ScSLINT: Time and Memory Efficient Interlinking Framework for Linked Data

Khai Nguyen and Ryutaro Ichise

The Graduate University for Advanced Studies, Japan
National Institute of Informatics, Japan
{nhkhai,ichise}@nii.ac.jp

Abstract. Data interlinking is the problem of detecting the instances of different repositories but co-refer to the same topic. The large scale of linked data pushes a challenge to current interlinking algorithms. ScSLINT is an extension of SLINT+ [4] and its focus is scalability. ScSLINT also includes several modified features for the resolution sub-steps. The impressive performance of ScSLINT is validated by an experiment on very large datasets.

1 Introduction

The goal of data interlinking is to detect all instances that co-refer to the same objects in two repositories, the source and the target. The interlinking problem on web-based data, such like linked data, is considered as more challenging than other types of data because of the heterogeneity and scalability issues. Linked data interlinking is a well-studied problem [1] because of its importance in data integration and indispensable role in linked data publication.

Among many proposed solutions, SILK [6] is known for its frontier in linked data interlinking framework. Unfortunately, SILK is not well optimized for large scale dataset. LIMES [2] is a state-of-the-art framework. LIMES focuses on speeding up the interlinking process by reducing the number of real comparisons using the characteristics of metric space. However, on really large dataset, LIMES still shows the limitation in scalability.

Following the success of SLINT+ [4, 3], we develop its new release, ScSLINT, as a highly scalable interlinking framework. Compared to SLINT+, ScSLINT also comes with more advanced built-in features.

2 The ScSLINT

The architecture of ScSLINT is depicted by Fig. 1. R_{source} and R_{target} are the input repositories. We describe the detail of each component in their order in the interlinking process.

2.1 Property alignment generator

This component creates the property alignments between R_{source} and R_{target}. A property alignment is expected to describe the same attribute. ScSLINT considers only the properties satisfying the requirement of coverage, discriminability

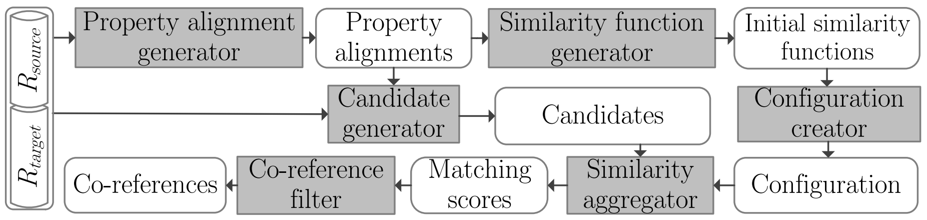

Fig. 1. The architecture of ScSLINT.

[5], and type compatibility (*string, decimal, date, URI*). An overlap measure on the tokens of the values described by the properties is used as the confidence of each alignment. The computation is as follows:

$$conf([p_{source}, p_{target}]) = \frac{|O_{p_{source}} \cap O_{p_{target}}|}{|O_{p_{source}}|} \tag{1}$$
$$O_{p_k} = \{E(o) | x \in R_k, < s, p_k, o > \in x\}$$

where $< s, p, o >$ stands for a RDF triple and E is a preprocessing function that extracts the tokens or normalizes the data format (e.g., date, time, and decimal) of the given RDF objects. ScSLINT is more scalable than SLINT+ because it does not need to consider all elements of $O_{p_{target}}$ as SLINT+ does.

2.2 Similarity function generator

This component uses the property alignments generated previously to create a list of initial similarity functions. A similarity function is specified by two pieces of information: a property alignment $[p_{source}, p_{target}]$ and a similarity measure S. For two instances $x \in R_{source}$ and $y \in R_{target}$, a similarity function calculates the similarity $S(value(x, p_{source}), value(y, p_{target}))$ where $value(a, p)$ extracts all RDF objects of the triples declared by p. Compared to SLINT+, this new module enables user to install any new similarity measure into ScSLINT.

2.3 Candidate generator

The mission of the candidate generator is to reduce the huge number of pairwise alignments between instances, at $|R_{source}| \times |R_{target}|$ pairs. This component detects the candidates of potentially co-referent instances. SLINT+ computes the rough similarity of instances using a weighted matrix structure, which is not scalable and inaccurate on ambiguous data. We recommend using token-based prefix blocking without weighting and currently install it in ScSLINT by default.

2.4 Configuration creator

A configuration contains the parameters for further components. It describes similarity functions, similarity aggregator, and co-reference filter. If no intervention (e.g., user and configuration learning algorithms) to this component is declared, a default configuration will be construct by taking all generated similarity functions and picking the frequently used similarity aggregator and co-reference filter.

2.5 Similarity aggregator

In this component, the similarity functions are executed and their results are accumulated into one final matching score. Currently, for computing the matching score of two instances x and y, ScSLINT supports the following equation:

$$score_{F_{sim}}(x,y) = \frac{1}{valid(U_{F_{sim}}(x,y))} \sum\nolimits_{v \in U_{F_{sim}}(x,y)} v^k \times weight(y) \quad (2)$$
$$U_{F_{sim}}(x,y) = \{sim(x,y) | sim(x,y) \geq \sigma_{sim}, sim \in F_{sim}\}$$

where F_{sim} is the similarity functions specified by the configuration. $k \in \{1,2\}$ controls the transformation for each similarity v. $weight$ is a function weighting the target instance, which is $\log_{\max_{t \in R_{target}} size(t)} size(y)$, where $size(y)$ counts the number of RDF triples of y. $valid$ returns the number of elements in $U_{F_{sim}}(x,y)$. σ_{sim} is the acceptance threshold for the respective similarity function sim.

This is the most expensive component in the interlinking process. However, since the candidates can be read and processed independently, ScSLINT applies parallel processing technique for this component.

2.6 Co-reference filter

This component uses the matching scores of the candidates to construct the final co-references. In this step, acceptance threshold δ is applied onto the matching scores in order to remove candidates with low similarity. In addition, for highly ambiguous data, filtering is recommended in order to obtain the high quality co-references. ScSLINT currently supports stable filtering, which applies the idea of *stable marriage* problem [4].

Implementation note. In order to optimize the memory load, ScSLINT uses pre-indexed structures of input RDF repositories. The complexity of indexing algorithm is small and is linear to the size of repository. Note that these indexes are created only one time and are reusable. ScSLINT is developed in C++. The source code this framework is available at http://ri-www.nii.ac.jp/ScSLINT.

3 Performance

We evaluate ScSLINT on a computer equipped with one Intel core i7 4770K CPU and 8GB memory. We enable multi-threading for the similarity aggregator. We test ScSLINT on 7 real datasets with very large size. The source repositories are three subsets of NYTimes, whose domain are locations (nyt.loc), organizations (nyt.org), and people (nyt.peo). The target repositories are Dbpedia (db), Freebase (fr), and Geonames (gn). We use the default token blocking, linear similarity aggregator ($k = 1$), and two complex similarity measures for strings, TF-IDF Cosine and Levenshtein. We use reverse distance for numeric values and exact matching for other data types. Using this configuration, in average, 90.57% of actual co-references are successfully detected.

Table 1 reports the complexity of these datasets and the execution time of each component of ScSLINT. In this table, the columns from 2 to 4 describe

Table 1. Complexity (column 2 to 4) and processing time in second (column 5 to 8).

Dataset	Size ($\times 10^9$)	Candidates ($\times 10^6$)	Similarity functions	Comp. 1	Comp. 3	Comp. 5	Comp. 6
nyt.loc-gn	32.69	32.2	12	37	7	70	3
nyt.loc-db	16.06	38.2	25	43	8	268	6
nyt.org-db	25.47	61.7	17	46	11	404	6
nyt.peo-db	41.66	46.9	22	46	11	251	6
nyt.loc-fr	154.97	222.7	23	14	111	641	29
nyt.org-fr	245.70	357.4	16	14	268	1023	46
nyt.peo-fr	401.89	620.1	18	15	507	1578	78

the parameters that make impacts to the complexity of the interlinking process. Column 2 is $|R_{source}| \times |R_{target}|$, which reflects the complexity of property alignment generator (Comp. 1) and candidate generator (Comp. 3). The multiplication of column 3 and 4 is the number of comparisons, which directly defines the complexity of the similarity aggregator (Comp. 5) and co-reference filter (Comp. 6).

In our experiment, while ScSLINT achieves a very impressive speed, LIMES and SILK fail to finish the interlinking task even within 10 times longer period for each dataset (and we terminate those programs in this case). The longest interlinking time is about 36 minutes, recorded on nyt.peo-fr, which requires 11.16×10^9 comparisons. On a dataset that is about 2000 times smaller than nyt.peo-fr, the required time for LIMES and SILK are almost 30 minutes and 9.5 hours, respectively [2].

ScSLINT can process large data in very short time when running on a usual personal computer. That is, the framework expresses its compatibility for even huge scale datasets. ScSLINT can far benefit from being conjuncted with other big data processing framework or deployed on a powerful machine. Also, more advanced candidate generators, similarity metrics, aggregation strategies, and learning algorithm can be implemented in ScSLINT easily.

References

[1] Ferrara, A., Nikolov, A., Scharffe, F.: Data linking for the semantic web. International Journal of Semantic Web and Information System 7(3), 46–76 (2011)

[2] Ngomo, A.C.N., Auer, S.: LIMES: A time-efficient approach for large-scale link discovery on the web of data. In: 22nd IJCAI. pp. 2312–2317 (2011)

[3] Nguyen, K., Ichise, R.: SLINT+ results for OAEI 2013 instance matching. In: 8th ISWC workshop on Ontology Matching. pp. 177–183 (2013)

[4] Nguyen, K., Ichise, R., Le, B.: Interlinking linked data sources using a domain-independent system. In: 2nd JIST. LNCS, vol. 7774, pp. 113–128. Springer (2013)

[5] Song, D., Heflin, J.: Automatically generating data linkages using a domain-independent candidate selection approach. In: 10th ISWC. LNCS, vol. 7031, pp. 649–664. Springer (2011)

[6] Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Discovering and maintaining links on the web of data. In: 8th ISWC. LNCS, vol. 5823, pp. 650–665. Springer (2009)