Preprint of C. Lantzaki, P. Papadakos, A. Analyti, Y. Tzitzikas, Radius-aware Approximate Blank Node Matching using Signatures, Journal Knowledge and Information Systems (KAIS) (accepted for publication in 2016).

KAIS manuscript No.
(will be inserted by the editor)

Radius-aware Approximate Blank Node Matching using Signatures

Christina Lantzaki · Panagiotis Papadakos · Anastasia Analyti · Yannis Tzitzikas

Received: date / Accepted: date

Abstract In the Linked Open Data (LOD) cloud, the biggest open data graph that currently exists, a remarkable percentage of data are *unnamed* resources, also called blank nodes. Several fundamental tasks, such as graph isomorphism checking and RDF data versioning, require computing a map between the sets of blank nodes of two graphs. This map aims at minimizing the delta size, i.e. the number of change operations that are required to make the graphs isomorphic. Computing the optimal map is NP-Hard in the general case and various approximation algorithms have been proposed. In this work we propose a novel radius-aware signature-based algorithm, that is not restricted to the direct neighbourhood of the compared blank nodes. Contrary to the older algorithms, the proposed algorithm manages to decrease the deviation from the optimal solution even for graphs that contain connected blank nodes in large and dense structures. The conducted experiments over real and synthetically generated datasets (including datasets from the Billion Triple Challenge 2012 and 2014) show the significantly smaller deltas. For isomorphism checking (simple RDF equivalence), with a wise configuration of radius, the proposed algorithm achieves optimality for 100% of the datasets, while in non-isomorphic datasets the deltas are on average $50\% - 75\%$ smaller than those of the previous algorithms. Finally, the trade-off between radius, deviation from the optimum and time efficiency, is analyzed.

Christina Lantzaki
Computer Science Department, University of Crete, and
Institute of Computer Science, FORTH-ICS, GREECE
E-mail: kristi@ics.forth.gr

Panagiotis Papadakos
Computer Science Department, University of Crete
E-mail: papadako@ics.forth.gr

Anastasia Analyti
Institute of Computer Science, FORTH-ICS, GREECE
E-mail: analyti@ics.forth.gr

Yannis Tzitzikas
Computer Science Department, University of Crete, and
Institute of Computer Science, FORTH-ICS, GREECE
E-mail: tzitzik@ics.forth.gr

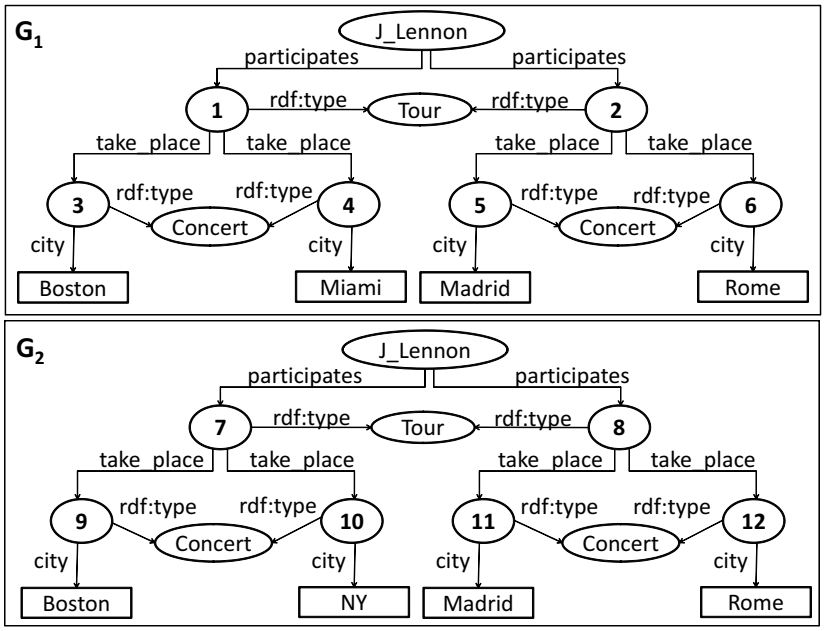
Keywords RDF, blank node matching, delta minimization, equivalence, isomorphism

1 Introduction

According to empirical surveys conducted in [22,13] about RDF data, a notable percentage of unique terms (25.7%) are blank nodes (for short bnodes), proving that they are prevalent in real-world data. Bnodes are useful for representing complex attributes, for describing multi-component structures (like RDF containers), for representing reification and provenance information [4], as well as for representing OWL classes defined by expressions (like unions, intersections, and others). However, due to their anonymity (i.e. lack of a global identifier), their existence requires special treatment in various web data management tasks [29,3, 23,17,28]. In particular, tasks such as RDF graph *isomorphism checking* [3], RDF *differential storage* [29,23,17,2,28,30], RDF data *synchronization* [26] and *notification* [19], RDF *entailment* [24,13] and data integration [7,8] require matching the participating bnodes (for a consolidated list of concrete tasks that require bnode matching see [20]). Moreover bnode matching can be useful for entity matching in the context of a semantic warehouse (e.g. such as [27]). The current work focuses on the first three of the aforementioned tasks. In particular, given two graphs, these tasks require a *differential task*, i.e. the comparison of the graphs and the computation of the change operations that need to be applied to the first graph to make it isomorphic with the second one (i.e. computation of delta). Computing a map between the sets of bnodes of the two graphs can be considered as a preprocessing task that is carried out before the differential task, contributing to a reduced delta size (i.e. number of change operations). [28] proved that computing the optimal bnode map, or else the map that minimizes the delta size, is NP-Hard in the general case and polynomial in the case where there are no directly connected bnodes. For the general case that work proposed various approximation algorithms, that match bnodes based only on information located in their direct neighbourhood. However, in some cases this information is not enough to compute a satisfying bnode map, e.g. in case of connected bnodes with identical information around them. In such cases the larger and denser the structures of bnodes are, the bigger the delta size of these algorithms might become.

To tackle this problem in this paper we propose a new radius-aware approximate matching approach. Since there is a need for a fast bnode matching method, we propose an algorithm that extends and generalizes the $O(n \cdot \log n)$ *signature-based* algorithm [28], where n is the max number of bnodes of the two graphs. That algorithm constructs a signature (i.e. string) for each bnode; thereby each graph is associated with a set of signatures of its bnodes. The signatures in these two sets are then compared and a mapping is computed based on string equality, or lexicographic distance in case of inequality. A key observation is that the signature of a bnode is constructed based on the triples that it participates, i.e. the information that is located in a radius equal to 1. Thereby, the matching decisions are based only on the information located in the direct neighborhoods of the bnodes.

In this work, we generalize and propose an algorithm that enables the matching of bnodes based on triples located in higher radii, in an attempt to further reduce the delta size (i.e. reduce the deviation from the optimal delta). To grasp the idea,



	M	$\lvert\Delta_e(G_1 \to G_2, M)\rvert$
No Matching	∅	32
SIGN	{(1,7) (2,8) (3,9) (4,10) (5,11) (6,12)}	2
	{(1,8) (2,7) (3,9) (4,10) (5,11) (6,12)}	10
r-SIGN	{(1,7) (2,8) (3,9) (4,10) (5,11) (6,12)}	2
Optimal	{(1,7) (2,8) (3,9) (4,10) (5,11) (6,12)}	2

Fig. 1: Running example

consider the graphs G_1 and G_2 of Fig. 1. Both represent information about two tours of John Lennon each having two concerts. Notice that each graph contains 6 bnodes. Comparing them without bnode matching results in 16 triple deletions and 16 triple additions to turn them equal (i.e. all the triples of G_1 that contain bnodes should be deleted and all the triples of G_2 that contain bnodes should be added); in other words the delta contains 32 change operations. However, by applying bnode matching, the delta will contain less or equal to 32 change operations. In the optimal case, where the bnodes are matched as the following pairs indicate $\{(1,7),(2,8),(3,9),(4,10),(5,11),(6,12)\}$, only 2 change operations are needed (i.e. the triple $(4, city, Miami)$ is deleted and the triple $(10, city, NY)$ is added).

The signature-based algorithm in [28] does not ensure optimality when applied in this example; notice that bnodes 1 and 2 of graph G_1 and 7 and 8 of graph G_2 have the same direct neighborhoods; thereby they are matched randomly. As a result, the algorithm returns either 10 or 2 change operations, depending on the selected map between their bnodes (see more in Section 2.3). In this paper we propose an algorithm that considers the connectivity of the bnodes and takes information in higher radii into account offering a more refined signature. For the running example, it always manages to find the optimal solution.

In particular, in this paper we firstly give the signature-based algorithm in a more accurate and enriched manner than it was given in [28] paving the way for the new algorithm. Afterwards, we propose a new *radius-based* matching approach and a new *radius signature-based* algorithm. The new algorithm is enriched with a new parameter, called radius. The value of the radius is configurable and affects the deviation from the optimal and the computational complexity. The proposed algorithm is founded on a theoretical analysis and is based on three types of matching relations. The properties of these relations are analyzed and exploited for describing the behavior of the approximation algorithm in terms of optimality and time efficiency.

The results of the experimental evaluation showed that over synthetically generated isomorphic datasets, the proposed algorithm manages to find the optimal solution in all cases. In non-isomorphic datasets the delta sizes were found to be 50%-75% smaller than those of the previous *signature-based* algorithm. We also report results regarding the effectiveness of the proposed algorithm over Billion Triple Challenge (BTC) Datasets [11, 14]. A wise configuration of radius can: a) attain optimality for isomorphism checking; indicatively optimality is achieved for the 100% of the checked BTC datasets, and b) guarantee a time efficient algorithm as shown in our experiments. It is worth noticing that increasing the radius reduces the deviation from the optimal, but on some occasions even increases the time efficiency. Therefore, the configuration of the radius according to graph statistics is of great importance.

In a nutshell, this paper: (a) provides a rigorous description of the *signature-based* algorithm, (b) proposes a novel *Radius-Signature-based* algorithm that can match signatures in the desired radius, and (c) reports comparative results with other bnode matching algorithms over a wide spectrum of datasets. The results show that the proposed algorithm offers a better trade off between the deviation from the optimal solution and the time efficiency, especially for graphs that contain bnodes in large and dense structures.

Finally, we should note that although we elaborate on this problem in the context of RDF, elements which are unnamed (i.e. with no externally referable identity) occur in several representation frameworks and contexts, e.g. in database modeling we have weak entities, in relational tables we have keys produced automatically by the DBMS (if no other attribute or attributes can serve as global identity), in programming language we have unnamed constructs (e.g. the unnamed classes in Java). Consequently also in such frameworks and contexts, tasks like equivalence or differential storage require ability to match unnamed elements. To this end the proposed algorithm could be implemented in these contexts, or the corresponding artifacts (ER diagrams, relational databases) could be rewritten in RDF and then the proposed algorithm can be applied straightforwardly.

The rest of this paper is organised as follows: §2 describes the required background and the previous *signature-based* algorithm, while §3 presents the radius-based approach. §4 reports and discusses the experimental results, and finally, §5 concludes the paper. Proofs are given in the Appendix.

2 Background and Related Work

At first, we introduce the required notions and notations (§2.1), discuss related works (§2.2), and then describe the signature-based algorithm in more detail (§2.3 - §2.4).

2.1 Preliminaries

Consider three pairwise disjoint infinite sets U (URIs), L (literals) and B (bnodes). An RDF triple is a tuple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$, where s is the subject, p is the predicate and o is the object. An RDF Knowledge Base (KB), K, or equivalently an RDF graph G is a finite set of RDF triples. For an RDF graph G_1 we shall use U_1, B_1, L_1 to denote its URIs, bnodes and literals resp. The *nodes* of G_1 are the values that appear as subjects or objects in the triples of G_1, denoted as $nodes(G_1)$.

We shall call *map* M an injective partial function M: $U \cup B \cup L \rightarrow U \cup B \cup L$, such that it is the identity function for URIs and literals, i.e. $M(u) = u$ for all $u \in U \cup L$, while each bnode can be mapped to any other bnode.

The isomorphism of *RDF graphs* is defined in [6] as:

Def. 1 *Isomorphism of RDF Graphs*
Two RDF graphs G_1, G_2 are *isomorphic*, denoted as $G_1 \equiv G_2$, if there is a bijective function m between the sets of nodes of the two graphs (N_1 and N_2), s.t.: $m(u)$=u for all $u \in U_1 \cap N_1$, and $m(l) = l$ for all $l \in L_1$, and finally, m maps bnodes to bnodes (i.e. for all $b \in B_1$ it holds $m(b) \in B_2$) so that it holds that a triple (s, p, o) $\in G_1$ if and only if a triple ($m(s)$, p, $m(o)$) $\in G_2$. □

Notice that in this case (isomorphism of RDF graphs), m is a map M, which is a function (total instead of partial) and is not only injective but also surjective, (i.e. bijective), as $|U_1| = |U_2|$, $|L_1| = |L_2|$ and $|B_1| = |B_2|$.

For the general context of comparing two RDF graphs, we first give the definition of the *explicit delta* (ignoring the special semantics of bnodes).

Def. 2 *Explicit Delta*
The *explicit delta* from G_1 to G_2, denoted by $\Delta_e(G_1 \rightarrow G_2)$ is the set of deletions and additions of triples that have to be applied over the graph G_1 in order to make it equal to G_2. Formally: $\Delta_e(G_1 \rightarrow G_2) =$
$\{Del((s,p,o)) \mid (s,p,o) \in G_1 \ \wedge \ (s,p,o) \notin G_2\} \ \cup$
$\{Add((s,p,o)) \mid (s,p,o) \in G_2 \ \wedge \ (s,p,o) \notin G_1\}$. □

Under the existence of bnodes in the two graphs *Explicit Delta* is not sufficient. The special treatment that bnodes should have is evident in the following definitions. We shall denote the *application* of a map M over a triple (s, p, o) as $(s, p, o)_M$ which is defined as: i) (s, p, o) itself if $M(s), M(o)$ are undefined, or ii) $(M(s), p, o)$ if $M(s) \in range(M)$, $M(o)$ is undefined or iii) $(s, p, M(o))$ if $M(o) \in range(M)$, $M(s)$ is undefined or iv) $(M(s), p, M(o))$ if $M(s), M(o) \in range(M)$.

Def. 3 *BNode-Aware Explicit Delta*
The *explicit delta* from G_1 to G_2 according to map M, denoted by $\Delta_e(G_1 \rightarrow G_2, M)$ is the set of deletions and additions of triples that have to be applied over

the graph G_1 in order to make it isomorphic to G_2, while considering the map M. Formally: $\Delta_e(G_1 \rightarrow G_2, M) =$
$\{Del((s,p,o)) \mid (s,p,o) \in G_1 \ \wedge \ (s,p,o)_M \notin G_2\} \ \cup$
$\{Add((s,p,o)) \mid (s,p,o) \in G_2 \ \wedge \ (s,p,o)_{M^{-1}} \notin G_1\}$.
where M^{-1} is the inverse of M (which is a function since M is injective). □

Notice that in any case *Explicit Delta* (*BNode aware* or not) ensures the soundness of the given output [30]. In other words, applying on G_1 the change operations of $\Delta_e(G_1 \rightarrow G_2, M)$ for any map M, we get a graph G_1' that is isomorphic to G_2.

The actual challenge lies on the process of *bnode matching* between the bnodes of the two graphs. Therefore, for notational simplicity, we shall hereafter use M to refer only to the map from B_1 to B_2, i.e. M: $B_1 \rightarrow B_2$. In the context of a comparison between two graphs G_1 and G_2, the authors of [28] formulate the problem of finding the optimal map as the problem of computing a map M that minimizes $\Delta_e(G_1 \rightarrow G_2, M)$. This map will be denoted as M_{opt}.

For the case where G_1 and G_2 are isomorphic, there is a map $M = M_{opt}$, such that $\Delta_e(G_1 \rightarrow G_2, M) = \emptyset$.

Using the above notations over the running example, without bnode matching, i.e. with $M = \emptyset$ we get $|\Delta_e(G_1 \rightarrow G_2, M)| = 32$. On the other hand, if $M = M_{opt}$ $= \{(1,7),(2,8),(3,9),(4,10),(5,11),(6,12)\}$ we get that $|\Delta_e(G_1 \rightarrow G_2, M)| = 2$.

2.2 Related Work

Tasks like RDF graph isomorphism checking [3] (usually offered in RDF Editors and APIs), RDF differential storage [29,23,17,2,28,30] (for building versioning systems), RDF data synchronization [26] (for applications that support synchronization), and notification [19], RDF entailment [24,13] and data integration [7, 8] require matching the participating bnodes. For a consolidated list of concrete tasks that require bnode matching see [20]. These tasks correspond to functionalities and services that are offered by various existing systems. For instance isomorphic checking is usually offered by APIs for managing RDF (e.g. Apache Jena[1]), while versioning services are offered by various triplestores (like KiWi triplestore[2]). Below we discuss the more related works on bnode matching.

In CWM[3] [2] the authors have proposed an algorithm that computes a delta between two RDF graphs giving the ability to match bnodes between them. The matching requires to assign labels to bnodes in order to uniquely identify them through functional (or inverse functional) properties. Jena [3] treats the problem of comparing RDF graphs using the already proposed algorithms for graph isomorphism. However, it is limited to answering if the graphs are isomorphic or not and does not compute any delta between them. In [29] the authors propose a structural and semantic-based diff algorithm, that treats bnodes by introducing bnode enrichment, a procedure that adds functional properties to each bnode leading to a URI. This procedure ensures matching bnodes that participate in the exact same triples, since they get the same identity. However, in the context of a system where

[1] `https://jena.apache.org/`
[2] `http://marmotta.apache.org/kiwi/`
[3] `https://www.w3.org/2000/10/swap/doc/cwm.html`

changes arise, even a small modification in one of the triples implies different identifiers, meaning that bnodes cannot be mapped (closest match is not supported). PromptDiff [23] is an ontology-versioning tool that finds a structural diff between two versions, applying an arbitrary number of heuristic matchers, matching also bnodes. Ontoview [18] also uses heuristic matchers to match bnodes. In [12] Hogan proposes a method to offer canonical labels for blank nodes such that they can be identified globally. However, that work aims exclusively in preserving isomorphism in RDF graphs and does not treat differential issues.

Generally, the aforementioned works do not offer any formulation of the problem based on the need for optimal bnode matching. [28] is the first paper that addresses theoretically this problem as an optimization problem. It proves that computing the optimal map is NP-Hard in the general case and proposes two approximation algorithms. The first algorithm, `HUNG`, uses a metric to compute the edit distance between two bnodes and applies the Hungarian algorithm to return a map in $O(n^3)$ time. The second is the *signature-based* algorithm, `SIGN`, that can compare large graphs in $O(n \cdot \log n)$ time, where n is the number of bnodes of the graph. In this work, we propose a radius-aware approach that extends and generalizes the *signature-based* algorithm.

2.3 The Signature-based Algorithm

Here we describe the *signature-based* algorithm, denoted by `SIGN`, that is already given in [28]. In order to create a stable background for the new algorithm, we present `SIGN` in a more explicit way than that in [28] and make some slight but significant changes that will be pointed out later. Let us first define the *Direct Neighborhood Graph* and the signature of a bnode.

Def. 4 *Direct Neighborhood Graph*
The *Direct Neighborhood Graph* of a bnode b of a graph G, denoted as $DNG(b, G)$, is defined as $DNG(b, G) = Class(b) \cup In(b) \cup Out(b)$, where

$$\begin{aligned} Class(b) &= \{(b, \texttt{rdf:type}, o) \in G\} \\ In(b) &= \{(s, p, b) \in G\} \\ Out(b) &= \{(b, p, o) \in G\} \setminus Class(b) \end{aligned}$$ □

The *Direct Neighborhood Graph* of a bnode b actually contains all the triples that bnode b participates. These can be classified into three categories, $Class(b)$ with the triples that define the class of the bnode, $In(b)$ with the incoming triples of b and $Out(b)$ with the outgoing triples of b.

The building block of the algorithm is the *signatures* of the bnodes in the participating graphs. The *signature* of each bnode b actually serializes all the triples of $DNG(b)$ in a specific order. For instance, the signature of bnode 3 of our example is "rdf: type△Concert♢take_place△ •1♢city△Boston", where ♢ is a delimiter between information coming from different triples of $DNG(b)$, △ is a delimiter between the predicates and the terms, and • denotes the existence of a bnode. Specifically, each adjacent bnode of b gets a counter that is represented in the signature by a • followed by that number. Counters are assigned by lexicographic ordering (the ordering for the radius-based algorithm will be explained in §3.2). In order to formally define it, let us first define the *label* of a triple in terms of a bnode (below we use "+" to denote string concatenation):

Def. 5 *Label of a triple in terms of a bnode*
The *label* of a triple $t = (s, p, o) \in G$ in terms of a bnode $b \in B$, denoted as $l_b(t)$, is the string defined as:

$$l_b(t) = \begin{cases} p + \triangle + s & , b = o \wedge s \notin B \\ p + \triangle + o & , b = s \wedge o \notin B \\ p + \triangle + \text{“}\bullet\text{”} & , (b = o \wedge s \in B) \vee (b = s \wedge o \in B) \\ \text{“”} & , (b \neq s \wedge b \neq o) \end{cases} \quad \square$$

Indicatively, for the graph G_1 of Figure 1 we have $l_3((3, city, Boston))$=“$city\triangle Boston$” while $l_1((1, take\_place, 3))$=“$take\_place\triangle\bullet$”.

We can now define the *signature* of a bnode b, denoted by $sign(b)$. It is a string that comprises three parts: *Class(b)*, *In(b)* and *Out(b)*. The first corresponds to a serialization of the `rdf:type` triples of b, the second to a serialization of the incoming triples to b, and the last to a serialization of the outgoing triples of b respectively. The selected order of the three parts of the signature, namely $\langle Class(b), In(b), Out(b)\rangle$ is based on the probability of the participating triples of each set to change between different versions of the graph. The motivation for this ordering follows:

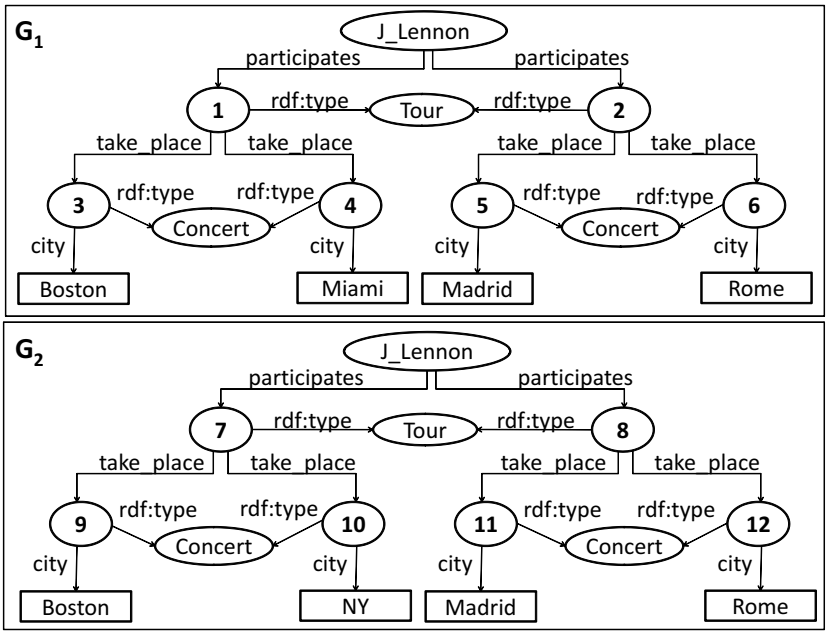
- According to a survey over BTC Datasets in [22], each bnode occurs on average 19% in the object position and 81% in the subject position of a non-rdf:type triple. Thereby, there is more probability for an outgoing than an incoming triple to change over versions.
- In versions of the datasets with data-level changes, it is less probable for the rdf:type of a resource to change.
- In RDF/XML, Turtle and N3 syntax by definition only *tree* BComponents can occur, when bnode identifiers are not used. Thereby, each bnode can participate in exactly one incoming triple, but in multiple outgoing triples.

Such a classification and ordering tries to construct signatures that will allow good matches to the binary search algorithm.

Returning to signature construction, and for grasping the idea, the second column of the tables in Figure 2b, shows the signatures of all bnodes of the running example. Notice that two or more bnodes may have the same signature, like bnodes 1 and 2 or 7 and 8 of the example. This is expected, since these bnodes have equivalent *Direct Neighborhood Graphs*. The last column, with heading “signBN(b)”, will be exploited by the new algorithm and it is explained in §3.2. The exact algorithm that produces $sign(b)$ is Algorithm 5 and is given in the Appendix A.

Now the algorithm that is responsible for the *matching* of the bnodes is given in Alg. 1 and is denoted by `SIGN`. It takes as input two graphs, G_1 and G_2, and outputs a map, denoted as M_{SIGN}.

In lines 4-8 algorithm `SIGN` performs matches between pairs of bnodes $(b_1, b_2) \in B_1 \times B_2$ that have the same signatures (i.e. *Exact Matches*), stores them in M_{SIGN} and updates the sets with the not yet matched bnodes. For the remainder bnodes of B_1, B_1', *Closest Matches* are performed with the remainder bnodes of B_2, B_2' (lines 9-12). In particular, the function $closestMatch(b_i, B_2')$ performs a lookup of each $sign(b_i)$ in the lexicographically ordered set $\{sign(b_j) \mid b_j \in B_2'\}$ using binary search. The bnode $b_j \in B_2'$ to which binary search stopped (while searching for b_i) is matched to b_i. Notice that the *Exact* matches precede the *Closest* matches,



(a) Graphs of the running example

graph G_1		
b	**sign(b)**	**signBN(b)**
3	rdf:type△Concert◊take_place△•1◊city△Boston	[1]
5	rdf:type△Concert◊take_place△•1◊city△Madrid	[2]
4	rdf:type△Concert◊take_place△•1◊city△Miami	[1]
6	rdf:type△Concert◊take_place△•1◊city△Rome	[2]
1	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[3, 4]
2	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[5, 6]

graph G_2		
b	**sign(b)**	**signBN(b)**
9	rdf:type△Concert◊take_place△•1◊city△Boston	[7]
11	rdf:type△Concert◊take_place△•1◊city△Madrid	[8]
10	rdf:type△Concert◊take_place△•1◊city△NY	[7]
12	rdf:type△Concert◊take_place△•1◊city△Rome	[8]
7	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[9, 10]
8	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[11, 12]

(b) Signatures of bnodes

Fig. 2: Signatures of bnodes over the running example

in order to avoid the case where a *Closest* match deters an *Exact* match at a later step.

Algorithm 1: `SIGN`

Input: graphs G_1 and G_2
Output: map M_{SIGN}

$M_{SIGN} \leftarrow \emptyset$
$B'_1 \leftarrow B_1$
$B'_2 \leftarrow B_2$
foreach $b_i \in B_1$ **do** ▷ Exact Matches
  if $\exists\ b_j \in B'_2$ *s.t.* $sign(b_i) = sign(b_j)$ **then**
    $M_{SIGN} \leftarrow M_{SIGN} \cup \{(b_i, b_j)\}$
    $B'_1 \leftarrow B'_1 \setminus \{b_i\}$
    $B'_2 \leftarrow B'_2 \setminus \{b_j\}$
foreach $b_i \in B'_1$ **do** ▷ Closest Matches
  $b_j \leftarrow closestMatch(b_i, B'_2)$
  $M_{SIGN} \leftarrow M_{SIGN} \cup \{(b_i, b_j)\}$
  $B'_2 \leftarrow B'_2 \setminus \{b_j\}$
return M_{SIGN}

2.4 Limitations of the Algorithm

The algorithm `SIGN` manages to compute a quite satisfying map (as regards its deviation from the optimum) in $O(n \cdot \log n)$ time, as both *Exact* and *Closest Matches* are performed using binary search of the elements of B_1 over the elements of B_2. Indicatively, `SIGN` can match $150,000$ pairs of bnodes in less than 10.5 secs. However, for graphs with dense structures of bnodes the algorithm may not be effective. For instance in the experiments reported in [28] it did not manage to detect equivalences in graphs with high density of bnodes. There are two main limitations in `SIGN`, as regards optimality. The first limitation is in the *Matching* Phase, where the signatures are matched according to their lexicographic proximity. This method ensures $O(n \cdot \log n)$ time complexity (where $n = \max(|B_1|, |B_2|)$). In particular, the key point as regards the efficiency is the serialized signatures, and their subsequent lexicographic sorting which in turn enables the application of binary search. Obviously, the construction of the signature of a bnode is important and the challenge does not only lie on selecting which information to encode in the signature, but also on selecting the order of the selected information inside the signature. Inevitably, the matching of bnodes according to this serialization of the knowledge is not perfect, since the decision for a matching lies mostly on the sameness of the first parts of the signatures. Nevertheless, the triples are serialized in such a way that triples with lower probability to change are represented in the first part of the signature. Alternative options would increase the time and probably space complexity of the matching algorithm, and go beyond the scope of the current paper.

Based on the above, in this paper we focus on the second limitation lying at the definition of *Signature*. Recall that the signature of a bnode is constructed using information in its *Direct Neighborhood*. As the structures between the bnodes become larger and denser, the signature of each bnode cannot capture the necessary information that will ensure its discreteness and therefore we may end up with less accurate mappings. Indicatively, for our running example bnodes 1, $2 \in G_1$ and 7, $8 \in G_2$ have the same signatures. The final map will contain the mappings $\{(3,9)(5,11)(4,10)(6,12)\}$ and either $\{(1,7)(2,8)\}$ or $\{(2,7)(1,8)\}$. The

decision between these two different maps is taken randomly (i.e. according to the specific repository's serialization order of the bnodes). In the first case the differential function returns the optimal solution (i.e. $|\Delta_e| = 2$), while in the second case it returns $|\Delta_e| = 10$ (i.e. 5 deletions and 5 additions of triples). In the following section we propose the *radius-aware signature-based* algorithm that can give signatures with better discrimination ability and thus offer a map closer to optimal.

3 The Radius-based Matching Approach

Let us now introduce the *Radius-Signature-based* approach. At first, we introduce the required definitions (§3.1), discuss the main idea (§3.2), and then describe the new algorithm (§3.2.1). Afterwards (in §3.3) we provide some properties of the algorithms, then (in §3.4) we describe its time complexity, and finally (in §3.5) we discuss its expected performance (best, worst cases, deviation from optimum).

3.1 Definitions

The radius-based approach imposes the need for new types of matches between the bnodes. We first introduce the notion of *BComponent*, and the notion of *radius-based neighborhood graphs*, and then we exploit them for defining *three kinds of matching* relations.

A triple is called *ground* if it does not contain any bnodes, *btriple* if it contains exactly one bnode (either as subject or object) and *bbtriple* if it contains two bnodes. Now we can give the definition of a *BComponent*.

Def. 6 *BComponent*
Each maximal set of connected *bbtriples* of an RDF graph G forms a sub-graph, called a *BComponent* of G. □

For example, Figure 1 has four BComponents, one of them is $\{(1, take\_place, 3), (1, take\_place, 4)\}$. BComponents are classified in *tree* and *graph*, depending on whether they form tree or graph structures respectively. The following holds:

Prop. 1 *Bnodes in BComponents*
Each bnode $b_1 \in B_1$ is part of exactly one BComponent of G_1, which is denoted as $BC(b)$ (obviously $BC(b) \subseteq G_1$). □

The *diameter* of a BComponent BC, denoted by $diam(BC)$, is defined as the largest number of nodes which must be traversed in order to travel from one node to another when paths which backtrack, or loops are excluded from consideration [10]. In the context of one graph G we define the set:

$$diam(G) = \{diam(BC_i) \mid BC_i \subseteq G\},$$

and then define the *mean value* of the diameter of G, denoted by $\overline{diam(G)}$ and the maximum value denoted by $max_{diam(G)}$.

The *Radius-Signature-based* algorithm that we will introduce will leverage information in distance up to radius r around each bnode b, instead of staying restricted to $DNG(b)$ (as the previous signature algorithm). This information forms

a subgraph of G which is called *Radius Neighborhood Graph* of b in radius r, denoted by $RNG(b, r)$, defined as:

Def. 7 *Radius Neighborhood Graph*
The *Radius Neighborhood Graph* of a bnode $b \in B$ of a graph G with radius r ($r \geq 1$), denoted by $RNG(b, r, G)$, is defined as the set of triples located in paths of the undirected version of G, where each path:
(a) starts from the bnode b, and
(b) has length equal to n with either n *bbtriples*, or $n-1$ *bbtriples* followed by one *btriple*, where $n \leq r$.

Specifically:

$$RNG(b, r, G) = \begin{cases} \bigcup\{\ DNG(b', G) \mid b' \in nodes(RNG(b, r-1, G)) \cap B\} & , r > 1 \\ DNG(b, G) & , r = 1 \end{cases}$$

It follows that the *Radius Neighborhood Graph* of bnode b with $r = 1$ is the *Direct Neighborhood Graph* of b (Def. 4), while for bigger radii it is the union of the *Direct Neighborhood Graphs* of the bnodes that are included in a path of length up to $r-1$, that starts from b and contains only bnodes.

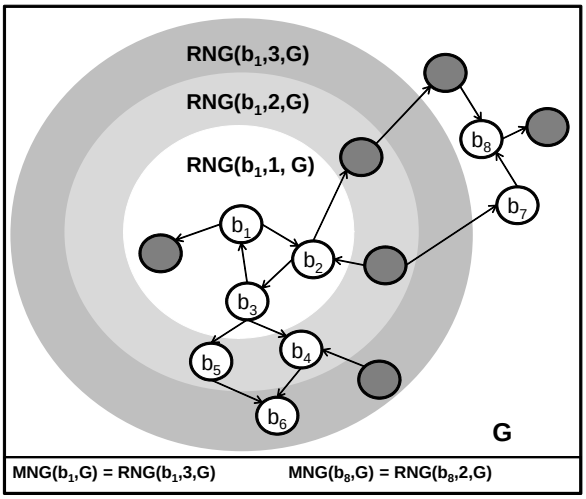


Fig. 3: Radius Neighborhood Graphs of b_1

Figure 3 depicts the *Radius Neighborhood Graphs* of bnode b_1 for various radii in a graph G where its bnodes are white and its named nodes grey. Notice that the ground triples and any triple of the path that follows after a *btriple* are not part of any $RNG(b_1, r, G)$.

We now define the *Maximum Neighborhood Graph* as:

Def. 8 *Maximum Neighborhood Graph*
The *Maximum Neighborhood Graph* of a bnode $b \in B$ of a graph G, denoted by

$MNG(b, G)$, is the maximum element of $\{RNG(b, r, G) \mid r \geq 1\}$, i.e. $MNG(b, G) = max_{\subseteq}\{RNG(b, r, G) \mid r \geq 1\}$. □

The following properties hold for the Maximum Neighborhood Graph.

Prop. 2 *Properties of Maximum Neighborhood Graph*

p1: There exist $r_{max} \geq 1$ such that for all $r \geq r_{max}$ it holds $RNG(b, r, G) = RNG(b, r_{max}, G) = MNG(b, G)$
p2: $MNG(b, G) \supseteq BC(b)$
p3: $nodes(MNG(b, G)) \cap B = nodes(BC(b))$ □

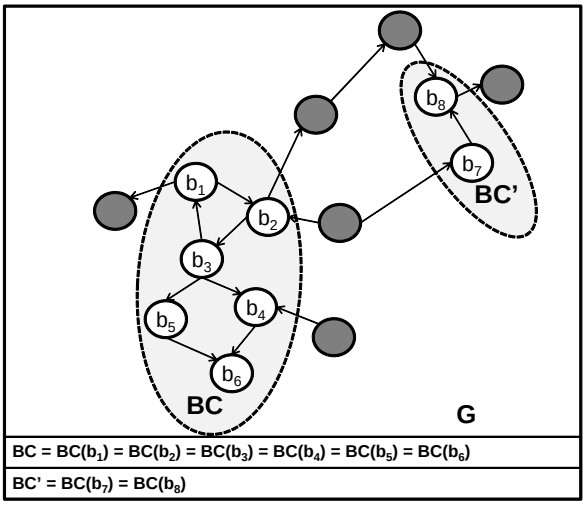


Fig. 4: BComponents of graph G

For the graph G of Figure 3, $MNG(b_1, G) = RNG(b_1, 3, G) = RNG(b_1, 4, G)$. Figure 4 shows the two BComponents of the same graph G.

The radius-based approach imposes the need for new types of matches between the bnodes. Therefore, here we define some binary relations, one for *non-Matches* and three types of *Matches* ranging from approximate "isomorphism" to isomorphism of the entire *BComponents*.

Def. 9 *Types of Matches*
For a radius $r \geq 1$ and a pair of bnodes $(b_1, b_2) \in B_1 \times B_2$ we define the following relations:

NoMatch: we say that bnodes b_1 and b_2 *do not match*, denoted by $b_1 \not\approx b_2$, iff $DNG(b_1, G_1) \not\equiv DNG(b_2, G_2)$.

Best $\frac{r'}{r}$-Match: we say that b_1 and b_2 *Best $\frac{r'}{r}$-match*, denoted by $b_1 \approx_r^{r'} b_2$, iff $RNG(b_1, r, G_1) \not\equiv RNG(b_2, r, G_2)$, but $\exists\, r'$, where $1 \leq r' < r$ s.t. $RNG(b_1, r', G_1) \equiv RNG(b_2, r', G_2)$.

Exact r-Match: we say that b_1 and b_2 *Exact r-match*, denoted by $b_1 \approx_r b_2$, iff $RNG(b_1, r, G_1) \equiv RNG(b_2, r, G_2)$.

BC r-Match: we say that b_1 and b_2 *BComponent r-match*, denoted by $b_1 \approx_r^B b_2$, iff $RNG(b_1, r, G_1) \equiv RNG(b_2, r, G_2)$, $RNG(b_1, r, G_1) = MNG(b_1, G_1)$ and $RNG(b_2, r, G_2) = MNG(b_2, G_2)$. □

Figure 5 shows the relationships between the various types of matches for radius $r = 3$. The left diagram illustrates that a BC 3-Match is an Exact 3-Match, but not the vice versa. The right diagram illustrates that a Best $\frac{2}{3}$-Match is also a Best $\frac{1}{3}$-Match, but not the vice versa. Analogous illustrations could be given for any radius r.

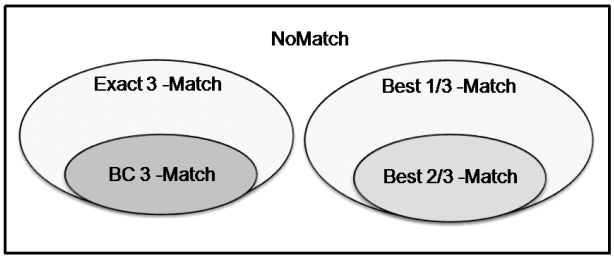


Fig. 5: Different types of Matches for radius r= 3

Here are some properties that stem from the previous definitions of matches:

Prop. 3 *Properties of Matches*

p1: $\approx_{r'} \subseteq \approx_r$ for all r, r' where $r' \geq r \geq 1$
p2: $\approx_{r'}^{r} \subseteq \approx_r$ for all r, r' where $r' \geq r \geq 1$
p3: $\approx_r^{r_b} \subseteq \approx_r^{r_a}$ for all r_a, r_b where $r_b \geq r_a \geq 1$
p4: $\approx_r^{r_b} \subseteq \approx_{r'}^{r_b}$ for all r_b, r, r' where $1 \leq r_b \leq r \leq r'$
p5: $\approx_r^B \subseteq \approx_r$ for all $r \geq 1$
p6: $b_1 \approx_r^B b_2 \Rightarrow$ there exist m: $nodes(BC(b_1)) \rightarrow nodes(BC(b_2))$ such that $b_3 \approx_r m(b_3)$ for all $b_3 \in nodes(BC(b_1))$
p7: $\approx_{r_a}^B \subseteq \approx_{r_b}^B$ for all r_a, r_b where $1 \leq r_a \leq r_b$ □

Figure 6 illustrates the relationships between the types of matches for two different radii $r_a = 2$ and $r_b = 3$.

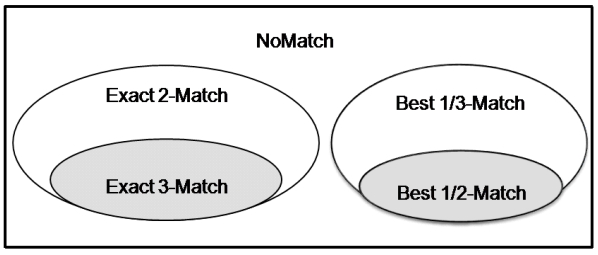


Fig. 6: Different types of Matches for radius r_a= 2 and r_b =3

3.2 The Main Idea of the Algorithm

The *Radius-Signature-based* algorithm applies the radius-based approach over the *signature-based* algorithm in order to exploit the information of $RNG(b, r, G)$, instead of staying restricted to $DNG(b, G)$. A simple way to exploit all the information could be the construction for each bnode b of a signature, say $rsign(b, r)$, that contains a specific serialization of *all* the triples in $RNG(b, r, G)$. However, there are two problems with this approach. The first one is that for dense BComponents we would end up having long signatures and therefore high space requirements. The other problem is that as r gets higher values, more triples become part of different *Radius Neighbourhood Graphs*, and thus are represented in different signatures, causing multiplication of information. To avoid such a waste of space, we consider only the absolutely necessary information that will be needed to *"reconstruct"* the signature $rsign(b, r)$. This information consists of: a) the signature $sign(b)$, and b) the directly adjacent bnodes of b, say $signBN(b)$, ordered in a way that will be made clear in the following example.

The main idea is illustrated by an example in Figure 7 where we see a bnode b_1 that has five outgoing properties; the first pointing to `Eva`, the rest four to bnodes. The new signature of b_1 is now dichotomized into two parts $\langle sign(b_1), signBN(b_1)\rangle$. The first, i.e. $sign(b_1)$, is a string, in our example 'f△Eva∗f△ •1∗f△ •2∗f△ •3∗f△ •4', while the second, i.e. $signBN(b_1)$, contains the *unnamed* information (i.e. blank nodes) connected to b_1 (here b_2, b_3, b_4, b_5). The order of the unnamed information is important to ensure a deterministic construction of the new signature. To order these bnodes we traverse the BComponent in a Breadth First Search-like manner. This is explained from the following example.

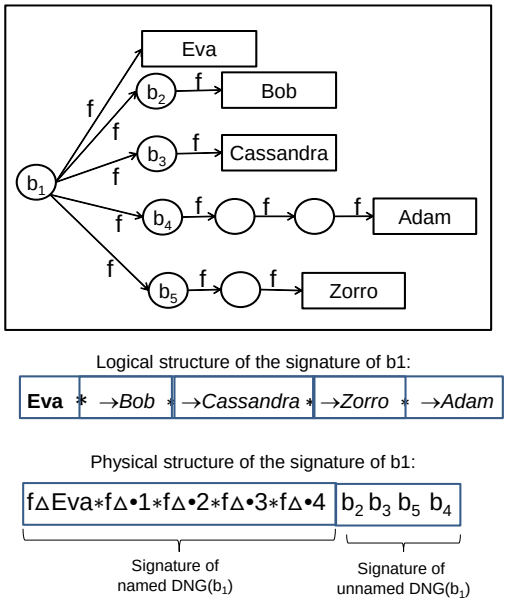


Fig. 7: Example of a signature (logical and physical structure)

Notice that initially $signBN(b_1) = \langle\{b_2, b_3, b_4, b_5\}\rangle$, since the bnodes have no discrimination ability in depth equal to 1 from b_1. Going one step deeper, we get that $sign(b_2) > sign(b_3) > \{sign(b_4), sign(b_5)\}$, since the literal 'Bob' precedes the literal 'Cassandra'. However, bnodes b_4 and b_5 cannot be differentiated yet, as $sign(b_4) = sign(b_5)$. Thereby, we go in more depth for these two nodes and discover 'f△ •' for b_4 and and 'f△Zorro' for b_5. Since 'f△Zorro' lexicographically precedes 'f△ •' we finally get $signBN(b_1) = \langle b_2, b_3, b_5, b_4\rangle$.

The bottom part of Figure 7 shows the physical form of the signature $\langle sign(b_1), signBN(b_1)\rangle$.

We observe that the signature does not get bigger in size as the radius increases. Instead, the radius affects only the ordering of the directly connected unnamed information that is represented in the signature.

3.2.1 The Main Steps of the Algorithm

The new algorithm attempts to find the match relations that were defined in Def. 9, starting from the stronger types of matches and moving to the weaker ones (i.e. from *BComponent r-Matches* to *Best* $\frac{r'}{r}$*-Matches*). The process of the *Radius-Signature-based* algorithm, for short `r-SIGN`, is given in abstract terms, having a declarative style, in Algorithm 2.

Algorithm 2: `Abstract r-SIGN`(G_1, G_2, r)

Input: graphs G_1 and G_2, radius $r \geq 2$
Output: map M_{rSIGN}

1 $M_{rSIGN} \leftarrow \emptyset$ ▷ the generated map
2 $MatchTypes \leftarrow \langle BC$ r-Match, *Exact* r-Match, *Best* $\frac{r-1}{r}$-Match, .. , Best $\frac{1}{r}$-Match $\rangle$
3 $CP \leftarrow B_1 \times B_2$ ▷ candidate pairs
4 **foreach** $mt \in MatchTypes$ **do** ▷ equivalent matches
5 Compute R_{mt} over CP
6 $f_{mt} \leftarrow$ maximum subset of R_{mt} that is a function
7 $CP \leftarrow CP \setminus f_{mt}$
8 $M_{rSIGN} \leftarrow M_{rSIGN} \cup f_{mt}$
9 $RestB1 \leftarrow B_1 \setminus \{ x \mid (x, y) \in M_{rSIGN}\}$
10 $RestB2 \leftarrow B_2 \setminus \{ y \mid (x, y) \in M_{rSIGN}\}$
11 **foreach** $b_1 \in RestB1$ **do** ▷ b_1 does not participate in any equiv. relation
12 $(b1, b2) \leftarrow (b_1, ClosestMatch(b_1, RestB2))$ ▷ closest matches
13 $M_{rSIGN} \leftarrow M_{rSIGN} \cup \{(b1, b2)\}$
14 $RestB1 \leftarrow RestB1 \setminus \{b_1\}$
15 $RestB2 \leftarrow RestB2 \setminus \{b_2\}$
16 **return** M_{rSIGN}

The algorithm gets as input two graphs G_1, G_2 and a radius r. It uses the ordered set, $MatchTypes$, that contains one constant for each type of *Matches* from the strongest to the weakest ones (line 2). In line 3 the variable CP is initialized with the Cartesian product of the blank nodes of the two graphs. For each type mt in $MatchTypes$ the algorithm computes the corresponding relation R_{mt} over CP (e.g. for $mt = BC$ r-Match, $R_{mt} = \approx_r^B$).

This is done by comparing the signatures of the bnodes, therefore the computed relation is actually an approximation of the real one.

Since it is possible for a bnode of B_1 to participate in several pairs in R_{mt}, one of them should be selected. This can be accomplished by deleting from R_{mt} those pairs that are required for turning it into a function (from B_1 to B_2), i.e. for having a $b \in B_1$ to be associated with at most one bnode of B_2. This is what line 6 does, i.e. it assigns to f_{mt} the maximum function that is a subset of R_{mt}.

The function f_{mt} becomes part of the final map M_{rSIGN}, and CP is updated accordingly. After the equivalent matches (computed in lines 1 - 8), the *closest matches* are computed over the remainder CP, and eventually they are added to M_{rSIGN} (lines 9-15).

3.2.2 The Exact Steps of the Algorithm

The exact steps of the *Radius-Signature-based* algorithm are given in Alg. 3. For each bnode $b_i \in B_1$, `r-SIGN` retrieves all the bnodes $b_j \in B'_2$ with the same signature (lines 4-5), i.e. those with isomorphic $DNG(b_i, G_1)$ and $DNG(b_j, G_2)$. Notice that B'_1 and B'_2 hold the currently not matched tuples of B_1 and B_2 respectively. The algorithm computes the strongest type of *Equivalent Match* for each pair (b_i, b_j) for the given radius r (i.e. *BC r-Match*, *Exact r-Match*, *Best* $\frac{r'}{r}$*-Match*), through the call of the recursive algorithm Alg. 4, named `findMatch` (line 9).

For each pair (b_i, b_j) the algorithm assigns to the variables $toBeVisited$ and $visited$ the current pairs of bnodes that need to be visited or the pairs that are already visited respectively for the evaluation of the particular pair (b_i, b_j). Notice that initially $visited$ contains no pairs, while $toBeVisited$ contains the pair (b_i, b_j) (lines 6-7). There is also the variable r' that denotes the depth that $RNG(b_i, r', G_1)$ and $RNG(b_j, r', G_2)$ are currently traversed and compared for isomorphism. These variables are used as the input parameters for the algorithm `findMatch` (line 9).

If `findMatch` returns that $b_i \approx_r b_j$ the algorithm simply stores the pair as candidate in the set M_{Exact} (lines 16-17). Similarly, if it returns $b_i \approx_r^{r'} b_j$ the pair is added in a list M_{Best}, where the pairs are sorted in descending order in terms of r' (lines 18-19). If $b_i \approx_r^B b_j$, then (b_i, b_j) is added to M_{rSIGN}, as well as all the pairs of bnodes that according to p6 of Prop.3 are included in m. In the case of the algorithm m is computed by the Algorithm `findMatch` and is stored in the variable $visited$ (lines 12-14). Unless a *BC r-Match* is detected, b_i is compared with any b_j with the same signature. As soon as all the bnodes of B_1 are examined (i.e. all the needed *BC r-Matches* are discovered), the algorithm adds to M_{rSIGN} firstly the pairs of M_{Exact} (lines 20-23) and secondly those of M_{Best} (lines 24-28) in a descending order in terms of r. Finally, the *Closest* matches are detected over the rest bnodes (lines 29-32).

Notice that for each bnode b_i in B_1 the algorithm stores several candidate matches (i.e. both in M_{Exact} and M_{Best}). We prefer storing several candidates, because some pairs may be consumed in a next step through an upcoming *BComponent r-Match*. However, for datasets with high number of bnodes, such storage needs may cumulatively cause high memory requirements that cannot be satisfied by the given main memory. To tackle this issue the algorithm makes a rough estimation of the main memory needs for the structures M_{Exact} and M_{Best} (through some statistics of the graphs) and proposes some guidelines for an efficient configuration of the value of radius (see more in Section 4.6).

Algorithm 3: `r-SIGN`(G_1, G_2, r)

Input: graphs G_1 and G_2, radius $r \geq 2$
Output: map M_{rSIGN}

1 $M_{rSIGN} \leftarrow \emptyset$ ▷ the returned map
2 $M_{Best}, M_{Exact} \leftarrow \emptyset$ ▷ computed Best and Exact Matches
3 $B'_1 \leftarrow B_1$, $B'_2 \leftarrow B_2$ ▷ copies of B_1 and B_2
4 **foreach** $b_i \in B'_1$ **do**
5 loop: **foreach** $b_j \in B'_2$ *with* $sign(b_i) = sign(b_j)$ **do**
6 $toBeVisited \leftarrow \{(b_i, b_j)\}$
7 $visited \leftarrow \emptyset$
8 $r' \leftarrow 2$
9 $type \leftarrow$ `findMatch`$(tobeVisited, visited, r', r)$
10 **switch** *type* **do** ▷ Equivalence Match for (b_i, b_j)
11 **case** $\approx_r^B$ ▷ BComponent r-Match
12 $M_{rSIGN} \leftarrow M_{rSIGN} \cup visited$
13 $B'_1 \leftarrow B'_1 \setminus \{b_1 \mid \forall\ (b_1, x) \in visited\}$
14 $B'_2 \leftarrow B'_2 \setminus \{b_2 \mid \forall\ (y, b_2) \in visited\}$
15 **break** loop
16 **case** $\approx_r$ ▷ Exact r-Match
17 $M_{Exact} \leftarrow M_{Exact} \cup \{(b_i, b_j)\}$
18 **case** $\approx_r^{r'}$ ▷ Best r'/r-Match
19 $M_{Best} \leftarrow M_{Best} \cup \{(b_i, b_j, r')\}$
20 **foreach** $\{(b_i, b_j) \in M_{Exact} \mid b_i \in B'_1 \wedge b_j \in B'_2\}$ **do**
21 $M_{rSIGN} \leftarrow M_{rSIGN} \cup \{(b_i, b_j)\}$
22 $B'_1 \leftarrow B'_1 \setminus \{b_i\}$
23 $B'_2 \leftarrow B'_2 \setminus \{b_j\}$
24 **for** $r' \leftarrow r - 1$ *to* 1 **do**
25 **foreach** $\{(b_i, b_j) \mid (b_i, b_j, r') \in M_{Best} \wedge b_i \in B'_1 \wedge b_j \in B'_2\}$ **do**
26 $M_{rSIGN} \leftarrow M_{rSIGN} \cup \{(b_i, b_j)\}$
27 $B'_1 \leftarrow B'_1 \setminus \{b_i\}$
28 $B'_2 \leftarrow B'_2 \setminus \{b_j\}$
29 **foreach** $b_i \in B'_1$ **do** ▷ Closest Match
30 $b_j \leftarrow closestMatch(b_i, B'_2)$
31 $B'_2 \leftarrow B'_2 \setminus \{b_j\}$
32 $M_{rSIGN} \leftarrow M_{rSIGN} \cup \{(b_i, b_j)\}$
33 **return** M_{rSIGN}

Let us now see how `r-SIGN` recursively compares the sub-graphs $RNG(b_i, r, G_1)$ and $RNG(b_j, r, G_2)$ for each pair (b_i, b_j). The algorithm `findMatch` (Alg. 4) takes as input four parameters, i.e. two sets with pairs of bnodes, $toBeVisited$ and $visited$ (as explained earlier) and two radii, where the first one (r') indicates the depth that the RNG subgraphs are examined at the current recursion and the second one (r) sets the upper bound for the recursions of `findMatch`. For each pair (b_i, b_j) the first call of `findMatch` has the following values $toBeVisisted = \{(b_i, b_j)\}$ $visited = \emptyset$, $r' = 2$ and actually computes if there is an isomorphism in $RNG(b_i, 2, G_1)$ and $RNG(b_j, 2, G_2)$. The next recursive call computes if there is an isomorphism in $RNG(b_i, 3, G_1)$ and $RNG(b_j, 3, G_2)$ and so on. In particular, `findMatch` retrieves the pairs (a_i, a_j), where $a_i = signBN(b_i)[k]$ and $a_j = signBN(b_j)[k]$ for each $k \in [0, |signBN(b_i)|)$. For each pair of bnodes (line 6) it checks if their signatures are equal (line 7). It is enough to compare the signatures

Algorithm 4: `findMatch`($toBeVisited$,$visited$,r',r)

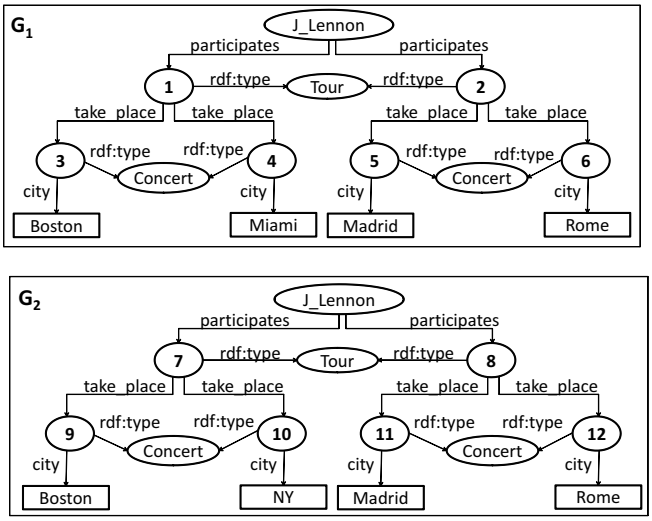
Input: two graphs G_1, G_2, two queues of pairs of bnodes *toBevisited*, *visited*, two radii r', r, where $r \geq r' \geq 2$
Output: type of Equivalent Match

1 $equivDNGs \leftarrow true$ ▷ isomorphic DNGs in tobeVisited
2 $tmp \leftarrow \emptyset$ ▷ pairs of adjacent bnodes
3 **while** $(toBeVisited \neq \emptyset)$ **do**
4 $(b_i, b_j) \leftarrow toBeVisited.dequeue()$
5 **for** $k \leftarrow 0$ *to* $|signBN(b_i)| - 1$ **do** ▷ pairs of adjacent bnodes
6 $(a_i, a_j) \leftarrow (signBN(b_i)[k], signBN(b_j)[k])$
7 **if** $(sign(a_i) = sign(a_j))$ **then**
8 **if** $((a_i, a_j) \notin visited \cup toBeVisited \cup tmp)$ **then**
9 $tmp.enqueue((a_i, a_j))$
10 **else**
11 $equivDNGs \leftarrow false$ ▷ $DNG(a_i, G_1) \not\equiv DNG(a_j, G_2)$
12 $visited \leftarrow visited \cup \{(b_i, b_j)\}$
13 $toBeVisited \leftarrow tmp$
14 **if** $(equivDNGs)$ **then**
15 **if** $(r' = r \wedge |toBeVisited| > 0)$ **then**
16 **return** $\approx_r$
17 **else**
18 **if** $(|toBeVisited| > 0)$ **then**
19 **return** `findMatch`$(toBeVisited,\ visited,\ ++r', r)$
20 **else**
21 **return** $\approx_r^B$
22 **else**
23 **return** $\approx_r^{r'-1}$

of the bnodes at k position, since both $signBN(b_i)$ and $signBN(b_j)$ are ordered in a deterministic way. If a pair of bnodes does not have the same signatures (lines 10-11), the value $equivDNGs$ is turned to $false$ (lines 10-11) and the algorithm only returns a *Best* $\frac{r'-1}{r}$*-Match* (lines 22-23). Otherwise, the algorithm stores the pairs (b_i, b_j) to $visited$ (i.e. marks them as visited) and the pairs (a_i, a_j) to $toBeVisited$ (will be compared in the next call). If there are pairs in $toBeVisited$ indeed, the algorithm calls itself to compare $RNG(b_i, 3, G_1)$ and $RNG(b_j, 3, G_2)$ (lines 18-19). This procedure continues until a non-matching signature is compared (i.e. *Best* $\frac{r'-1}{r}$*-Match*), or $toBeVisited$ is empty (i.e. *BC r-Match*) or $r' = r$ with $toBeVisited$ being non empty (i.e. *Exact r-Match*).

Example Trace

Fig. 8 gives a quick overview of how `r-SIGN` works over the running example (i.e. it shows the main method calls and their outputs).



(a) Graphs of the running example

graph G_1		
b	**sign(b)**	**signBN(b)**
3	rdf:type△Concert◊take_place△•1◊city△Boston	[1]
5	rdf:type△Concert◊take_place△•1◊city△Madrid	[2]
4	rdf:type△Concert◊take_place△•1◊city△Miami	[1]
6	rdf:type△Concert◊take_place△•1◊city△Rome	[2]
1	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[3, 4]
2	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[5, 6]

graph G_2		
b	**sign(b)**	**signBN(b)**
9	rdf:type△Concert◊take_place△•1◊city△Boston	[7]
11	rdf:type△Concert◊take_place△•1◊city△Madrid	[8]
10	rdf:type△Concert◊take_place△•1◊city△NY	[7]
12	rdf:type△Concert◊take_place△•1◊city△Rome	[8]
7	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[9, 10]
8	rdf:type△Tour◊participates△J_Lennon◊take_place△•1∗take_place△•2	[11, 12]

(b) Signatures over the running example

Algorithm r-SIGN(G_1, G_2, 2)					
Lines	**Algorithm calls**	**Equivalent Matches**			M_{rSIGN}
		$\approx^{B}_{2}$	$\approx_{2}$	$\approx^{1}_{2}$	
4 – 19	findMatch({**(1,7)**}, ∅, 2, 2)	∅	∅	**(1,7)**	∅
4 – 19	findMatch({**(1,8)**}, ∅, 2, 2)	∅	∅	(1,7) **(1,8)**	∅
4 – 19	findMatch({**(3,9)**}, ∅, 2, 2)	∅	**(3,9)**	(1,7) (1,8)	∅
4 – 19	findMatch({**(2,8)**}, ∅, 2, 2)	**(2, 8)**	(3,9)	(1,7) (1,8)	{**(2, 8) (5, 11) (6, 12)**}
20 – 23			**(3,9)**	(1,7) (1,8)	{(2, 8) (5, 11) (6, 12) **(3, 9)**}
24 – 28			(3,9)	**(1,7)** (1,8)	{(2, 8) (5, 11) (6, 12) (3, 9) **(1,7)**}
29 - 32			(3,9)	(1,7) (1,8)	{(2, 8) (5, 11) (6, 12) (3, 9) (1,7) **(4, 10)**}

(c) Plain trace of algorithm r-SIGN for radius r = 2

Fig. 8: Application of r-SIGN over the running example for radius r = 2

3.3 Properties

Here we discuss some properties of the algorithm concerning the deviation from optimum (since it is an approximation algorithm), its relationship with the previous algorithm, and the impact of radius increase.

Prop. 4 *DNG and Signatures*
For two bnodes $b_1 \in B_1$ and $b_2 \in B_2$ it holds that $DNG(b_1, G_1) \equiv DNG(b_2, G_2)$ $\Leftrightarrow sign(b_1) = sign(b_2)$. □

Now let $\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_x)$ denote the *Delta of the Unnamed* parts of two graphs G_1 and G_2, using a map M_x, without considering the change operations about the named parts of the graphs. In other words $\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_x)$ contains those operations of $\Delta_e(G_1 \rightarrow G_2, M_x)$ that add or delete triples that contain bnodes. The following upper bound holds for any bnode matching algorithm.

Prop. 5 *Upper Bound of "Unnamed" Delta*
For any map M_x between two graphs G_1 and G_2 it holds that $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_x)|$ $\leq |btriples(G_1)| + |btriples(G_2)| + |bbtriples(G_1)| + |bbtriples(G_2)|$. □

Below we prove a less upper bound (than that of Prop. 5) for `SIGN` and `r-SIGN` for the case of equivalent graphs (in comparison to Prop. 5 these bounds have two addends less).

Prop. 6 *Deviation from Optimum for Equivalent Graphs of* `SIGN` *and* `r-SIGN`
If $G_1 \equiv G_2$ then
(i) $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_{SIGN})| \leq |bbtriples(G_1)| + |bbtriples(G_2)|$.
(ii) $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_{rSIGN})| \leq |bbtriples(G_1)| + |bbtriples(G_2)|$.
This upper bound is actually an upper bound of the deviation from the optimum, since the optimal $\Delta_e^{\mathcal{B}}$ is empty. □

The next proposition proves that `r-SIGN` is as good as `SIGN` if there are not connected bnodes.

Prop. 7 `SIGN` *vs* `r-SIGN` *if no connected bnodes*
If G_1 and G_2 have no connected bnodes, i.e. none of them contains any *bbtriple*, then $M_{SIGN} = M_{rSIGN}$ for any $r \geq 2$. □

The next proposition shows that the "greedy nature" of `r-SIGN` guarantees the finding of the maximum BComponent match if the radius is big enough.

Prop. 8 `SIGN` *vs* `r-SIGN`
If $G_1 \equiv G_2$ then `r-SIGN` will find the maximum in diameter (say $dmax$) BC $dmax$-Match if it is unique, and $r \geq dmax$. This is not always true for `SIGN`. □

3.4 Complexity

The time complexity of the previous algorithm `SIGN` is $O(n \cdot \log n)$, where $n = \max(|B_1|, |B_2|)$. In a more refined analysis we should also consider the parameter d that stands for the average degree of each bnode in the graph (for simplicity common for both graphs). During the *Construction* Phase of `SIGN`, fetching the

triples of all the *Direct Neighborhoods* of the bnodes to construct their signatures costs $O(d \cdot n)$, while sorting lexicographically the triple representations inside the signatures costs $O(d \cdot \log(d) \cdot n)$. Notice that in our analysis we neglect the length of each triple representation. For the *Matching* Phase the complexity is $O(n \cdot \log(n))$, as we perform a lookup for each tuple in T_1 to the tuples of T_2. Table 1 shows the complexity of both algorithms for the *Signature Construction* and *Matching* Phase separately.

Table 1: Complexities of `SIGN` and `r-SIGN`

Algorithm	*Signature Construction*	*Matching*
`SIGN`	$O(d \cdot \log(d) \cdot n)$	$O(n \cdot \log(n))$
`r-SIGN`	$O(d_b^{r-1} \cdot d \cdot \log(d) \cdot n)$	$O((d_b^{r-1} + \log(n)) \cdot n^2)$

For the proposed `r-SIGN` we need to consider two additional parameters: the radius r, and the value d_b standing for the average number of *bbtriples* that each bnode participates. Without loss of generality, let us assume that the values of these parameters are common for both G_1 and G_2. Regarding the *Signature Construction* Phase, in the worst case scenario for the construction of each $sign(b)$ the algorithm has to construct d_b^{r-1} signatures in order to decide on the ordering of bnodes in $signBN(b)$. So, the time complexity gets $O(d_b^{r-1} \cdot d \cdot \log(d) \cdot n)$. In the *Matching* Phase, at most d_b^{r-1} signature comparisons should be made to decide if a pair of bnodes is exactly matched. In the worst case scenario (which does not practically occur) all the bnodes have the same signature and are all part of the same BComponent. In this case each bnode of G_1 has to be compared with n bnodes of G_2, so the complexity of the algorithm is $O((d_b^{r-1} + \log(n)) \cdot n^2)$.

For a chain graph, r could be equal to $n-1$, yielding a clear exponential with respect to n complexity (recall that according to [28] the problem of finding the optimal map is NP-Hard).

Because of the exponential time complexity over radius, for large and extremely dense BComponents (i.e. high d_b) a high value of r could be prohibitive. However, in the experimental evaluation we shall see that it is possible to run the algorithm for radius equal to 1500 on real and million-sized datasets in 15 seconds.

3.5 Good and Bad Cases for the Algorithm

Here we list some good and bad cases, i.e. cases where `r-SIGN` is expected to work very well (in terms of optimality), and cases where it is expected to work less well (in terms of optimality or time/space complexity).

- *Good Cases.*
  In general it is a good case if the compared graphs have small percentage of bnodes that get the same signatures. In this case the bnode matching problem is easier in general. The algorithm is good in such cases, in the sense that (a) it can uniquely identify bnodes even with small radii and (b) it will not save a lot of intermediate results (M_{Exact}, M_{Best} of Algorithm 3).
- *Bad Cases.*
  If the compared graphs have high percentage of bnodes that get the same

signature, then the problem is more difficult and the algorithm will require increased space for the intermediate results. For instance, in non-isomorphic graphs, there is at least one BComponent that does not match. The algorithm while looking for a match for one bnode of that BComponent could keep in memory a lot of candidate exact and best matches which can be high especially when there is a high percentage of bnodes that have the same signature.
However, if the changes are not so scattered in most of BComponents, the high requirements can be avoided, or alleviated, through an appropriate configuration of the value of radius (more in Section 4.6).
Finally, if the *classes* of the bnodes in the compared graphs have changed, then this may affect negatively the optimality because $Class(b)$ is placed first of the signatures.

Let us discuss the deviation from the optimum. Given two graphs G_1 and G_2, the optimal delta if they are isomorphic is $|\Delta_{opt}| = \min_M(|\Delta_e(G \rightarrow G', M)|) = 0$, while in the worst case the biggest possible delta is $|\Delta_{worst}| = \max_M(|\Delta_e(G \rightarrow G', M)|) \leq |BTriples(G)|+|BTriples(G')|$, where $BTriples(G)$ is the set of triples of G that contain one or two bnodes (recall Prop. 5).

A normalized value of the deviation from the optimum can therefore be defined as: $\frac{|\Delta_{sign}|-|\Delta_{opt}|}{|\Delta_{worst}|}$. Note that for isomorphic datasets Δ_{opt}=0, thus the value is finally reduced to $\frac{|\Delta_{sign}|}{|\Delta_{worst}|} = \frac{|\Delta_{sign}|}{|BTriples(G)|+|BTriples(G')|}$. In general, for isomorphic KBs the greater the radius gets, the less (or same) the deviation from the optimum gets. For non-isomorphic KBs the greater the radius gets, the more probable to get a lower deviation from the optimum becomes. Below we will report measurements about the deviation.

4 Experimental Evaluation

We comparatively evaluate `r-SIGN` with previous bnode matching algorithms (`SIGN` and `HUNG` [28]), to experimentally test the merits of `r-SIGN` over them, and check how the radius r affects the results. In brief, below we report the results of experiments conducted over real and synthetically generated datasets of variable complexity. For isomorphic datasets the proposed algorithm managed to find the optimal solution for radius close to the average depth of the contained BComponents. In non-isomorphic datasets `r-SIGN` managed to reduce the delta size from 2 to 28 times in comparison to the older algorithms with reasonable time requirements. Regarding scalability, it is worth stressing that this is the first time that bnode matching is applied to millions (almost 1.7 millions) of blank nodes.

Comparative results with CWM [2] would not make sense because that approach requires assigning functional properties (so it is not a general solution that can be applied straightforwardly). The algorithms in Jena (n^2 complexity, no experimental results) and Hogan [12] can only detect equivalent graphs; cannot reduce the delta in the general case (in both equivalent and nonequivalent graphs). Ontoview [18] does not state explicitly how blank nodes are treated, while PromptDiff [23] does not provide details about the used algorithm for enabling comparative evaluation. Finally approaches that return error-tolerant subgraph matching (like [5]) are beyond the scope of this work because they are appropriate for searching, not for equivalence testing, versioning or synchronization.

Below we describe in detail (a) the experimental tasks, (b) the features of the graphs that are critical for the algorithms, (c) the statistics of the used datasets, (d) the results of the evaluation, (e) the tradeoff between time complexity and deviation from optimum, and (f) guidelines for selecting which radius to use.

4.1 Experimental Tasks

Two main tasks were conducted in our experiments.
Isomorphism task. This task requires two RDF graphs, G_1 and G_2, that are isomorphic according to Def. 1. Both graphs represent the same version of a dataset, but the serialization of their triples and bnodes might change from one graph to the other. In the context of these experiments the *Isomorphism task* requires that the serialization of triples in G_2 is the *reverse* of the one in G_1. This is because such a serialization is the worst case scenario for the isomorphism checking problem and ensures that the algorithms cannot abuse text-based features. Bear in mind that for this task the optimal solution always equals to 0, i.e. $\Delta_e(G_1 \rightarrow G_2, M) = 0$.
Differential task. This task requires two RDF graphs, G_1 and G_2, that are different, i.e. two versions of a dataset. Similarly to the previous task, the serialization of triples in G_2 is the *reverse* of the one in G_1. The optimal solution is not a-priori known and it can only be computed through a controlled construction of the new version or through an exhaustive bnode matching algorithm.

4.2 Related Graph Features

Below we identify the main features of graphs and BComponents that affect their connectivity and could impact the efficiency and optimality of bnode matching algorithms. The first three features are related to the graph G, while the rest eight are related to the BComponents of G.

Notation	Semantics
`|triples|`	number of triples (quads)
`|B|`	number of bnodes
`|BC|`	number of BComponents
$\texttt{sz}_{\texttt{avg}}$	average number of bnodes in BComponents
$\texttt{sz}_{\texttt{max}}$	maximum number of bnodes in BComponents
$\texttt{bb}_{\texttt{avg}}$	average number of bbtriples to which a bnode participates in a BComponent
$\texttt{ext}_{\texttt{avg}}$	average extroversion of BComponents where extroversion is the average number of btriples that a bnode participates
$\texttt{tw}_{\texttt{avg}}$	average treewidth of BComponents
$\texttt{tw}_{\texttt{max}}$	maximum treewidth of BComponents
$\texttt{dp}_{\texttt{avg}}$	average depth of BComponents
$\texttt{dp}_{\texttt{max}}$	maximum depth of BComponents

The feature *treewidth* is useful for characterizing graphs that contain *cycles* (if its value is greater than 1 then this means that the graph contains an undirected cycle). The *treewidth* is computed using an implementation[4] of the *GreedyFillIn* upper bound algorithm [16] and the lower bound *Ramachandramurthi* algorithm

[4] `http://www.treewidth.com/treewidth/`

[25]. The *depth* is given as the depth of a spanning tree of the *BComponent* exported when breadth-first-search is applied over it. The *diameter* [10] would be more representative, but its high computational cost is prohibitive in the general case. The computation of the *depth* depends on the starting node of the BComponent (taken arbitrarily); however, it is an efficient approach even for large BComponents and is a good reference point as it is bounded in [*diameter*/2, *diameter*].

4.3 The Used Datasets

The experiments were conducted over real and synthetic datasets[5], coming from three different sources. Below we give the datasets of each source separately.

4.3.1 Real Datasets

In order to be as realistic and representative as possible, we have used the Billion Triple Challenge (BTC) 2012 and 2014 Datasets ([11,14]). For the needs of our experiments we focus on some particular datasets, all of them being part of the LOD cloud and having high percentages of bnodes. Specifically we selected the datasets having the highest percentage of bnodes. Table 2 gives the names of the datasets and their features.

For the domains `bbc.co.uk`, `fao.org` and `openlylocal.com` we give their datasets for both 2012 and 2014, while for `geovocab.org` we only give the dataset for 2012. The datasets with the simplest structures of bnodes are the ones of the `openlylocal.com` domain.

Hereafter, we shall call a bnode that does not participate in any *bbtriple* as *singleton* BComponent. Regarding the `openlylocal.com`, the 2012 dataset contains exclusively singleton BComponents (i.e. bnodes are not directly connected between them), as $\texttt{sz}_{max} = 1$. For this reason, applying `r-SIGN` for the *Isomorphism Task* over this dataset gives the optimal solution for any radius. The 2014 dataset contains both singleton and *tree* BComponents (i.e. bnodes are connected in a structure that forms a tree), as $\texttt{sz}_{avg}$=2.1, $\texttt{dp}_{avg}$=1 and $\texttt{tw}_{avg} \approx 0$. Similarly, both datasets of `bbc.co.uk` contain *singleton* and small *tree* BComponents. The datasets of `fao.org` are more complex, as the 2012 dataset has $\texttt{sz}_{max}$ = 12, $\texttt{tw}_{avg}$ = 0.3 and $\texttt{dp}_{max}$=11. The 2014 dataset is the largest one containing 1,679,414 bnodes and 7,973,958 triples, where more than 2,000,000 are *bbtriples* and 4,000,000 are *btriples*. The contained BComponents are more complex, than those of 2012, as the average size of a BComponent is doubled and the average treewidth is close to 2 ($\texttt{tw}_{avg} = 1.95$) meaning that they contain cycles (or DAGs). Finally, the 2012 dataset of `geovocab.org` contains mostly *singleton* BComponents ($\texttt{tw}_{avg} \approx 0$) and very few *tree* BComponents ($\texttt{tw}_{max} = 1$). An eminent feature of this dataset is $\texttt{dp}_{\texttt{max}} = 1,719$ meaning that there is one (or even some more) *tree* BComponents that have depth equal to 1,719. Comparing the two different versions (2012 and 2014) of each dataset, it is easy to notice that important changes have occurred from 2012 to 2014. For instance, in the `fao.org` dataset, the triples are increased by two orders of magnitude, while the average size of the BComponents is doubled. These variations indicate that the two versions may not share a

[5] See `http://www.ics.forth.gr/isl/bnodeland/radius`

common basis anymore. The experimental results shed more light to this indication.

Table 2: Features of Real Datasets from BTC 2012 and 2014

Dataset	|triples|	|B|	|BC|	$\mathbf{sz_{avg}}$	$\mathbf{sz_{max}}$	$\mathbf{bb_{avg}}$	$\mathbf{ext_{avg}}$	$\mathbf{tw_{avg}}$	$\mathbf{tw_{max}}$	$\mathbf{dp_{avg}}$	$\mathbf{dp_{max}}$
openlylocal12	104,185	8,100	8,100	1	1	0	4.5	0	0	0	0
openlylocal14	1,742,832	79,306	37,996	2.1	4	1	1	~ 0	1	1	2
bbc12	1,465,541	126,138	91,910	1.4	4	0.3	5.6	~ 0	1	0.4	3
bbc14	3,059,237	360,101	221,955	1.6	3	0.8	4.8	~ 0	1	0.6	2
fao12	56,646	4,602	1,141	4	12	2.14	5.0	0.297	2	1.7	11
fao14	7,973,958	1,679,414	211,668	7.9	12	2.5	2.4	1.957	2	4	11
geovocab12	912,404	228,160	89,273	2.6	4,570	1.2	5.6	~ 0	1	0.7	1,719

Table 3: Features of Synthetic Datasets from BGen

Dataset	|triples|	$\mathbf{bb_{avg}}$	$\mathbf{tw_{avg}}$	$\mathbf{tw_{max}}$	$\mathbf{dp_{avg}}$	$\mathbf{dp_{max}}$
$\mathtt{BGen}_1$	77,990	2.3	3	3	10.8	12
$\mathtt{BGen}_2$	81,990	2.6	3	3	11.1	12
$\mathtt{BGen}_3$	83,238	2.8	3.8	5	10.8	11
$\mathtt{BGen}_4$	86,882	3.2	4.4	5	10.4	11
$\mathtt{BGen}_5$	92,684	3.8	6.2	8	11.5	12
$\mathtt{BGen}_6$	104,210	4.9	8.7	12	11.1	12
$\mathtt{BGen}_7$	108,530	5.4	10	12	11.1	12
$\mathtt{BGen}_8$	117,530	6.3	12	12	4.9	6

Table 4: Features of Synthetic Datasets from Universities

Dataset	|triples|	|B|	|BC|	$\mathbf{sz_{avg}}$	$\mathbf{sz_{max}}$	$\mathbf{bb_{avg}}$	$\mathbf{ext_{avg}}$	$\mathbf{tw_{avg}}$	$\mathbf{tw_{max}}$	$\mathbf{dp_{avg}}$	$\mathbf{dp_{max}}$
$\mathtt{Univ}_{5a}$	1,399	240	120	2	8	0.55	3.50	~ 0	1	1.3	4
$\mathtt{Univ}_{5b}$	1,399	240	120	2	8	0.55	3.50	~ 0	1	1.32	4
$\mathtt{Univ}_{6a}$	919	240	120	2	8	0.55	2.05	~ 0	1	1.18	4
$\mathtt{Univ}_{6b}$	919	240	120	2	8	0.57	2.06	~ 0	1	1.34	4
$\mathtt{Univ}_{7a}$	909	240	90	2.66	10	0.8	2.0	~ 0	1	1.66	4
$\mathtt{Univ}_{7b}$	908	240	91	2.63	10	0.79	2.0	~ 0	1	1.72	4
$\mathtt{Univ}_{8a}$	1,002	240	18	13.3	142	1.42	2.04	1.1	3	8.89	68
$\mathtt{Univ}_{8b}$	1,001	240	19	12.63	109	1.4	2.08	~ 0	1	8.57	45

4.3.2 Synthetic Datasets

To evaluate the algorithms in a wider range of complex BComponents (i.e. various values of treewidth, depth, extroversion), we also conducted experiments over synthetic datasets from two different generators.

The first generator is *BGen* [21], a Semantic Web data generator that offers control over the connectivity of bnodes through a variety of input parameters. We generated eight datasets, $\{\mathtt{BG}_1, ..., \mathtt{BG}_8\}$. All of them have the following common features: $\mathtt{|B|} = 20,000$, $\mathtt{|BC|} = 10$, $\mathtt{sz}_{avg} = \mathtt{sz}_{max} = 2,000$ and $\mathtt{ext}_{avg} = 2.75$. The rest of the features, shown at Table 3, change gradually from the one dataset

to the other so as to increase the complexity of their BComponents. The datasets are used for the *Isomorphism task*, since controlled generation of different versions is not supported by *BGen*.

Therefore for the *Differential task*, we use a more static generator (used in [28]) with known optimal maps (by construction), customized over the UBA generator [9] to generate datasets with structures of bnodes. Table 4 gives the features of two versions of 4 datasets i.e. a pair ($\texttt{Univ}_{\texttt{ia}}$, $\texttt{Univ}_{\texttt{ib}}$) for each dataset $\texttt{Univ}_{\texttt{i}}$. The main changes from the one version to the other are connectivity modifications.

4.4 Experimental Results

The experiments were conducted on a machine with Intel Xeon CPU E5-2630 at 2.30GHz using one core and 64GB RAM, running Debian Linux wheeze version and Sesame 2.7.13 SailRepository over Main Memory Store[6].

BTC 2012, 2014: Regarding the real datasets, experiments were conducted for both *Isomorphism* and *Differential tasks*. Note that the algorithm `HUNG` (from [28]) could not be applied on datasets with big number of bnodes due to its quadratic main memory requirements.

Let us start from the *Isomorphism Task*. Figures 9 and 10 report the delta size ($|\Delta_e(G_i \rightarrow G_i, M)|$) and the matching time for the *Isomorphism task*.

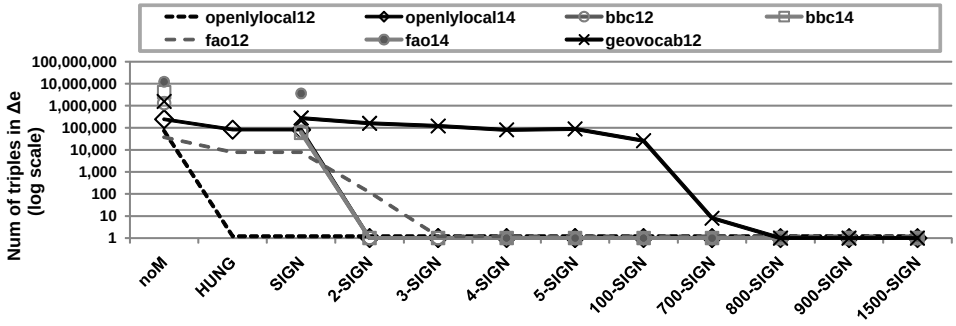


Fig. 9: Delta size (BTC - Isomorphism task)

Regarding the effectiveness of `r-SIGN`, Figure 9 shows the delta size for 7 datasets (their plot type is shown in the upper area). The x-axis corrsponds to the matching options, specifically it contains values corresponding to noMatching, `HUNG`, `SIGN`, and `r-SIGN` for various values of r (from 2 up to 1500). Notice that the `SIGN` computes a map $M = M_{SIGN}$ that gives on average 84.5% smaller delta than that of `noMatching` (i.e. $M = \emptyset$). `2-SIGN` computes a map that gives on average 68.5% and 93% smaller delta than `SIGN` and `noMatching` respectively. A crucial observation is that `r-SIGN` achieves the smallest delta for all the datasets for radii $r \geq \texttt{dp}_{\texttt{max}}$. Recall (from Prop. 2) that for each dataset there is a radius r_{max} such that for any $r \geq r_{max}$ the *Maximum Neighborhood Graph* is discovered. Practically,

[6] `http://rdf4j.org/`

we observe that there is a pointer, say, $r_{opt} \leq r_{max}$, for which `r-SIGN` achieves the best solution. Notice that in all experiments $r_{opt} \in [\mathtt{dp}_{\mathtt{avg}}, \mathtt{dp}_{\mathtt{max}}]$. In particular, notice that for all the datasets with singleton and small *tree* BComponents we get that $r_{opt} = \lceil \mathtt{dp}_{\mathtt{avg}} \rceil + 1$ and for `fao14` that contains *cycle* BComponents we get that $r_{opt} = \mathtt{dp}_{\mathtt{avg}}$. However, for datasets with high diversity in the depth of their BComponents, such as `geovocal12` (i.e. $\mathtt{dp}_{\mathtt{avg}} = 0.7$ and $\mathtt{dp}_{\mathtt{max}} = 1,719$), r_{opt} is influenced more by the value of $\mathtt{dp}_{\mathtt{max}}$ (i.e. $r_{opt} = 600$). Intuitively, as r gets values closer to $\mathtt{dp}_{\mathtt{avg}}$ more *BComponent Matches* are detected by the algorithm. As soon as r becomes equal to $\mathtt{dp}_{\mathtt{max}}$ `r-SIGN` is able to detect all the *BComponent Matches*. Configuring the value of radius based on the values of $\mathtt{dp}_{\mathtt{max}}$ and $\mathtt{dp}_{\mathtt{avg}}$ is compelling. A function that heuristically approaches r_{opt} is $r_{approxopt} = \mathtt{dp}_{\mathtt{avg}} + 1$ (we further discuss the issue of radius selection in §4.6).

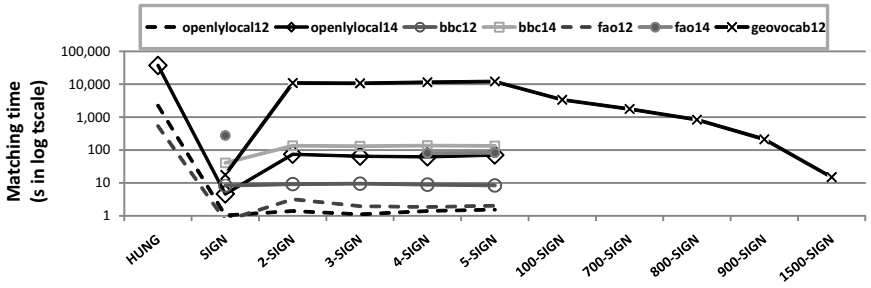


Fig. 10: Matching time (BTC - Isomorphism task)

Regarding the efficiency of `r-SIGN` Figure 10 shows the matching times for the datasets and matching options that were used in Figure 9. Notice that the new algorithm is on average 570 times more efficient than `HUNG` and 35 times less efficient than `SIGN`. Another important observation is that the detection of *BC Matches decreases* the matching time, since less signature comparisons are made. Indicatively, for `geovocab12` the matching time for radius $r = 100$ is 68% smaller than that of $r = 3$.

Comparing the curves among them, we get that the number of bnodes of a dataset plays a determining role in the matching time, as the time complexity of the algorithm reveals, too. Indicatively, `openlylocal12`, the dataset with the least bnodes (i.e. $|B| = 8{,}100$), gives the lowest matching times, while `fao14`, the dataset with the most bnodes (i.e. $|B| = 1{,}679{,}414$), gives the highest matching times. The rest of the features that impact on time complexity will be observed in the results of the synthetic datasets, where the number of bnodes is stable from the one dataset to the other.

Also note that `fao14` was not applied for `r-SIGN` with radius $r = 2$ and $r = 3$, because of its high space requirements due to the large number of intermediate results. However, for radii $r \geq 4$ `r-SIGN` was efficient and the optimal solution was achieved (with $r \geq 4$ BComponent matches are spotted early enough and this reduces the number of intermediate results).

Finally, experiments were not conducted for most datasets for radius greater than 5, and therefore no matching times are reported for them. The reason for this

is that the optimal solution is achieved for a smaller radius and the matching time cannot become less for $r > 5$, since their maximum depth (dp_{max}) is less than 5.

Differential Task: For the *Differential task* the conducted experiments reveal that BTC 2012 and BTC 2014 differ very much, therefore the differential task is not beneficial in general. For instance, the best achieved delta size in `bbc.co.uk` and `fao.org` is the 97% and 99% of the total size of the two versions respectively. The datasets that follow better suit for evaluating the differential task.

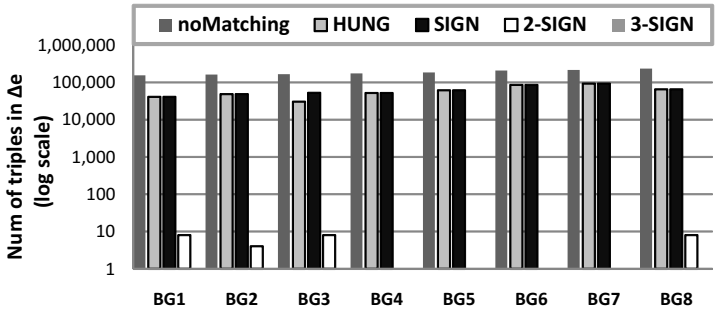


Fig. 11: Delta size (BGen - Isomorphism task). The number of triples for 2-SIGN and 3-SIGN are zero in some datasets and therefore the corresponding bars are invisible.

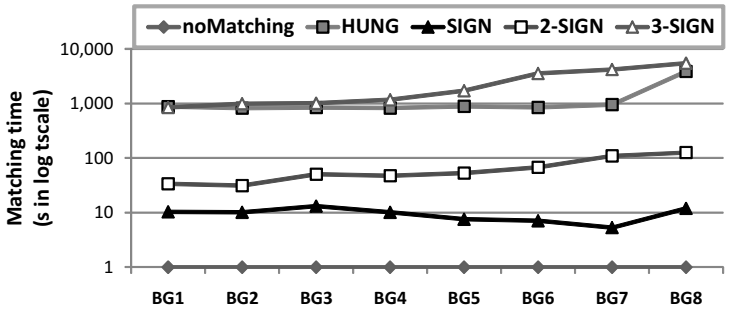


Fig. 12: Matching time (BGen - Isomorphism task)

BGen: Figures 11 and 12 report the delta size and the matching time of the *Isomorphism task* over the *BGen* datasets. A very impressive observation is that `2-SIGN` managed to find the optimal solution for the 50% of the datasets and `3-SIGN` for 100%. `2-SIGN` offered on average a significantly smaller delta (99.9%) than `SIGN` and `HUNG`, i.e. thousands of triples smaller. The matching time is 2-20 times higher than that of `SIGN`, and 18-70 times lower than that of `HUNG`. The algorithm remains time efficient, as it needs less than one minute for all the datasets except $\texttt{BG}_7$ and $\texttt{BG}_8$ that require almost 2 minutes. Note that `HUNG` needs on average 32 minutes and returns a delta that is on average 369,400 times bigger than that of `r-SIGN` (for radius $r = 2$ and $r = 3$ on average).

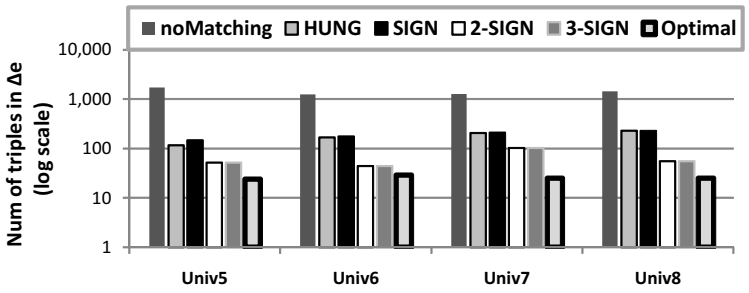


Fig. 13: Delta size (Universities - Differential task)

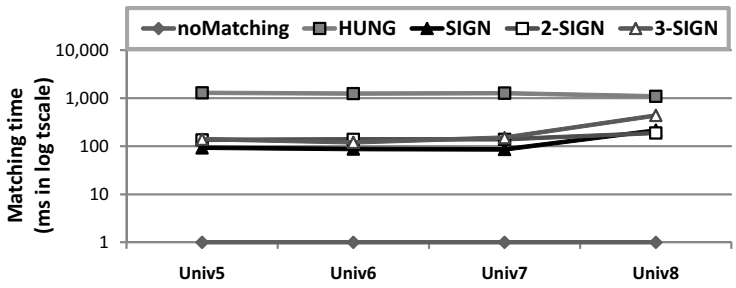


Fig. 14: Matching time (Universities - Differential task)

Universities: Let us now focus on the the *Differential task* over the synthetic datasets coming from [28]. Figures 13 and 14 give the delta size and the matching time. `2-SIGN` managed (a) to give deltas 50% - 75% smaller than those of `SIGN` and `HUNG` and (b) to deviate 34% - 75% from the optimal delta. `2-SIGN` reports 45% - 78% higher matching times than `SIGN`. Notice that `3-SIGN` does not further reduce delta.

4.5 Understanding the Tradeoff between Deviation from Optimality and Time

To understand the trade off between the deviation from the optimum and time efficiency below we present plots that summarize the aforementioned measurements.

In the plots we use the following notations: For a dataset d and a natural number r, we denote with dr the application of the algorithm `r-SIGN` over two versions (i.e. G and G') of d. Each dr is represented in the plots as a point with coordinates ($dr.x$,$dr.y$) where $dr.x$ corresponds to execution time, specifically it is the matching time of dr as percentage of the maximum matching time of any dr', while $dr.y$ represents the deviation of delta size from the delta size obtained with the optimal mapping. We will use Δ_{sign} to denote $\Delta_e(G \rightarrow G', M)$ when `r-SIGN` is applied and we will use $|\Delta_{opt}|$, $|\Delta_{worst}|$, and the normalized deviation from the optimum as defined in §3.5.

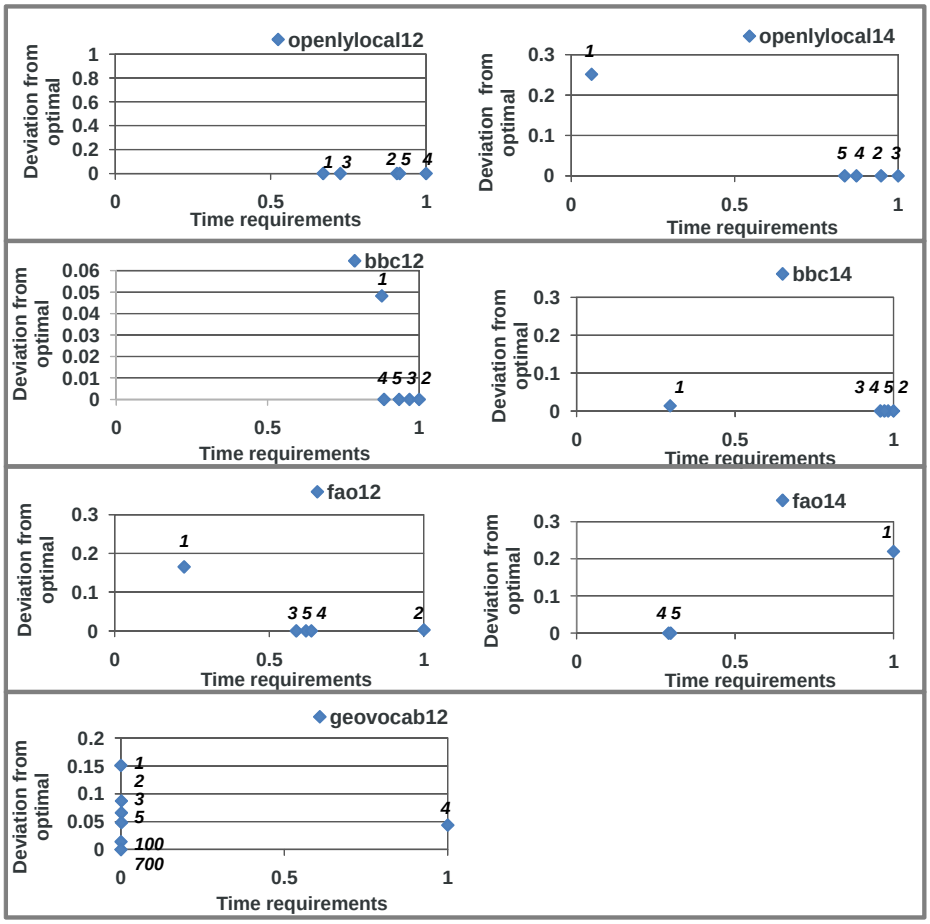


Fig. 15: The trade off for the Real BTC datasets (Isomorphism Task)

Real Datasets. Regarding the real datasets, given in Figure 15, we get the following observations:

- As radius r gets higher values the deviation from the optimum is always reduced.
- There is a value for radius, say r_{opt}, for which $dr_{opt}.y$ becomes 0 (i.e. the optimal map is computed). This value is on average between 2 or 3.
- For values of radius $r > r_{opt}$ it stands $dr.x = 0$.
- As r gets higher values and remains lower than $\texttt{dp}_{\texttt{max}}$ the time efficiency will possibly improve, as more BComponents are detected).

BGen Datasets. Figure 16 shows the tradeoff for the synthetic BGen datasets. For radius $r = 2$ the deviation from the optimal delta is decreased for all the datasets. For higher radius values (i.e. $r = 3$) the time is increased. This is reasonable, as the average depth of all the datasets (see Table 3) is greater than 5 and therefore the algorithm searches in more depth for each possible *Exact* Match.

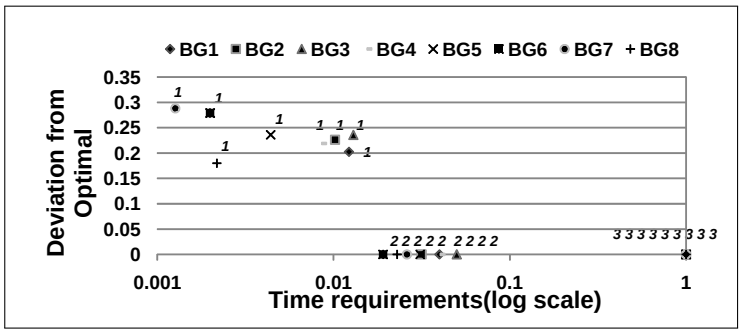


Fig. 16: The trade off for the Synthetic BGen datasets

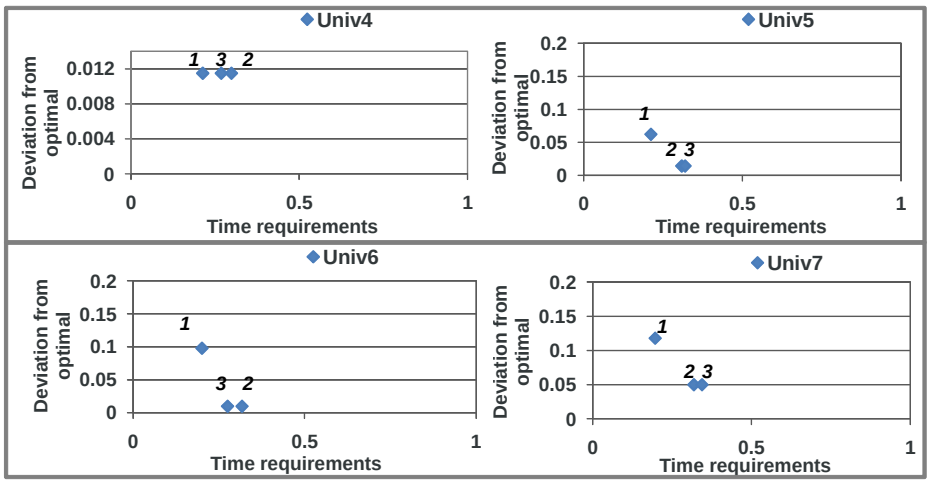


Fig. 17: The trade off for the Synthetic University datasets

University Datasets. Figure 17 shows the tradeoff for the synthetic University datasets. The deviation from the optimum is reduced as radius gets higher values. However, for radius $r > 3$ no more reduction of the deviation can be achieved.

4.6 Discussion and Guidelines for Selecting the Radius to Use

From the evaluation it is made clear that the proposed algorithm `r-SIGN` offers a fair compromise between deviation from the optimum and time efficiency, in comparison to the other algorithms (original `SIGN` and `HUNG`). Note that the algorithm behaves well even for graphs that contain bnodes that form cycles (i.e. `fao14` and `BGen` datasets). The experimental results confirm the need for customized radius selection. Here we provide some guidelines that can help the user of the algorithm to decide how to *configure the radius*. The objective is to select a radius whose memory requirements will not exceed the available memory, and is big enough for

finding a mapping with the least possible deviation from the optimum. Given two graphs G_1 and G_2, one can select the radius to use according to the following suggestions:

1 If the datasets are small in comparison to the available memory, then use $r_1 = \frac{1}{2}\max(max_{diam(G_1)}, max_{diam(G_2)})$, i.e the half of the maximum diameter of the BComponents of the two graphs. With such a radius the algorithm `r-SIGN`, based on the experiments, gives the smallest deviation from the optimum, since it ensures that all the *BComponent Matches* are detected. The space complexity that is imposed by the algorithm can be estimated as $O(n)$ in the best case and $O(n^2)$ in the worst case.
2 If the radius r_1, as defined in Step 1, is still big for the available memory, a second choice is to investigate whether the value $r_2 = \frac{1}{2}\max(\overline{diam(G_1)}, \overline{diam(G_2)}$, allows the algorithm to run using the available memory.
3 If the radius r_2 as defined in Step 2 is still big, then $r = 2$ is suggested.

Of course, the `r-SIGN` gives the ability to the graph database managers to use the desired radius based on application context and the corresponding non functional requirements.

5 Concluding Remarks

In this paper we introduced a radius-based approach for blank node matching. Contrary to the previous blank node algorithms, the introduced algorithm extends to sub-graphs of higher diameter for offering better discrimination ability to each blank node which in turn yields better mappings and consequently smaller delta size. This is significantly important for datasets that contain connected blank nodes, including cyclic structures. The algorithm is parametric with respect to the radius, and this allows controlling the deviation from optimality and the time complexity. Another important characteristic of the algorithm is that the radius does not increase exponentially the required space enabling its application over big datasets. It is worth stressing that this is the first time that bnode matching is applied to millions of blank nodes.

The introduced algorithm has been comparatively evaluated upon real and synthetically generated datasets of variable complexity enabling us to understand the tradeoff between radius, deviation from the optimal delta, and time efficiency. Over the isomorphic datasets, the proposed algorithm always found the optimal solution for radius close to the average depth of the contained BComponents. In non-isomorphic datasets it reduced the delta size from 2 to 28 times in comparison to the older algorithms with reasonable time requirements.

Overall, the proposed algorithm can be useful in various fundamental tasks for effective information linking and integration in the semantic web. Various directions are worth further research, including the identification of special cases where optimality is guaranteed, and methods for parallelizing the task of bnode matching in the context of a parallel matching or differential algorithm [15, 1].

References

1. Jinhyun Ahn, Dong-Hyuk Im, Jae-Hong Eom, Nansu Zong, and Hong-Gee Kim. G-Diff: A Grouping Algorithm for RDF Change Detection on MapReduce. In *4th Joint International Conference on Semantic Technology (JIST-2014)*, pages 230–235, 2014.
2. T. Berners-Lee and D. Connolly. Delta: An Ontology for the Distribution of Differences Between RDF Graphs, 2004. Available at http://www.w3.org/DesignIssues/Diff (version: 2004-05-01).
3. Jeremy J. Carroll. Matching RDF Graphs. In *First International Semantic Web Conference (ISWC-2002)*, pages 5–15, 2002.
4. Lei Chen, Haifei Zhang, Ying Chen, and Wenping Guo. Blank Nodes in RDF. *Journal of Software (JSW)*, 7(9):1993–1999, 2012.
5. Luca Costabello. Error-tolerant rdf subgraph matching for adaptive presentation of linked data on mobile. In *The Semantic Web: Trends and Challenges*, pages 36–51. Springer, 2014.
6. Richard Cyganiak, David Wood, and Markus Lanthaler. RDF 1.1 Concepts and Abstract Syntax. W3C Recommendation, 25 February 2014. Available at http://www.w3.org/TR/rdf11-concepts/.
7. Carina F. Dorneles, Rodrigo Gonçalves, and Ronaldo dos Santos Mello. Approximate Data Instance Matching: A Survey. *Knowledge and Information Systems*, 27(1):1–21, 2011.
8. Daniel Engmann and Sabine Maßmann. Instance Matching with COMA++. In *Datenbanksysteme in Business, Technologie und Web (BTW-2007), Workshop Proceedings*, pages 28–37, 2007.
9. Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. LUBM: A Benchmark for OWL Knowledge Base Systems. *Journal of Web Semantics*, 3(2-3):158–182, 2005.
10. F. Harary. *Graph Theory*. Addison-Wesley, 1969.
11. A. Harth. Billion Triples Challenge data set. Downloaded from http://km.aifb.kit.edu/projects/btc-2012/, 2012.
12. Aidan Hogan. Skolemising Blank Nodes while Preserving Isomorphism. *24th International World Wide Web Conference (WWW-2015)*, 2015.
13. Aidan Hogan, Marcelo Arenas, Alejandro Mallea, and Axel Polleres. Everything you Always Wanted to Know about Blank Nodes. *Journal Web of Semantics*, 27:42–69, 2014.
14. T. Käfer and A. Harth. Billion Triples Challenge data set. Downloaded from http://km.aifb.kit.edu/projects/btc-2014/, 2014.
15. Toralf Kirsten, Lars Kolb, Michael Hartung, Anika Groß, Hanna Köpcke, and Erhard Rahm. Data Partitioning for Parallel Entity Matching. *CoRR*, abs/1006.5309, 2010.
16. Uffe Bro Kjrulff. Triangulation of Graphs - Algorithms Giving Small Total State Space. *Technical Report, Depart. of Computer Science, Aalborg University*, 1990.
17. Michel C. A. Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology Versioning and Change Detection on the Web. In *13th International Conference on Knowledge Engineering and Knowledge Management (EKAW-2002)*, pages 197–212, 2002.
18. Michel C. A. Klein, Dieter Fensel, Atanas Kiryakov, and Damyan Ognyanov. Ontology Versioning and Change Detection on the Web. In *13th International Conference on Knowledge Engineering and Knowledge Management - Ontologies and the Semantic Web (EKAW-2002)*, pages 197–212, 2002.
19. Magnus Knuth, Dinesh Reddy, Anastasia Dimou, Sahar Vahdati, and George Kastrinakis. Towards linked data update notifications. *Workshop on Negative or Inconclusive Results in Semantic Web (NoISE'15)*, 2015.
20. Christina Lantzaki and Yannis Tzitzikas. Tasks that Require or can Benefit from Matching Blank Nodes. *CoRR*, abs/1410.8536, 2014.
21. Christina Lantzaki, Thanos Yannakis, Yannis Tzitzikas, and Anastasia Analyti. Generating Synthetic RDF Data with Connected Blank Nodes for Benchmarking. In *11th Extended Semantic Web Conference (ESWC-2014)*, pages 192–207, 2014.
22. Alejandro Mallea, Marcelo Arenas, Aidan Hogan, and Axel Polleres. On Blank Nodes. In *10th International Semantic Web Conference (ISWC-2011)*, pages 421–437, 2011.
23. Natalya Fridman Noy and Mark A. Musen. PromptDiff: A Fixed-point Algorithm for Comparing Ontology Versions. In *8th National Conference on Artificial Intelligence and 14th Conference on Innovative Applications of Artificial Intelligence*, pages 744–750, 2002.
24. Reinhard Pichler, Axel Polleres, Fang Wei, and Stefan Woltran. dRDF: Entailment for Domain-Restricted RDF. In *5th European Semantic Web Conference (ESWC-2008)*, pages 200–214, 2008.

25. Siddharthan Ramachandramurthi. The Structure and Number of Obstructions to Treewidth. *SIAM Journal on Discrete Mathematics*, 10(1):146–157, 1997.
26. Giovanni Tummarello, Christian Morbidoni, Reto Bachmann-Gmür, and Orri Erling. RDFSync: Efficient Remote Synchronization of RDF Models. In *6th International Semantic Web Conference, 2nd Asian Semantic Web Conference, (ISWC-2007) + (ASWC-2007)*, pages 537–551, 2007.
27. Y. Tzitzikas, C. Alloca, C. Bekiari, Y. Marketakis, P. Fafalios, M. Doerr, N. Minadakis, T. Patkos, and L. Candela. Integrating heterogeneous and distributed information about marine species through a top level ontology. In *Proceedings of the 7th Metadata and Semantic Research Conference (MTSR'13)*, Thessaloniki, Greece, November 2013.
28. Yannis Tzitzikas, Christina Lantzaki, and Dimitris Zeginis. Blank Node Matching and RDF/S Comparison Functions. In *11th Intern. Semantic Web Conference (ISWC-2012)*, pages 591–607, 2012.
29. Max Volkel and Tudor Groza. SemVersion: RDF-based Ontology Versioning System. In *IADIS International Conference WWW/Internet (ICWI-2006)*, 2006.
30. Dimitris Zeginis, Yannis Tzitzikas, and Vassilis Christophides. On Computing Deltas of RDF/S Knowledge Bases. *ACM Transactions on the Web (TWEB)*, 5(3):14, 2011.

A Algorithm Signature

Algorithm 5: `Signature(`b`,` G`)`

```
1  sign(b) ← ∅                                              ▷ signature of b
2  Types ← ⟨Class(b), In(b), Out(b)⟩                        ▷ categories of DNG
3  L, L_Class(b), L_In(b), L_Out(b) ← ∅                     ▷ sets of tuples
4  foreach type ∈ Types do
5      foreach t_i ∈ type do
6          b' ← AdjacentNode(t_i, b)
7          L_type.add((l_b(t_i), b'))
8      sort(L_type)                                         ▷ sort pairs in terms of the first element
9  foreach type ∈ Types do
10     foreach (l, b') ∈ L_type do
11         if ∃x such that (x, b') ∈ L then
12                                                          ▷ x is unique by construction
13             L.remove((x, b'))
14             L.add((x + * + l, b'))
15         else
16             if (b' ∈ B) then
17                 L.add((l, b'))
18     foreach ((x, b') ∈ L) do
19         L.remove((x,b'))
20         L.add((x+◇,b'))
21 sort(L)                                                  ▷ sort pairs in terms of the first element
22 foreach type ∈ Types do
23     foreach (l, b') ∈ L_type do
24         sign(b) ← sign(b) + l                            ▷ concatenate the labels
25         if (b' ∈ B) then
26             sign(b) ← sign(b) + pos(b', L)
27         sign(b) ← sign(b) + *                            ▷ delimiter between labels
28     sign(b) ← sign(b) + ◇                                ▷ delimiter between categories
29 return sign(b)
```

Algorithm *Signature* (Alg. 5) takes as input a bnode b and its graph G and produces $sign(b)$. Firstly, at line 2, it classifies the triples of $DNG(b, G)$ into the three categories, $Class(b), In(b)$ and $Out(b)$, as given in Def. 4, and for each category an ordered set is initialized (line 3), i.e. $L_{Class(b)}$, $L_{In(b)}$, $L_{Out(b)}$ respectively. These ordered sets store one pair of the form ($l_b(t_i)$, b') for each triple t_i that is part of the respective category (line 7), where b' is the adjacent node of b through the triple t_i (line 6) and $l_b(t_i)$ is the label of t_i in terms of b, as defined in Def. 5. Each set is sorted lexicographically in terms of its labels (line 8).

As an optimization to the algorithm given in [28], the algorithm also creates a list L, which is useful to produce counters for the adjacent bnodes of b and thereby ensures the deterministic construction of the signature. In particular, L stores one pair of the form ($labels$, b') for each adjacent bnode b', where $labels$ is a strict concatenation of all the labels of the triples that contain both b and b'. Lines 9-19 construct each pair of L by traversing the already ordered sets $L_{Class(b)}$, $L_{In(b)}$, $L_{Out(b)}$. For each pair (l,b') of these lists if b' is not yet part of L a new pair (l,b') is added to L (line 15-16). Alternatively, if a pair (x,b') already exists in L (line 11), the label l is concatenated with x and the new pair replaces the old one (lines 12-13). Notice that delimiters are also added between the labels (line 13) and between the categories (line 17-19). Afterwards, L is sorted in terms of its labels (line 20) and by this new contribution the adjacent nodes of b are ordered in a deterministic way.

Finally, the ordered sets $L_{Class(b)}$, $L_{In(b)}$, $L_{Out(b)}$ are once more traversed (lines 21-22). For each pair (l,b') of these sets l is concatenated to the signature $sign(b)$ (line 23). In case the adjacent node b' is a bnode (line 24) the position of b' in L is also added next to l (line 25). In order to ensure a deterministic construction of the signature, delimiters are also added between the labels (line 26) and between the categories (line 27). The signature $sign(b)$ is returned.

B Proofs

Prop. 1 *Bnodes in BComponents*
Each bnode $b_1 \in B_1$ is part of exactly one BComponent, denoted as $BC(b)$ (obviously $BC(b) \subseteq G_1$).
Proof:
From Def. 6 we get that a BComponent is a maximal set of connected *bbtriples*. A node of a graph cannot be part of two different maximal sets. □

Prop. 2 *Properties of Maximum Neighborhood Graph*

p1: $\exists\ r_{max} \geq 1$ s.t. $\forall\ r \geq r_{max}\ RNG(b, r, G) = RNG(b, r_{max}, G) = MNG(b, G)$
p2: $MNG(b, G) \supseteq BC(b)$
p3: $nodes(MNG(b, G)) \cap B = nodes(BC(b))$

Proof:

p1: It is clear from Def. 8 that r_{max} gives the maximum element of the Radius Neighborhood Graph and therefore each higher value will not make a difference.
p2: It holds because $MNG(b, G)$ contains not only all the bbtriples of the BComponent, but also all the btriples that are directly connected to the *bbtriples*.
p3: Since we get from p2 of Prop. 2 that $MNG(b, G)$ is a superset of a BComponent, it entails that it contains all the bnodes of the BComponen and in addition it cannot contain more bnodes. □

Prop. 3 *Properties of Matches*

p1: $\approx_{r'} \subseteq \approx_r$, $\forall\ r$, r', where $r' \geq r \geq 1$
p2: $\approx_r^{r'} \subseteq \approx_{r'}$, $\forall\ r$, r', where $r' \geq r \geq 1$
p3: $\approx_r^{r_b} \subseteq \approx_r^{r_a}$, $\forall\ r_a$, r_b, where $r_b \geq r_a \geq 1$
p4: $\approx_r^{r_b} \subseteq \approx_{r'}^{r_b}$, $\forall\ r_b$, r, r' where $1 \leq r_b \leq r \leq r'$
p5: $\approx_r^B \subseteq \approx_r$, $\forall\ r \geq 1$
p6: $b_1 \approx_r^B b_2 \Rightarrow \exists\ m{:}nodes(BC(b_1)) \rightarrow nodes(BC(b_2))$ s. t. $b_3 \approx_r m(b_3), \forall\ b_3 \in nodes(BC(b_1))$
p7: $\approx_{r_a}^B \subseteq \approx_{r_b}^B$, $\forall\ r_a$, r_b, where $1 \leq r_a \leq r_b$

Proof:

p1: $r' \geq r \geq 1 \xrightarrow{Def.7} RNG(b,r,G) \subseteq RNG(b,r',G)$ (1)
$b_1 \approx_{r'} b_2 \xrightarrow{Def.9}$ RNG(b_1,r',G_1) $\equiv$ RNG(b_2,r',G_2) $\xrightarrow{(1)}$
RNG(b_1,r,G_1) $\equiv$ RNG(b_2,r,G_2) $\xrightarrow{Def.9}$ $b_1 \approx_r b_2$. As a result, we get $\approx_{r'} \subseteq \approx_r$.

p2: $b_1 \approx_r^{r'} b_2 \xrightarrow{Def.9} RNG(b_1,r',G_1) \equiv RNG(b_2,r',G_2) \xrightarrow{Def.9} b_1 \approx_{r'} b_2$. As a result, we get $\approx_r^{r'} \subseteq \approx_{r'}$. As a result, we get $\approx_r^{r'} \subseteq \approx_{r'}$.

p3: $r_b \geq r_a \Rightarrow RNG(b,r_a,G) \subseteq RNG(b,r_b,G)$ (1)
$b_1 \approx_r^{r_b} b_2 \xrightarrow{Def.9} RNG(b_1,r_b,G_1) \equiv RNG(b_2,r_b,G_2) \xrightarrow{(1)} RNG(b_1,r_a,G_1) \equiv RNG(b_2,r_a,G_2)$
$\Rightarrow b_1 \approx_r^{r_a} b_2$. As a result, we get $\approx_r^{r_b} \subseteq \approx_r^{r_a}$.

p4: $b_1 \approx_r^{r_b} b_2 \xrightarrow{Def.9}$ RNG(b_1,r_b,G_1) $\equiv$ RNG(b_2,r_b,G_2) and $RNG(b_1,r',G_1) \not\equiv RNG(b_2,r',G_2)$
$\xrightarrow{Def.9} b_1 \approx_{r'}^{r_b} b_2$. As a result, we get $\approx_r^{r_b} \subseteq \approx_{r'}^{r_b}$.

p5: $b_1 \approx_r^B b_2 \xrightarrow{Def.9} RNG(b_1,r,G_1) \equiv RNG(b_2,r,G_2) \xrightarrow{Def.9} b_1 \approx_r b_2$. As a result,$\approx_r^B \subseteq \approx_r$. As a result, we get $\approx_r^B \subseteq \approx_r$.

p6: $b_1 \approx_r^B b_2 \xrightarrow{Def.9} \left.\begin{array}{c} RNG(b_1,r,G_1) = MNG(b_1,G_1) \\ RNG(b_2,r,G_2) = MNG(b_2,G_2)(2) \end{array}\right\} \Rightarrow$
$b_1 \approx_r^B b_2 \xrightarrow{Def.9} b_1 \approx_r b_2 \xrightarrow{Def.9} \exists\ m : nodes(BC(b_1)) \rightarrow nodes(BC(b_2))$ s.t.
$RNG(b_1,r,G_1)$, $RNG(b_2,r,G_2)$ are equivalent $\xrightarrow{(2)} \forall\ (b,b') \in m$
$\begin{array}{c} RNG(b,r,G_1) \subseteq MNG(b_1,G_1) = RNG(b_1,r,G_1) \\ RNG(b',r,G_2) \subseteq MNG(b_2,G_2) = RNG(b_2,r,G_2)\ (4) \end{array}$. Since $RNG(b_1,r,G_1)$, $RNG(b_2,r,G_2)$ are equivalent it follows that $RNG(b,r,G_1) \equiv RNG(b',r,G_2)$. This implies that $b \approx_r m(b')$ $\Rightarrow \exists\ m : nodes(BC(b_1)) \Rightarrow nodes(BC(b_2))$ s.t. $b' \approx_r m(b')$, $\forall\ b' \in nodes(BC(b_1))$.

p7: $b_1 \approx_{r_a}^B b_2 \xrightarrow{Def.9} \left.\begin{array}{c} RNG(b_1,r_a,G_1) = MNG(b_1,G_1) \\ RNG(b_2,r_a,G_2) = MNG(b_2,G_2)\ (1) \end{array}\right\} \xrightarrow{r_b \geq r_a, Prop.2}$
$\begin{array}{c} RNG(b_1,r_b,G_1) = MNG(b_1,G_1) \\ RNG(b_2,r_b,G_2) = MNG(b_2,G_2)\ (2) \end{array}$
$b_1 \equiv_B^{r_a} b_2 \xrightarrow{Def.9} b_1 \approx_{r_a} b_2 \xrightarrow{Def.9} RNG(b_1,r_a,G_1)$, $RNG(b_2,r_a,G_2)$ are equivalent $\xrightarrow{(1)}$ $MNG(b_1,G_1)$, $MNG(b_2,G_2)$ are equivalent $\xrightarrow{(2)} RNG(b_1,r_b,G_1)$, $RNG(b_2,r_b,G_2)$ are equivalent $\xrightarrow{Def.9} b_1 \approx_{r_b} b_2 \xrightarrow{(2),Def.9} b_1 \approx_{r_b}^B b_2$. As a result, $\approx_{r_a}^B \subseteq \approx_{r_b}^B$. □

Prop. 4 *DNG and Signatures*
For two bnodes $b_1 \in B_1$ and $b_2 \in B_2$ it holds that $DNG(b_1,G_1) \equiv DNG(b_2,G_2) \Leftrightarrow sign(b_1) = sign(b_2)$.
Proof:
Let us first show that if $DNG(b_1,G_1) \equiv DNG(b_2,G_2)$ then $sign(b_1) = sign(b_2)$. Since the Direct Neigborhood Graphs are equivalent we get from Definition 2 that there is a bijection m between the terms of the two graphs s.t. $m(u) = u\ \forall\ u \in U \cup L$ and m maps bnodes to bnodes so that it holds that a triple (s, p, o) $\in DNG(b_1,G_1)$ if and only if a triple ($m(s)$, p, $m(o)$) $\in DNG(b_2,G_2)$. Thus, for each triple t = (s, p, o) $\in DNG(b_1,G_1)$ there is a triple t' = ($m(s)$, p, $m(o)$) $\in DNG(b_2,G_2)$. It holds that $l_{b_1}(t) = l_{b_2}(t')$, since if s is a Uri or a literal, $m(s) = s$. Same goes for o. If s is a bnode then $m(s)$ is also a bnode and the representation for all the bnodes is the character •. Same goes for o. Both signatures $sign(b_1)$ and $sign(b_2)$ are going to have the same substrings but now we have to prove that these substrings will be ordered in the same way. As long as the $DNGs$ are equivalent the subgraphs of their incoming and outgoing triples are also equivalent. So, each set of substrings is the same, it is sorted lexicographically in the same way ($L_{Class(b_1)} = L_{Class(b_2)}$ and so on), and is separated with the same special characters in the same positions. Regarding the counters that follow the bnodes, as soon as both the lists (L) of b_1 and b_2 contain the same strings the ordering will be the same and by extension the given counters will also be the same inside the signatures. We get that $sign(b_1) = sign(b_2)$.

Let us now show that if $sign(b_1) = sign(b_2)$ then $DNG(b_1,G_1) \equiv DNG(b_2,G_2)$. If we separate the two signatures according to the delimiters then we get all the triple representations of both signatures. Since the signatures are the same, their triple representations are also the same, and by extension both b_1 and b_2 have the same triples in their *Direct Neighborhoods*. Since, the numbering of the adjacent bnodes is the same the map m between the adjacent

bnodes of b_1 and b_2 will form a bijection that guarantees equivalence. From the above we get that the DNGs of b_1 and b_2 are equivalent. □

Prop. 5 *Upper Bound of "Unnamed" Delta*
For any map M_x between two graphs G_1 and G_2 it holds that $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_x)| \leq |btriples(G_1)| + |btriples(G_2)| + |bbtriples(G_1)| + |bbtriples(G_2)|$.
Proof:
In the worst case (e.g. if all bnodes are treated as different, equivalently if $M_x = \emptyset$) the delta will require the deletion of all the triples in G_1 that bnodes participate (i.e. all the *btriples* and *bbtriples*) and the addition of all the triples in G_2 that bnodes participate. □

Prop. 6 *Deviation from Optimal for Equivalent Graphs of* `SIGN` *and* `r-SIGN`
If $G_1 \equiv G_2$ then
(i) $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_{SIGN})| \leq |bbtriples(G_1)| + |bbtriples(G_2)|$.
(ii) $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_{rSIGN})| \leq |bbtriples(G_1)| + |bbtriples(G_2)|$.
This upper bound is actually upper bound of the deviation from the optimal, since the optimal $\Delta_e^{\mathcal{B}}$ is empty.
Proof:
(i) $G_1 \equiv G_2 \xrightarrow{Def.1} \exists\ M : B_1 \rightarrow B_2$ s.t. $\forall\ (b_1, b_2) \in M$ it holds that $DNG(b_1, G_1) \equiv DNG(b_2, G_2) \xLeftrightarrow{Prop.4} \forall\ (b_1, b_2) \in M$ it holds that $sign(b_1) = sign(b_2)$ (1). Since (1) holds we get that `SIGN` will find for each $b_1 \in B_1$ a $b_2 \in B_2$, such that $sign(b_1) = sign(b_2)$. Therefore, M_{SIGN} will contain only *Exact Matches* and $\forall\ (b_1, b_2) \in M_{SIGN}$ it holds that $(b_1, p, o) \in G_1$ and $(b_2, p, o) \in G_2$ and $(s, p, b_1) \in G_1$ and $(s, p, b_2) \in G_2$. In other words, we get that $\forall\ (b_1, b_2) \in M_{SIGN}$ it holds $\begin{matrix}\{(b_1, p, o) \in G_1\} \equiv \{(b_2, p, o) \in G_2\} \\ \{(s, p, b_1) \in G_1\} \equiv \{(s, p, b_2) \in G_2\}\end{matrix}$ $\Rightarrow$ their union is also equivalent or else $\forall\ (b_1, b_2) \in M_{SIGN}$ $btriples(b_1) \equiv btriples(b_2)$. Thus, we get that $|\Delta_e^{\mathcal{B}}(G_1 \rightarrow G_2, M_{SIGN}| \leq |bbtriples(G_1)| + |bbtriples(G_2)|$.
(ii) Since G_1 and G_2 are equivalent for each b_1 in $nodes(BC1)$, where BC_1 is a BComponent of G_1, $rSIGN$ will find and will make either
1) an Exact r-Match with a bnode b_2 in $nodes(BC2)$, where BC_2 is a BComponent of G_2 or
2) a BC r-Match with a bnode b_2 in $nodes(BC2)$ where BC_2 is a BComponent of G_2 .
In case 1) then from the Proof of (i) we know that the $btriples(b1) \equiv btriples(b2)$ and only the $bbtriples(b1) \cup bbtriples(b2)$ are possibly in Delta.
In case 2) the Bcomponent Match ensures that there is a map $m : nodes(BC1) \rightarrow nodes(BC2)$ such that each $(b_1, b_2) \in m$ makes an exact Match. Again from i) we get that $btriples(b_1) \equiv btriples(b_2)$. Thus again only $bbtriples(b_1)$ and $bbtriples(b_2)$ are possibly in Delta. □

Prop. 7 `SIGN` *vs* `r-SIGN` *if no connected bnodes*
If G_1 and G_2 have no connected bnodes, i.e. none of them contains any *bbtriple*, then $M_{SIGN} = M_{rSIGN}$ for any $r \geq 2$.
Proof:
The constructed signatures will be the same. What `SIGN` reports as *Exact Match* is what `r-SIGN` reports as *BC r-Match*. The *Closest Matches* of `SIGN` are the same those of `r-SIGN` since they will be constructed in the same way. □

Prop. 8 `SIGN` *vs* `r-SIGN`
If $G_1 \equiv G_2$ then `r-SIGN` will find the maximum in diameter (say $dmax$) BC $dmax$-Match if it is unique, and $r \geq dmax$. This is not always true for `SIGN`.
Proof:
`r-SIGN` (for $r \geq dmax$) first looks for BC r-Matches. It will always find the BC $dmax$-Match and report it in M_{rSIGN}. It is not possible to miss it, because there is not any other bcomponent with the same or bigger diameter. Instead, `SIGN` makes only exact matches, therefore a different matching could occur. □