order to increase the efficiency, we do not want to block all q-Grams of a given string, but just as many needed to avoid any false dismissals. We can make the following observation [10]: Given a maximum Levenshtein distance k between two strings, they differ by at most $k \cdot q + 1$ q-Grams, wherein the maximum Levenshtein distance is given as $k := max(|str1|, |str2|) \cdot (1.0 - \theta)$. Consequently, the minimal number of q-grams which must be blocked in order to avoid false dismissals is:
$$c(\theta) := max(|str1|, |str2|) \cdot (1.0 - \theta) \cdot q + 1 \tag{13}$$

By combining both functions, we can define the blocking function as:
$$index_s(\sigma, \theta) := \{block_q(\sigma_q); \sigma_q \in qgrams(\sigma)[0...c(\theta)]\} \tag{14}$$
The function starts with decomposing the given string into its q-Grams. From this set, it takes as many q-Grams as needed to avoid false dismissals and assigns a block to each. Finally, it returns the set of all blocks.

Numeric Similarity

The similarity of two numbers is computed with:
$$sim_s(d_1, d_2) := \frac{|d_1 - d_2|}{d_{max} - d_{min}} \text{ where } d_1, d_2 \in [d_{min}, d_{max}] \tag{15}$$

Using standard blocking without overlapping blocks may lead to false dismissals. For that reason, we define an overlapping factor *overlap*, which we set to 0.5 by default. The overlapping factor specificies to what extend the blocks overlap. Using the overlapping factor, the maximum number of blocks which does not lead to false dismissals can be computed with:
$$size_s(\theta) := \frac{1}{\theta} \cdot overlap \tag{16}$$

Based on the block count we can define the blocking function as follows:
$$index_s(d, \theta) := \begin{cases} \{0\} & \text{if } scaled(d) <= 0.5 \\ \{size_s(\theta) - 1\} & \text{if } scaled(d) >= size_s(\theta) - 0.5 \\ \{i(d), i(d) - 1\} & \text{if } scaled(d) - i(d) < overlap \\ \{i(d), i(d) + 1\} & \text{if } scaled(d) - i(d) + 1 < overlap \\ \{i(d)\} \end{cases}$$
$$\text{with } scaled(d) := size_s(\theta) \cdot \frac{d - d_{min}}{d_{max}}$$
$$i(d) := floor(\frac{d - d_{min}}{d_{max}}) \tag{17}$$

Geographic Similarity
Blocking geographic coordinates can be reduced to indexing numbers, by using the numeric similarity functions on both the latitude and the longitude of the coordinate which results in 2-dimensional blocks.

6.2 Aggregations

In this section we focus on the most common aggregations: Computing the average similarity and selecting the minimum or maximum similarity value.

Average Aggregation
The average aggregation computes the weighted arithmetic mean of all provided similarity values.

$$aggSim_a(v,w) := \frac{v_0 * w_0 + v_1 * w_0 + ... + v_n * w_n}{n} \tag{18}$$

Two indexes are combined by concatenating their index vectors:

$$aggIndex_a(A,B) := \{(a_1, ..., a_n, b_1, ...b_m), a \in A, b \in B\} \tag{19}$$

In order to preserve the condition that two entities within the given threshold share a block, the local threshold of the underlying similarity measures is modified according to:

$$threshold_a(\theta, w) := 1 - (1-\theta)\frac{1}{w} \tag{20}$$

Minimum/Maximum Aggregation
The minimum and maximum aggregations simply select the minimum/maximum value:

$$aggSim_a(v,w) := min/max(v_0 + v_1 + ... + v_n) \tag{21}$$

In this case, we can not just aggregate the blocks to separate dimensions in the same way as in the average aggregator. The reason for this is that if one similarity value exceeds the threshold, the remaining similarity values may be arbitrary low while the entities are still considered as matches. For this reason, the index vectors of all indexes are mapped into the same index space:

$$aggIndex_a(A,B) := \{(a_1, ..., a_n), (b_1, ..., b_n); a \in A, b \in B\} \tag{22}$$

In case the dimensionality of the two indexes does not match, the vectors of the lower dimensional index are expanded by setting the values in the additional dimensions to zero.

For minimum and maximum aggregations we can leave the threshold unchanged:

$$threshold_a(\theta, w) := \theta \tag{23}$$

7. IMPLEMENTATION

The Silk Link Discovery Framework generates RDF links between data items based on user-provided link specifications which are expressed using the Silk Link Specification Language (Silk-LSL). The Silk Link Discovery Framework is implemented in Scala[1] and can be downloaded from the project homepage[2] under the terms of the Apache Software License.

Until version 2.2, Silk supports basic blocking with overlapping blocks. It provides a separate configuration directive to configure the property paths which are used as blocking keys as well as the number of block and the overlapping factor.

The current version of Silk includes the MultiBlock method as specified in this paper. The blocking is configured by the link specification and does not need any further configuration. The blocking function has been implemented for all common similarity measures. For similarity measures which do not define a blocking function yet, the fallback is to use a single block for this specific similarity measure. In this case, Silk still blocks by the remaining similarity measures.

[1]http://scala-lang.org
[2]http://www4.wiwiss.fu-berlin.de/bizer/silk/

Method	Comparisons	Runtime	Links
Full evaluation	108,301,460,054	305,188s	70,037
Blocking, 100 blocks	3,349,755,846	22,453s	69,403
Blocking, 1,000 blocks	1,049,015,356	7,909s	60,025
MultiBlock	37,667,462	420s	70,037

Table 2: Results of experiment 1

8. EVALUATION

MultiBlock has been evaluated regarding scalability and effectiveness.

This section reports on the results of two experiments in which we used Silk with MultiBlock to generate RDF links between different Linked Data sets. The link specifications used for the experiments are included in the current release of Silk.

All experiments have been run on a 3GHz Intel(R) Core i7 CPU with 4 cores and 8GB of RAM.

8.1 Scalability

In order to be used for link discovery in Linked Data, MultiBlock must be able to scale to very large datasets. Thus, it is essential that MultiBlock reduces the number of comparisons drastically without dismissing correct pairs. We evaluated the scalability of MultiBlock by applying it to interlink two large geographic datasets. For this purpose, we used a dataset consisting of 204,109 settlements from DBpedia[3] and 530,606 settlements from LinkedGeoData[4].

First, we interlinked both datasets by evaluating the complete Cartesian product without the use of any blocking method. As this results in over 100 billion comparisons it is a clear case when matching the complete Cartesian product is not reasonable anymore. The link generation took about 85 hours and generated 70,037 links. The generated links have been spot-checked for correctness.

After that, we evaluated how standard blocking reduces the number of comparisons. We used the labels of the entities as blocking keys. We ran the blocking multiple times with different parameters in each run. In order to reduce the loss of recall we used overlapping blocks. An overlapping factor of 0.2 was chosen for that purpose as it provides a good trade-off between efficiency and minimizing the number of false dismissals.

Finally, we evaluated how MultiBlock compares to standard blocking. By blocking in 3 dimensions instead of a single dimension, MultiBlock was able to further reduce the number of comparisons to 37,667,462. Furthermore, it generated the identical 70,037 links as in the full evaluation, but ran in only 420 seconds.

Table 2 summarizes the results. The evaluation shows that MultiBlock reduces the number of comparisons by a factor of 2,875 and is over 700 times faster than evaluating the complete Cartesian product. It is also almost 20

[3]http://dbpedia.org
[4]http://linkedgeodata.org

Phase	Time
Build index	14 %
Generate comparison pairs	41 %
Similarity comparison	45 %

Table 3: The runtimes of the different phases

Setting	Comparisons	Runtime	Links
Full comparison	22,242,292	430s	1,403
Blocking, 100 blocks	906,314	44s	1,349
Blocking, 1,000 blocks	322,573	14s	1,287
MultiBlock	122,630	6s	1,403

Table 4: Results of experiment 2

times faster than standard blocking with 1,000 blocks, but contrary to it does yield all correct links. We can also observe that due to the usage of a multidimensional index, the overhead of MultiBlock is higher than that of standard blocking. This can be concluded from the fact that MultiBlocks reduces the number of comparisons by 28 compared to standard blocking with 1,000 blocks, but its runtime only increases by a factor of 19.

In order to determine which phase of MultiBlock is responsible for the overhead, we evaluated the runtimes of the different phases of the matching process. As we can see in Table 3, a big part is spent for creating the comparison pairs. For this reason future improvements will focus on using a more efficient algorithm for the comparison pair generation.

8.2 Effectiveness

In order to be applicable to discover links between arbitrary data sources, MultiBlock must be flexible enough to be applied even to complex data sets without losing recall. For this purpose we employed Silk with MultiBlock to interlink drugs in DBpedia and DrugBank[5]. Here it is not sufficient to compare the drug names alone, but also necessary to take the various unique bio-chemical identifiers, e.g. CAS number, into consideration. Therefore the corresponding Silk link specification compares the drug names and their synonyms as well as a list of well-known and used identifiers of which not all have to be set on the entity.

The employed Silk link specification results in 1,403 links. Table 4 shows the different runtimes using Silk and Silk with MultiBlock. Using Silk with MultiBlock, we achieved a speedup factor of 71 with full recall. The gain in this example is smaller than in the previous one because the dataset here is much smaller and the link specification is more complicated.

9. REFERENCES

[1] R. Baxter, P. Christen, and C. F. Epidemiology. A comparison of fast blocking methods for record linkage. 2003.

[2] C. Bizer, T. Heath, and T. Berners-Lee. Linked data - the story so far. *Int. J. Semantic Web Inf. Syst*, 5(3):1–22, 2009.

[3] W. W. Cohen, P. Ravikumar, and S. E. Fienberg. A comparison of string distance metrics for name-matching tasks. 2003.

[5]DrugBank is a large repository of almost 5,000 FDA-approved drugs and has been published as Linked Data on `http://www4.wiwiss.fu-berlin.de/drugbank/`

[4] U. Draisbach and F. Naumann. A comparison and generalization of blocking and windowing algorithms for duplicate detection. 2009.

[5] A. K. Elmagarmid, P. G. Ipeirotis, et al. Duplicate record detection: A survey. *IEEE Trans. on Knowl. and Data Eng.*, 19(1), 2007.

[6] J. Euzenat, A. Ferrara, C. Meilicke, et al. First Results of the Ontology Alignment Evaluation Initiative 2010. *Ontology Matching*, page 85, 2010.

[7] J. Euzenat and P. Shvaiko. *Ontology matching.* Springer-Verlag, Heidelberg (DE), 2007.

[8] C. Faloutsos and K. Lin. FastMap: A fast algorithm for indexing, data-mining and visualization of traditional and multimedia datasets. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data.* ACM, 1995.

[9] I. P. Fellegi and A. B. Sunter. A Theory for Record Linkage. *Journal of the American Statistical Association*, 64(328), 1969.

[10] L. Gravano, P. G. Ipeirotis, H. V. Jagadish, et al. Approximate string joins in a database (almost) for free. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, San Francisco, USA, 2001.

[11] O. Hassanzadeh, R. Xin, R. J. Miller, et al. Linkage query writer, 2009.

[12] M. A. Hernández and S. J. Stolfo. The merge/purge problem for large databases. In *Proceedings of the 1995 ACM SIGMOD international conference on Management of data*, SIGMOD '95, New York, NY, USA, 1995. ACM.

[13] M. A. Hernández and S. J. Stolfo. Real-world data is dirty: Data cleansing and the merge/purge problem. *Data Min. Knowl. Discov.*, 2, 1998. Introduced the multi-pass approach for blocking.

[14] G. R. Hjaltason and H. Samet. Properties of embedding methods for similarity searching in metric spaces. *PAMI*, 25, 2003.

[15] G. Hristescu and M. Farach-Colton. Cluster-preserving embedding of proteins, 1999.

[16] A. Jentzsch, R. Isele, and C. Bizer. Silk - Generating RDF Links while publishing or consuming Linked Data. In *Poster at the International Semantic Web Conference (ISWC2010), Shanghai*, 2010.

[17] L. Jin, C. Li, and S. Mehrotra. Efficient Record Linkage in Large Data Sets. In *Database Systems for Advanced Applications (DASFAA 2003).* IEEE, 2003.

[18] V. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. In *Soviet Physics Doklady*, volume 10, 1966.

[19] G. Navarro. A guided tour to approximate string matching. *ACM Comput. Surv.*, 33, 2001.

[20] A.-C. N. Ngomo and S. Auer. Limes - a time-efficient approach for large-scale link discovery on the web of data.

[21] J. Wang, X. Wang, K. Lin, et al. Evaluating a class of distance-mapping algorithms for data mining and clustering. In *Proceedings of the 5th ACM SIGKDD international conference on Knowledge discovery and data mining.* ACM, 1999.

[22] W. Winkler. String Comparator Metrics and Enhanced Decision Rules in the Fellegi-Sunter Model of Record Linkage. In *Proceedings of the Section on Survey Research Methods, American Statistical Association.*, 1990.

[23] W. E. Winkler. Matching and Record Linkage. In *Business Survey Methods*, pages 355–384, 1995.

[24] S. Yan, D. Lee, M.-Y. Kan, et al. Adaptive sorted neighborhood methods for efficient record linkage. In *Proceedings of the 7th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '07, USA, 2007.