11th International Semantic Web Conference (ISWC'12),
Nov 2012, Boston USA

Blank Node Matching and RDF/S Comparison Functions

Yannis Tzitzikas, Christina Lantzaki, and Dimitris Zeginis*

Computer Science Department, University of Crete,
Institute of Computer Science, FORTH-ICS, GREECE
{tzitzik,kristi,zeginis}@ics.forth.gr

Abstract. In RDF, a *blank node* (or anonymous resource or bnode) is a node in an RDF graph which is not identified by a URI and is not a literal. Several RDF/S Knowledge Bases (KBs) rely heavily on blank nodes as they are convenient for representing complex attributes or resources whose identity is unknown but their attributes (either literals or associations with other resources) are known. In this paper we show how we can exploit blank nodes anonymity in order to reduce the delta (diff) size when comparing such KBs. The main idea of the proposed method is to build a mapping between the bnodes of the compared KBs for reducing the delta size. We prove that finding the optimal mapping is NP-Hard in the general case, and polynomial in case there are not directly connected bnodes. Subsequently we present various polynomial algorithms returning approximate solutions for the general case.
For making the application of our method feasible also to large KBs we present a signature-based mapping algorithm with $n \log n$ complexity. Finally, we report experimental results over real and synthetic datasets that demonstrate significant reductions in the sizes of the computed deltas. For the proposed algorithms we also provide comparative results regarding delta reduction, equivalence detection and time efficiency.

1 Introduction

The ability to compute the differences that exist between two RDF/S Knowledge Bases (KBs) is an important step to cope with the evolving nature of the Semantic Web (SW). In particular, RDF/S Deltas can be employed to aid humans understand the evolution of knowledge, and to reduce the amount of data that need to be exchanged and managed over the network in order to build SW synchronization [19, 1], versioning [7, 8, 1, 4, 21] and replication [17] services. A rather peculiar but quite flexible feature of RDF is that it allows the representation of *unnamed* nodes, else called *blank nodes* (for short bnodes), a feature that is convenient for representing complex attributes (e.g. an attribute `address` as shown in Figure 1) without having to name explicitly the auxiliary node that is used for connecting together the values that constitute the complex value (i.e.

* Current affiliation: Information Systems Lab, University of Macedonia, Thessaloniki, Greece, zeginis@uom.gr

the particular `street, number` and `postal code` values). A recent paper [10] that surveys the treatment of bnodes in RDF data, proves that blank nodes is an inevitable reality. Just indicatively, and according to their results, the data fetched from the "hi5.com" domain consist of 87.5% of blank nodes, while those from the "opencalais.com" domain, which is part of LOD (Linked Open Data) cloud, has 44.9% bnodes. The authors also state that the inability to match bnodes increases the delta size and does not assist in detecting the changes between subsequent versions of a KB.

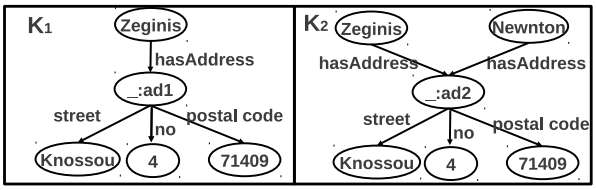

Fig. 1. Examples of blank nodes

Previous works on comparing RDF KBs have not elaborated on this issue thoroughly. There are works (e.g. [21, 22]) proposing differential functions that yield reduced in size deltas (in certain cases) but treat bnodes as named nodes. Other works and systems (specifically Jena [3]) focus only on deciding whether two KBs that contain bnodes are equivalent or not, and do not offer any delta size saving for the case where the involved KBs are not equivalent. In brief, and to the best of our knowledge, there is not any work that attempts to establish a bnode mapping for reducing the delta size for the case of non equivalent KBs. Note that finding such a mapping can be considered as a preprocessing step, a task that is carried out before a differential function (like those described in [17, 20, 16, 15, 8, 21]) is applied.

We prove that finding the optimal mapping is NP-Hard in the general case and polynomial if there are not directly connected bnodes. Subsequently we present various polynomial algorithms returning approximate solutions for the general case. For making the application of this method feasible also to large KBs one of these algorithms has $n \log n$ complexity.

The experimental results over real and synthetic datasets showed that our method significantly reduces the sizes of the computed deltas, while the time required is affordable (just indicatively the $n \log n$ algorithm requires a few seconds for KBs with up to 150,000 bnodes). For the proposed algorithms we also provide comparative results regarding time efficiency and their potential for delta reduction and equivalence detection.

The rest of this paper is organized as follows. Section 2 discusses RDF KBs with bnodes and the equivalence of such KBs. Section 3 elaborates on the problem of finding the optimal mapping. Section 4 proposes bnode matching algorithms and Section 5 reports experimental results. Section 6 discusses the applicability of the method at the presence of inference rules and various semantics, Section 7 discusses related work, and finally, Section 8 concludes the paper and identifies issues for further research.

Software and datasets are available to download and use from `http://www.ics.forth.gr/isl/BNodeDelta`.

2 RDF KBs with Blank Nodes

Consider there is an infinite set U (RDF URI references), an infinite set B (blank nodes) and an infinite set L (literals). A triple $(s, p, o) \in (U \cup B) \times U \times (U \cup B \cup L)$ is called an *RDF triple* (s is called the *subject*, p the *predicate* and o the *object*). An RDF Knowledge Base (KB) K, or equivalently an *RDF graph* G, is a set of RDF triples.

For an RDF Graph G_1 we shall use U_1, B_1, L_1 to denote the URIs, bnodes and literals that appear in the triples of G_1 respectively. The *nodes* of G_1 are the values that appear as subjects or objects in the triples of G_1.

The equivalence of *RDF graphs* that contain *blank nodes* is defined in [9] as:

Def. 1 (Equivalence of RDF Graphs that contain Bnodes)
Two RDF graphs G_1 and G_2 are *equivalent* if there is a bijection[1] M between the sets of nodes of the two graphs (N_1 and N_2), such that:
- $M(uri) = uri$ for each $uri \in U_1 \cap N_1$
- $M(lit) = lit$ for each $lit \in L_1$
- M maps bnodes to bnodes (i.e. for each $b \in B_1$ it holds $M(b) \in B_2$)
- The triple (s, p, o) is in G_1 if and only if the triple $(M(s), p, M(o))$ is in G_2.

$\diamond$

It follows that if two graphs are equivalent then it certainly holds $U_1 = U_2$, $L_1 = L_2$ and $|B_1| = |B_2|$.

Let us now relate the problem of equivalence with *edit distances*.

Def. 2 (Edit Distance over Nodes given a Bijection)
Let o_1 and o_2 be two nodes of G_1 and G_2, and suppose a bijection between the nodes of these graphs, i.e. a function $h : N_1 \rightarrow N_2$ (obviously $|N_1| = |N_2|$). We define the edit distance between o_1 and o_2 over h, denoted by $dist_h(o_1, o_2)$, as the number of additions or deletions of triples which are required for making the "direct neighborhoods" of o_1 and o_2 the same (considering h-mapped nodes the same). Formally, $dist_h(o_1, o_2) =$
$|\{(o_1, p, a) \in G_1 \mid (o_2, p, h(a)) \notin G_2\}| + \quad |\{(a, p, o_1) \in G_1 \mid (h(a), p, o_2) \notin G_2\}| +$
$|\{(o_2, p, a) \in G_2 \mid (o_1, p, h^{-1}(a)) \notin G_1\}| + |\{(a, p, o_2) \in G_2 \mid (h^{-1}(a), p, o_1) \notin G_1\}|$ $\diamond$

Now recall that if G_1 is equivalent to G_2 then there exists a bijection h such that $(a, p, b) \in G_1 \Leftrightarrow (h(a), p, h(b)) \in G_2$. We will denote this by $G_1 \equiv_h G_2$. It follows that:

Theorem 1 (RDF Graph Equivalence and Edit Distance).
$G_1 \equiv_h G_2 \Leftrightarrow dist_h(o, h(o)) = 0$ for each $o \in N_1$.

Obviously the above theorem is useful for the case where the bijection h respects the constraints of Def. 1 (i.e. maps named elements to named elements, and anonymous elements to anonymous).

[1] A function that is both one-to-one (injective) and onto (surjective).

3 On Finding the Optimal Bnode Mapping

Let us now focus on the case where two KBs, K_1 and K_2, are *not necessarily equivalent* and do contain bnodes. We would like to find a mapping over their bnodes that reduces the size (i.e. the number of change operations) of their delta and allows detecting whether K_1 is equivalent to K_2. Furthermore we want an efficient (tractable at least) method for finding such a mapping.

3.1 RDF/S Differential Functions.

[21, 22] described and analyzed various differential functions for comparing RDF/S knowledge bases. Each differential function returns a set of primitive change operations, i.e. $Add(t)$ and $Del(t)$ where t is an RDF triple. For the needs of this paper, it is enough to use the differential function Δ_e which is defined as follows ("$-$" denotes set difference): $\Delta_e(K_1 \rightarrow K_2) = \{Add(t) \mid t \in K_2 - K_1\} \cup \{Del(t) \mid t \in K_1 - K_2\}$. We call its output *delta*.

3.2 Bnode Name Tuning and Delta Reduction Size

The basic idea for reducing the delta is the following: if we match a bnode b_1 (of B_1) to a bnode b_2 (of B_2), through a bijection M, then these bnodes can be considered as equal at the computation of delta. For example, if K_1 contains a triple $(b_1, name, Joe)$ and K_2 contains a triple $(b_2, name, Joe)$ and we match b_1 to b_2, then these two triples will be considered equal and thus no difference will be reported. However we should note that in the context of versioning or synchronization services the change operations derived by a differential function *should not* be used as they are. For example, consider $K_1 = \{(b_1, name, Joe)\}$ and $K_2 = \{(b_2, name, Joe), (b_2, lives, UK)\}$ and suppose that we match again b_1 to b_2. In this case a mapping-aware comparison function will return the delta $\{Add((b_2, lives, UK))\}$. If we want to apply it on K_1 then we have to replace b_2 by b_1, i.e we should apply on K_1 the operation $Add((b_1, lives, UK))$, and in this way, we will obtain $K_1' = \{(b_1, name, Joe), (b_1, lives, UK)\}$ which is equivalent to K_2. We call this step *Bnode Name Tuning*, and it actually replaces (renames) in the delta the local names of the bnodes of B_2 by the local names of the matched bnodes in B_1. In this way the delta does not need any *rename* operation (i.e. $rename(b_1, b_2)$) and hence not any particular execution order.

Delta reduction size Bnode matching cannot increase delta size. Without bnode matching any pair of bnodes from different KBs is considered different, and thus all triples to which they participate will be different and reported as change operations in the delta. On the other hand, if two bnodes are matched then the delta size is reduced if they participate to triples with the same predicate and the same other node (i.e. the same *subject* or *object*). In the case where all predicates/nodes of these triples are different, the delta size that will be reported is what will be reported without bnode matching.

3.3 Bnode Matching as an Optimization Problem

Here we formulate the problem of finding a mapping between the bnodes of two KBs as an optimization problem. Let $n_1 = |B_1|$, $n_2 = |B_2|$ and $n = min(n_1, n_2)$. We have to match n elements of B_1 with n elements of B_2, i.e. our objective is to find the unknown part of the bijection M. To be more precise, M a priori contains the mappings of all the URIs and literals of the KBs (URIs and literals are mapped as an identity function as in Def. 1), and its unknown part concerns B_1 and B_2. Suppose that $n = n_1 < n_2$. Let $\mathcal{J}$ denote the set of all possible bijections between B_1 and a subset of B_2 that comprises n elements. The number of all possible bijections (i.e. $|\mathcal{J}|$) is $n_2 * (n_2 - 1) * ... * (n_2 - n_1 + 1)$, i.e. the first element of B_1 can be matched with n_2 elements of B_2, the second with $n_2 - 1$ elements, and so on. Consequently, the set of candidate solutions is exponential in size. Since our objective is to find a bijection $M \in \mathcal{J}$ that reduces the delta size (as regards the "unamed" parts of the KBs), we define the cost of a bijection M as follows:

$$Cost(M) = \sum_{b_1 \in B_1} dist_M(b_1, M(b_1)) \quad (1)$$

Def. 3 (The bijection yielding the less delta size) The best solution (or solutions) is defined as the bijection with the minimum cost, i.e. we define:

$$M_{sol} = \arg_M \min_{M \in \mathcal{J}} (Cost(M)) \qquad \diamond$$

The notation arg_M returns the M in $\mathcal{J}$ that gives the minimum cost.

Theorem 2 (Equivalence and Mapping Cost).
If $G_1 \equiv_{M_{sol}} G_2$ (according to Def. 1) then $Cost(M_{sol}) = 0$.

The proof follows easily from the definitions. It is also clear that the inverse of Th. 2 does not hold (i.e. $Cost(M_{sol}) = 0 \not\Rightarrow G_1 \equiv_{M_{sol}} G_2$) because the cost is based on the distance between the direct neighborhoods of the blank nodes *only*, and not between the named parts of the graphs.

From the algorithmic perspective, one naive approach for finding the best solution (i.e. M_{sol}) would be to examine the set of all possible bijections. That would require at least $n!$ examinations (true if $n_1 = n_2 = n$, while if $n_1 < n_2$ then their number is higher than $n!$). However, the problem is intractable in general:

Theorem 3. Finding the optimal bijection (according to Def. 3) is NP-Hard.
Proof*:*
We will show that subgraph-isomorphism *(which is NP-complete problem) can be reduced to the problem of finding the optimal bijection (meaning that our problem is at least as hard as* subgraph-isomorphism*). Let us make the* hypothesis *that we can find the optimal bijection in polynomial time. We will prove that if that hypothesis were true, then we would be able to solve the subgraph isomorphism in polynomial time. The* subgraph isomorphism decision problem *is stated as: given two plain graphs G_1 and G_2 decide whether G_1 is isomorphic to a subgraph of G_2. Let $G_1 = (N_1, R_1)$ and*

$G_2 = (N_2, R_2)$. We can consider these graphs as two RDF graphs such that: all of their nodes are bnodes and all property edges have the same label. Assume that $|N_1| \leq |N_2|$ and let $n = \min(|N_1|, |N_2|)$. If we can find in polynomial time whether there is a bijection between the n nodes of G_1 and n nodes of G_2 such that $Cost(M_{sol}) = 0$, then this means that we have found whether G_1 is isomorphic to a subgraph of G_2. Specifically, to decide whether there is a subgraph isomorphism, (a) we compute the optimal bijection, say M_{sol}, and (b) we compute its cost. If the cost returned by step (b) is 0 then we return YES, i.e. that there is a subgraph isomorphism. Otherwise we return NO (i.e. there is no subgraph isomorphism). Note that step (a) is polynomial by hypothesis*, while step (b) relies on Def. 2 and its cost is again polynomial. Regarding the latter, note that M_{sol} contains n pairs, and to compute $dist_M(b_1, b_2)$ for each (b_1, b_2) pair of M, we consider only the direct neighborhoods of the two nodes in the two graphs (for G_2 we have to consider only those that connect nodes that participate in M_{sol})[2]. It follows that its computational cost is analogous to the number of edges of the graphs, and thus polynomial. Therefore given a bijection M_{sol}, to compute $Cost(M_{sol})$ requires polynomial time. Also note that Th. 1 holds also for plain graphs assuming a distance function over not labeled edges. We conclude that if our hypothesis were true, then we would be able to decide subgraph isomorphism in polynomial time.*

We conclude that finding the optimal bijection is NP-Hard.$\diamond$

Below we will show that there are algorithms of polynomial complexity for a frequently occurring case. For the general case, we will propose algorithms of polynomial complexity that return an approximate solution.

3.4 Polynomially-solved (and Frequently Occurring) Cases

Consider the KBs in Figure 2 and suppose that we want to compute $dist_h(\_ : 1, \_ : 6)$ (according to Def. 2). It is not hard to see that this distance depends on the mappings (by h) of the bnodes that are connected to $\_ : 1$ and $\_ : 6$, i.e. on the mappings of $\_ : 3, \_ : 4, \_ : 8$ and $\_ : 9$. However several datasets do not have directly connected bnodes. For this reason, here we study a variation of the problem that is appropriate for this case. The key point is that the distance between two bnodes does not depend on how the rest bnodes are mapped.

This is very important because in this case we can solve the optimization problem (as defined in Definition 3) using the *Hungarian algorithm* [12] (for short Alg_{Hung}, an algorithm for solving the *assignment problem*. Here the elements (bnodes) of B_1 play the role of *workers*, the elements (bnodes) of B_2 play the role of *jobs*, and the edit distances of the pairs in $B_1 \times B_2$ play the role of the *costs*. Consider for the moment that $|B_1| = |B_2|$. If we compute the edit distances between all possible n^2 pairs, then Alg_{Hung} can find the optimal assignment at the cost of $O(n^3)$ time. This means that finding the optimal solution costs polynomial time. An extension of Alg_{Hung} giving the ability to assign the problem in rectangular matrices (i.e. when $|B1| \neq |B2|$) is already provided in [2]. We conclude that if there are not directly connected bnodes then the optimal mapping can be found in polynomial time.

[2] Alternatively, if $Cost(M_{sol}) \neq 0$ (using the distance as defined in the main paper), we return YES only if $\Delta_e(G_1 \rightarrow G_2)$ as defined in section 3.1, after bnode name tuning, contains only triples each containing one bnode in B_1 and one not in B_1.

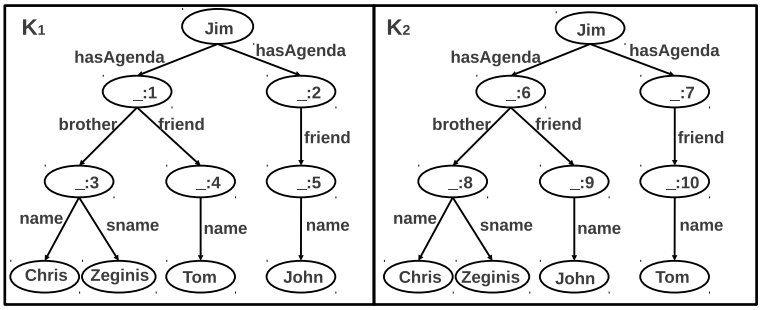

Fig. 2. Two KBs with directly connected bnodes

Theorem 4. Finding the optimal bijection (according to Def. 3) is a polynomial task if there are no directly connected bnodes.◇

4 Bnode Matching Algorithms

At section 4.1 we present a variation of Alg_{Hung} for getting an *approximate* solution for the general case, then at Section 4.2 we present a *signature*-based algorithm appropriate for larger datasets.

4.1 Hungarian BNode Matching Algorithm

We have already stated that Alg_{Hung} can find the optimal mapping in polynomial time if no directly connected bnodes exist in the compared KBs. For the cases where there are directly connected bnodes, Alg_{Hung} enriched with an assumption regarding how to treat the connected bnodes at the computation of $dist_h$, could be used for producing an *approximate* solution. Also in this case the algorithm will make $n_1 \times n_2$ distance computations (where $n_1 = |B_1|$ and $n_2 = |B_2|$), and the complexity of the algorithm will be again $O(n^3)$.

Regarding connected bnodes, at the computation of $dist_h$, one could either assume that all of the connected bnodes are different, or all of them are the same. The first assumption does not require any bijection (h contains only the identity functions of the URIs and literals). According to Definition 2, the fact that all the bnodes are different means by extension that the triples in the direct neighborhoods connecting blank nodes are different too, even in the case where these triples have the same properties. For instance, applying the Definition 2 between bnodes (_ : 1, _ : 6) and (_ : 1, _ : 7) of Figure 2, we get that $dist_h(\_ : 1, \_ : 6) = 4$ and $dist_h(\_ : 1, \_ : 7) = 3$ respectively. However, bnodes _ : 1, _ : 6 have two outgoing triples with exactly the same properties, while bnodes _ : 1, _ : 7 have only one. We observe that this assumption is not very good because we would prefer _ : 1 to be "closer" to _ : 6 than to _ : 7.

According to the alternative assumption, when comparing bnodes (_ : 1, _ : 6) in Figure 2, bnode _ : 3 can be matched either with bnode _ : 8 or with bnode _ : 9, depending on the existence of a common property between them. This yields $dist_h(\_ : 1, \_ : 6) = 0$ since both bnodes have two outgoing triples with

common properties (i.e. $(\_:1, brother, \_:3)$ is matched with $(\_:6, brother, \_:8)$ and $(\_:1, friend, \_:4)$ is matched with $(\_:6, friend, \_:9)$). Regarding $\_:1$ and $\_:7$, we get $dist_h(\_:1, \_:7) = 1$ because of the deleted triple $(\_:1, brother, \_:3)$. It follows that the results of this assumption are better over this example, as $\_:1$ is "closer" to $\_:6$ than to $\_:7$. In general it is better because it exploits common properties, and therefore we adopt this assumption in our experiments.

4.2 A Fast ($O(n \log N)$) Signature-based Algorithm

The objective here is to devise a faster mapping algorithm that could be applied to large KBs, at the cost of probably bigger deltas. We propose a *signature-based* mapping algorithm, for short Alg_{Sign}, which consists of two phases: the signature construction and the mapping construction phase. For each bnode b we produce a string based on the direct neighborhood of b. This string is called the *signature* of bnode b. This phase gives us two lists of signatures, one for the bnodes of each KB. These lists should be considered as bags rather than sets, as there is a probability that two or more bnodes get the same signature. The probability depends on the way the signature is built (we discuss this later).

Alg. SignatureMapping	(9) for *each* $bs_1 \in BS_1$
Input: two sets of bnodes B_1 and B_2, where $\|B_1\| < \|B_2\|$	(10) $bs_2 = Lookup(BS_2, bs_1)$
	(11) if $(bs_2 == bs_1)$ // exact match
Out: a bij. M between B_1 and B_2	(12) $M = M \cup \{(bn_1[bs_1], bn_2[bs_2])\}$
(1) $M = \emptyset$	// $bn_1[str]$ returns the $b \in B_1$ corresponding to str
(2) $BS_1 = BS_2 = emptybag$	(13) $BS_2.remove(bs_2)$
(3) for *each* $b_1 \in B_1$	(14) $BS_1.remove(bs_1)$
(4) $BS_1 = BS_1 \cup \{Signature(b_1)\}$	(15) for *each* $bs_1 \in BS_1$
(5) for *each* $b_2 \in B_2$	(16) $bs_2 = Lookup(BS_2, bs_1)$ // closest match
(6) $BS_2 = BS_2 \cup \{Signature(b_2)\}$	(17) $M = M \cup \{(bn_1[bs_1]), bn_2[bs_2])\}$
(7) $sort(BS_1)$	(18) $BS_2.remove(bs_2)$
(8) $sort(BS_2)$	(19) return M

Fig. 3. Alg. The Signature-based bnode matching algorithm

The mapping phase takes these two bags of strings and compares the elements of the first bag with those of the second. To make *binary search* possible, both bags are sorted lexicographically. In particular, we start from the smaller list, say BS_1, and for each string bs_1 in that list we perform a lookup in the second list BS_2 using *binary search*. If an *exact match* exists (i.e. we found the string bs_1 also in BS_2) we produce a mapping, i.e. the pair $(bn_1[bs_1], bn_2[bs_1])$. Since more than one bnodes may have the same signature we select one. We prefer the order as provided by the managing software, which in many cases reflects the order by which bnodes appear in files. As there is a high probability for subsequent versions to keep the same serialization, using the original order increases the

probability of matches in case of same signatures[3]. We continue in this way for all strings of BS_1. For each element bs_1 of BS_1 for which *no exact match* was found in BS_2 we perform a second lookup over the remainder part of BS_2, say BS'_2, which will produce a mapping based on the *closest* element of BS'_2 to the bs_1 element. Specifically, we will match bs_1 to the element of BS'_2 to which binary search stopped, i.e. to the *lexicographically closer* element. Note that we perform the closest matches after finishing with the exact matches in order to avoid the situation where an approximate match deters an exact match at a later step.

The complexity of this algorithm is $O(n \log N)$ where $N = \max(n_1, n_2)$ and $n = \min(n_1, n_2)$, assuming that the average graph degree of bnodes (and thus signature size) does not depend on N. The algorithm is shown in Figure 3 and relies on an algorithm *Signature* for producing signatures, and on a algorithm *Lookup* as described earlier. As regards the signature construction method we would like to derive strings that allow matches that will yield small deltas even if the bnodes do not match exactly. To this end, we should give priority (i.e. bring to the front part of the string) to the items of the bnode that have lower probability to be changed from one version to the other.

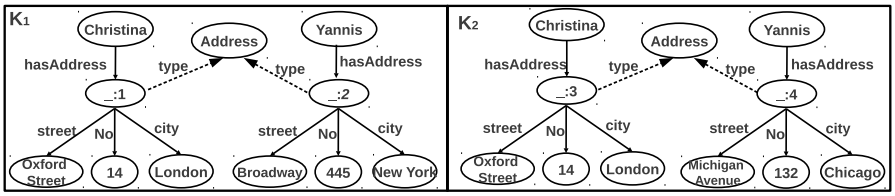

Fig. 4. Two versions of an address Knowledge Base

Table 1. *Signatures* on bnodes of K_1 and K_2 of Fig. 4 according to the given option

Local Name	Signature
_ : 1	$ChristinahasAddress \diamondsuit typeAddress \diamondsuit cityLondon * No14 * streetOxfordStreet$
_ : 3	$ChristinahasAddress \diamondsuit typeAddress \diamondsuit cityLondon * No14 * streetOxfordStreet$
_ : 2	$YannishasAddress \diamondsuit typeAddress \diamondsuit cityNewYork * No445 * streetBroadway$
_ : 4	$YannishasAddress \diamondsuit typeAddress \diamondsuit cityChicago * No132 * streetMichiganAvenue$

Consider bnode _ : 1 of Figure 4 which is involved in the following triples: *Incoming*: $\{(Christina, hasAddress, \_:1)\}$, *Outgoing*: $\{(\_:1, street, OxfordStreet)$, $(\_:1, No, 14)$, $(\_:1, city, London)\}$, *Class Type*: $\{(\_:1, type, Address)\}$. Each of these triples will be mapped to a substring (e.g. "ChristinahasAddress" for the triple $(Christina, hasAddress, \_:1)$). The set *Class Type* contains the triples with the `rdf:type` ("type" in the figure) property of the respective bnode. For the three different sets of triples (Incoming, Outgoing, Class Type) we are going to construct three sets of substrings respectively. The substrings inside each set are sorted lexicographically and separated by a special character, here denoted by $*$. The concatenation of these sets of substrings will yield the signature.

[3] We do the same in Alg_{Hung} in case of ties in costs.

A key point is the order by which the sets are concatenated. One option is to give a first priority to the set of the incoming triples, a second priority to the set with type information (i.e. "typeAddress"), and the last priority to the set of the outgoing triples. We should also mention that inside the signature the sets are separated by a special character, here denoted by $\diamondsuit$. Table 1 shows the signatures of all the bnodes of Figure 4 according to this option. The proposed ordering of the substrings inside the signature stems from the assumption that the probability for the outgoing statements to change is higher than the incoming (e.g. in Figure 4 updating the address of a person is more probable than changing his/her name). Under this assumption the incoming statements should precede the outgoing inside the signature. Similarly for the class type of the bnode, it is not usual to be changed from one version to the other.

We represent the blank nodes which are subjects of incoming statements or objects of the outgoing statements, by a special character $\clubsuit$ (i.e. we treat them as equal, as we did in the 2nd assumption of approximation version of Alg_{Hung}).

5 Experimental Evaluation

Real Datasets. We performed experiments for evaluating the potential for delta reduction, equivalence detection and time efficiency. In our experiments[4], we used two real datasets available in the LOD cloud: the *Swedish open cultural heritage* dataset[5], and the *Italian Museums* dataset[6], published from LKDI[7]. From each one we downloaded two versions with a time difference of one week or month. A preprocessing was necessary for corrections (e.g. missing URIs for some classes) and for merging the files. The features of these two datasets are given in Table 2. In both datasets there are *no directly connected bnodes*.

Table 2. Features of two real LOD datasets

	Swedish		Italian	
	File 1	File 2	File 1	File 2
Date	15/10/11	22/10/11	2/11/11	4/12/11
$\|Triples\|$	3,750	3,589	49,897	49,897
$\|BNodes\|$	535	509	6,390	6,390
$\|Triples$ with $bnodes\|$	77.7%	77.2%	43.85%	43.85%
Total Size	378 KB	365 KB	5.49 MB	5.46 MB

Experiments were conducted *with* and *without* bnode mapping. Regarding mapping we tested: (a) the *random*, (b) the *Hungarian*, and (c) the *Signature*-based mapping methods. The results are shown in Table 3. The first rows show the size of the yielded deltas and the last rows the time required for loading the bnodes (BLoad), constructing signatures (SC), bnode maping (BM), delta

[4] Using Sesame RDF/S Repository (main memory), using a PC with Intel Core i3 at 2.2 Ghz, 3.8 GB Ram, running Ubuntu 11.10.

[5] http://thedatahub.org/dataset/swedish-open-cultural-heritage used from http://kringla.nu/kringla/ for providing information on cultural data of Sweden.

[6] http://thedatahub.org/dataset/museums-in-italy

[7] http://www.linkedopendata.it/

computation (Diff), bnode name tuning (Tuning Time), and the total time. We observe that the algorithms provide a delta of 12.7 to 7,294 times smaller than without bnode mapping. Alg_{Hung} yields an equal (for the Italian) or smaller (0.34 times smaller for the Swedish) delta than Alg_{Sign}, but it requires more time (from 15 to 624 times).

Table 3. Experimental results over real datasets

	Swedish				Italian			
	without BM	*with BM (bnode matching)*			*without BM*	*with BM (bnode matching)*		
		Random	Alg_{Hung}	Alg_{Sign}		*Random*	Alg_{Hung}	Alg_{Sign}
$\|Added\|$	2,805	2,726	75	127	21,885	19,762	3	3
$\|Deleted\|$	2,966	2,887	236	288	21,885	19,762	3	3
$\|\Delta_e\|$	5,771	5,613	311	419	43,770	39,524	6	6
BLoad Time(ms)	-	631	630	634	-	428	423	421
SC Time(ms)	-	-	-	210	-	-	-	840
BM Time(ms)	-	1.3	5,391	130	-	4.9	576,592	82.5
Diff Time(ms)	55	64	30	47	145	166	169	163
Tuning Time(ms)	-	15	0.2	0.5	-	3,332	9.4	9.5
Total Time(ms)	57	715	5,931	1,024	147	3,935	577,197	1,521

Synthetic Datasets. Although semantic data generators already exist in the bibliography, none of them deals with the blank node connectivity issues. Therefore we designed and developed a synthetic generator over the UBA (Univ-Bench Artificial data generator) [5] that can generate datasets with the desired bnode structures. Each dataset corresponds to an RDF graph G. Let $Nodes$ be the set of all nodes in the graph, B be the set of bnodes ($B \subseteq Nodes$), and $conn(o)$ be the nodes of G that are directly connected with a node $o \in Nodes$. We define $b_{density}$ as $b_{density} = \text{avg}_{b \in B} \frac{|conn(b) \cap B|}{|conn(b)|}$. Note that if there are no directly connected bnodes then $b_{density} = 0$. The extended generator can create datasets with the desired $b_{density}$ and the desired maximum length of paths that consist of edges that connect bnodes (we denote by b_{len} their average). Using the synthetic generator, we created a sequence of 9 pairs of KBs (each pair has two subsequent versions of a KB). For instance, the first KB is K_0 and its pair is K'_0. Each time we compare the subsequent versions of a pair with respect to mapping time and yielded delta size. From now on we express the delta size as a percentage of the number of triples of the KB, i.e. as $\frac{|\Delta_e(K,K')|}{\frac{|K|+|K'|}{2}}$. Table 4 shows the blank node properties of each pair of KBs, its optimal delta size over its subsequent version (known by construction) and its variation over the next pair of KBs (we call *b_Neighborhood* every subgraph having as nodes only bnodes, and we call *b_named* triple every triple that contains one bnode). With D_a we denote the average number of direct edges of the bnodes (i.e. average number of triples to which a bnode participates).

Figure 5(left) gives the delta reduction potential of each algorithm in logarithmic scale. Without bnode mapping the delta size ranges from 95% (for the second pair of KBs) to 143% (for the ninth pair of KBs). Instead for Alg_{Hung} it ranges from 0.47% to 10.67% and for Alg_{Sign} it ranges from 1% to 11.5%. Notice that Alg_{Sign} does not reduce the delta to the optimal for any pair of datasets,

Table 4. Blank node Features of the synthetic dataset

K	$\|triples\|$	$\|B\|$	D_a	$b_{density}$	b_{len}	Optimal delta size	Variation
K_{0a}	5,846	240	13.4	0	0	1%	No connected blank nodes
K_{1a}	5,025	240	10.5	0.1	1	0.5%	$b\_Neighborhoods$ of 2 bnodes, reduced b_named triples
K_{2a}	2,381	240	7	0.15	1	1.5%	Reduced b_named triples
K_{3a}	1,628	240	5	0.2	1	1.5%	Reduced b_named triples
K_{4a}	1,636	240	5	0.2	1.15	1%	$b\_Neighborhoods$ of up to 8 bnodes
K_{5a}	1,399	240	4	0.25	1.15	1.7%	Reduced b_named triples
K_{6a}	919	240	3	0.32	1.15	3.2%	$b\_Neighborhoods$ of up to 15 bnodes, reduced b_named triples
K_{7a}	909	240	3.25	0.4	1.35	2.7%	Connect $b\_Neighborhoods$, reduced b_named triples
K_{8a}	1,001	240	3.94	0.5	21.5	2.5%	Connect $b\_Neighborhoods$

while Alg_{Hung} achieves the optimal delta for most of the pairs.
Figure 5 (right) shows the delta reduction potential for the same pairs with the difference that the two bnode lists are not scanned in the original order (as in the left figure), but the second list is reversed. We notice that as the areas of directly connected bnodes become bigger (after the sixth pair of datasets), we get different (here higher) deltas. In such areas the direct neighborhoods lose their discrimination ability and thus the delta reduction potential becomes more unstable, increasing the probability to get a bigger delta.

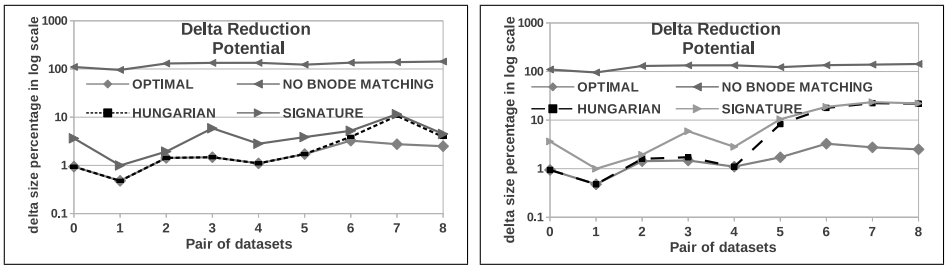

Fig. 5. Delta Reduction over the synthetic datasets

If we use the optimal delta as baseline, and compute the percentage $\frac{|\Delta_x|-|\Delta_{opt}|}{|\Delta_{opt}|}$, in the first diagram this percentage for Alg_{Hung} falls in [0, 2.88], while the Alg_{Sign}'s percentage falls in [0.4,3.2] (in the second diagram they fall in [0,8] and [0.4,8] resp.).

Figure 6 (left) shows the mapping times of each algorithm in logarithmic scale. Alg_{Sign} gives two orders of magnitude lower mapping times.

Equivalences. Regarding equivalent KBs, if there are no directly connected bnodes then Alg_{Hung} detects them at polynomial time (recall Th. 4). To investigate what happens if there are directly connected bnodes we compared the pairs (Kia, Kia) for i=0 to 8 of the synthetic KBs. In case of similarly ordered bnode

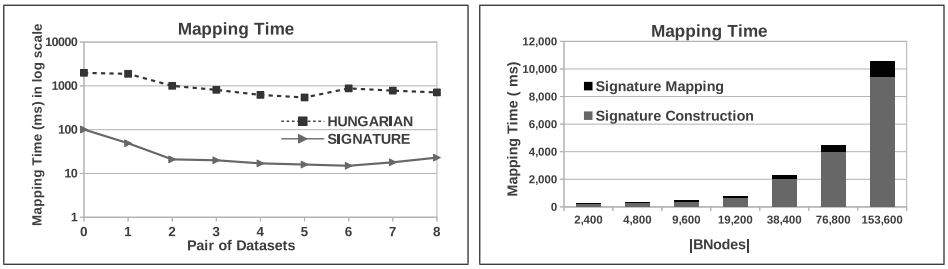

Fig. 6. Mapping times over the synthetic datasets

lists both Alg_{Hung} and Alg_{Sign} detected equivalences for all the KBs, while for reverse scanned bnodes lists they detected 5 of the 9 equivalences. They did not detect equivalences for the KBs with $b_{density} \geq 0.25$.

Bigger Datasets. To investigate the efficiency of Alg_{Sign} in bigger datasets, we created 7 pairs of KBs: the first pair contains 23,827 triples and 2,400 bnodes, the second pair has the double number of triples and bnodes, and so on, until reaching the last pair containing 153,600 bnodes. From Fig. 6 (right) we can see that the mapping time for Alg_{Sign} was only 10.5 seconds for the seventh pair of KBs (153,600 bnodes). Alg_{Hung} could not be applied even to the third pair of KBs due to its high (quadratic) requirements in main memory space.
The results are summarized in the concluding section.

Measuring the approximation.

The upper bound of the reduction of the delta size that can be achieved with bnode matching is the min number of bnodes of the two KBs multiplied by their average degree. Experimentally we have investigated whether $b_{density}$ (which is zero if there are no directly connected bnodes, and equal to 1 if all nodes are bnodes as in the proof of Th. 3), is related with the deviation from the optimal delta $d_x = \frac{|\Delta_x| - |\Delta_{opt}|}{|\Delta_{opt}|+1}$. Results over equivalent and non-equivalent KBs are shown at Figure 7. Both algorithms give a much smaller deviation from optimal than without bnode matching (its d_x ranges [47,114]). We also observe that keeping the original order of the bnodes is beneficial for both algorithms. For the non equivalent KBs the Alg_{Hung} gives always equal or (mostly) smaller delta than the Alg_{Sign}, while for the equivalent both algorithms give exactly the same deviation.

6 Discussing Semantics and Inference Rules

Apart from the explicitly specified triples of a KB, other triples can be inferred based on the RDF/S semantics [6], or other custom inference rules. In some cases one may want to decide whether two KBs are equivalent or to compute their delta with respect to a particular set of rules. In such scenarios, equivalence can be based again on the Def. 1 and the edit distance over nodes on the Def

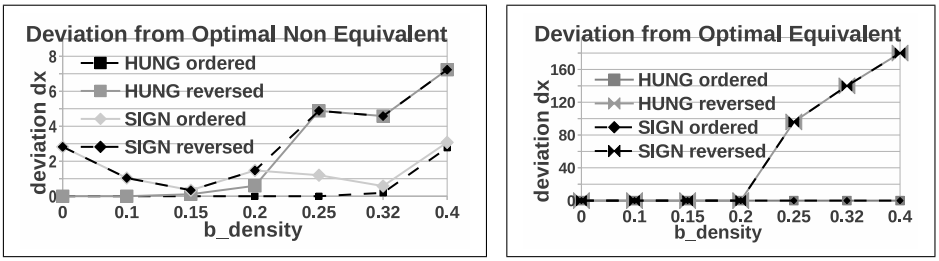

Fig. 7. d_x over non equivalent (left) and equivalent (right) KBs

2 with the only difference that the graphs should be *completed* according to the inferred triples. It follows that if the semantics is based on a set of inference rules yielding a finite closure, then the graph is finite and thus our method can be applied. Some semantics offering finite closures are RDF/S semantics, Minimal RDFS semantics [11], ter Horst's pD* semantics and OWL 2 RL, or even application-specific like [18].

It is worth mentioning, that the optimal bnode mapping over the *complete* graphs may be different from the optimal mapping when considering the *explicit* graphs. In the example of Figure 8, where fat arrows denote `rdfs:subClassOf` relationships and dotted arrows `rdf:type` relationships, the bijection with the minimum cost over the explicit graphs (left) is {(_:1,_:4),(_:2,_:3)}, while at the completed graphs (right) the bijection with the minimum cost is {(_:1,_:3),(_:2,_:4)}

Furthermore, for checking equivalence (at the presence of bnodes) or computing deltas, one could use the *reduced graphs* in case they are unique (note that the *reduction* of a K_a, is the smallest in size K_b that is equivalent to K_a, i.e. K_a and K_b have the same closure).

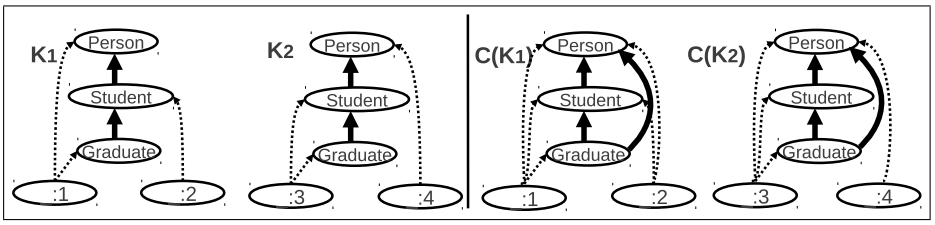

Fig. 8. Comparing the explicit versus the complete graphs of two KBs

7 Related Work

Jena [3] provides a method for deciding whether two KBs that contain bnodes are equivalent (assuming Def. 1) and the adopted algorithm is GI-Complete. PromptDiff [16] and Ontoview [8] employ heuristic matchers to decide whether two bnodes from different KBs match or not, while CWM [1] is able to match two blank nodes only if they have functional term labels. Semversion [20] creates and assigns unique identifiers to bnodes so that to be able to identify the matching

bnodes across versions. However, this is possible only if all versions have been derived from the same system. RDFSync [19] aims at fast synchronization, i.e. at reducing the parts of the KBs that have to be compared, and no effort is dedicated for finding a bnode mapping for reducing the delta size. [13] introduced a blank node mapping with $O(n^2)$ complexity aiming at merging sets of RDF triples (RDF molecules). However, this mapping presupposes that bnodes are parts of uniquely identified triples. This mapping method is not applicable in the general case and cannot be used for delta reduction. To the best of our knowledge our work is the first one that attempts to find a bnode mapping that reduces the size of deltas between KBs (that are not equivalent). Although there are several works for constructing RDF/S *mappings* (e.g. see [14]), they are not directly related since they map the *named* entities of the two KBs, and thus they take into account lexical similarities, something that is not possible with bnodes.

8 Concluding Remarks

In this paper we showed how we can exploit bnode anonymity to reduce the delta size when comparing RDF/S KBs. We proved that finding the optimal mapping between the bnodes of two KBs, i.e. the one that returns the smallest in size delta regarding the *unnamed* part of these KBs, is NP-Hard in the general case, and polynomial in case there are not directly connected bnodes. To cope with the general case we presented polynomial algorithms returning approximate solutions.

In real datasets with no directly connected bnodes Alg_{Sign} was two orders of magnitude faster than Alg_{Hung} (less than one second for KBs with 6,390 bnodes), but yielded up to 0.34 times (or 34%) bigger deltas than Alg_{Hung}, i.e. than the optimal mapping. Alg_{Hung} also identified all equivalent KBs.
For checking the behavior of the algorithms in KBs with directly connected bnodes, we created synthetic datasets, over which we compared Alg_{Sign} and the Alg_{Hung} *approximation* algorithm. Alg_{Hung} yielded from 0 to 3 times smaller deltas than Alg_{Sign}, but the latter was from 18 to 57 times faster. Alg_{Sign} requires only 10.5 seconds to match 153,600 bnodes.

This is the first work on this topic. Several issues are interesting for further research. For instance, it is worth investigating other special cases where the optimal mapping can be found polynomially (e.g. directly connected bnodes that form graphs of bounded tree width). Another direction is to comparatively evaluate various (probabilistic) signature construction methods and greedy approximation algorithms.

Software and datasets are available to download and use from:
`http://www.ics.forth.gr/isl/BNodeDelta`.

Acknowledgement Many thanks to the anonymous reviewers for their comments which helped us to improve the paper, as well to the members of FORTH-ICS-ISL. This work was partly supported by the NoE *APARSEN* (Alliance Permanent Access to the Records of Science in Europe, FP7, Proj. No 269977, 2011-2014), and the FP7 Research Infrastructures projects *SCIDIP-ES* (SCIence Data

Infrastructure for Preservation - Earth Science, 2011, 2014), and *iMarine* (FP7 Research Infrastructures, 2011-2014).

References

1. T. Berners-Lee and D. Connoly. "Delta: An Ontology for the Distribution of Differences Between RDF Graphs", 2004. http://www.w3.org/DesignIssues/Diff.
2. François Bourgeois and Jean-Claude Lassalle. An extension of the Munkres algorithm for the assignment problem to rectangular matrices. *Commun. ACM*, 1971.
3. J. J. Carroll. "Matching RDF graphs". In *Procs of the ISWC'02*, 2002.
4. R. Cloran and B. Irwin. "Transmitting RDF graph deltas for a Cheaper Semantic Web". 2005.
5. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for OWL knowledge base systems. In *Selected Papers from the Intern. Semantic Web Conf. ISWC, 2004.*
6. P. Hayes. "RDF Semantics, W3C Recommendation", 2004.
7. J. Heflin, J. Hendler, and S. Luke. "Coping with Changing Ontologies in a Distributed Environment". In *AAAI-99 Workshop on Ontology Management*, 1999.
8. M. Klein, D. Fensel, A. Kiryakov, and D. Ognyanov. "Ontology versioning and change detection on the web". In *Procs of EKAW'02*, 2002.
9. G. Klyne and J. J. Carroll. Resource Description Framework (RDF):Concepts and Abstract Syntax, 2004.
10. A. Mallea, M. Arenas, A. Hogan, and A. Polleres. On blank nodes. In *Procs of the 10th Intern. Semantic Web Conference (ISWC 2011)*. Springer, October 2011.
11. Sergio Muñoz, Jorge Pérez, and Claudio Gutierrez. Simple and Efficient Minimal RDFS. *Web Semantics*, 2009.
12. J. Munkres. Algorithms for the assignment and transportation problems. *J-SIAM*, 5(1), 1957.
13. A. Newman, YF Li, and J. Hunter. A scale-out rdf molecule store for improved co-identification, querying and inferencing. In *Intern. Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS)*, 2008.
14. J. Noessner, M. Niepert, C. Meilicke, and H. Stuckenschmidt. Leveraging terminological structure for object reconciliation. *Procs of ESWC'10*, 2010.
15. N. F. Noy, S. Kunnatur, M. Klein, and M. A. Musen. "Tracking Changes During Ontology Evolution". In *Procs of ISWC'04*, 2004.
16. N. F. Noy and M. A. Musen. "PromptDiff: A Fixed-point Algorithm for Comparing Ontology Versions". In *Procs of AAAI-02*, 2002.
17. B. Schandl. Replication and versioning of partial rdf graphs. ESWC'10, 2010.
18. C. Strubulis, Y. Tzitzikas, M. Doerr, and G. Flouris. Evolution of Workflow Provenance Information in the Presence of Custom Inference Rules. In *3rd Intern. Workshop on the role of Semantic Web in Provenance Management (SWPM'12), co-located with ESWC'12, Heraklion, Crete*, 2012.
19. G. Tummarello, C. Morbidoni, R. Bachmann-Gmur, and O. Erling. RDFSync: efficient remote synchronization of RDF models. In *(ISWC-07)*, 2007.
20. M. Volkel, W. Winkler, Y. Sure, S. Ryszard Kruk, and M. Synak. "SemVersion: A Versioning System for RDF and Ontologies". In *Procs of ESWC'05.*, 2005.
21. D. Zeginis, Y. Tzitzikas, and V. Christophides. "On the Foundations of Computing Deltas Between RDF Models". In *Procs of ISWC-07*, 2007.
22. D. Zeginis, Y. Tzitzikas, and V. Christophides. "On Computing Deltas of RDF/S Knowledge Bases". *ACM Transactions on the Web (TWEB)*, 2011.