Monte-Mor and Cunha *Journal of the Brazilian Computer Society* (2015) 21:3
DOI 10.1186/s13173-015-0022-3

Journal of the Brazilian Computer Society
a SpringerOpen Journal

RESEARCH Open Access

# ApproxMap - a method for mapping blank nodes in RDF datasets

Juliano de Almeida Monte-Mor[1,2*] and Adilson Marques da Cunha[2]

**Abstract**

**Background:** Versioning has proven to be essential in areas like software development or data and knowledge management. For systems or applications making use of documents formatted according to the Resource Description Framework (RDF) standard, it is difficult to calculate the difference between two versions, owing to the presence of blank nodes, also known as bnodes in RDF graphs. These are anonymous nodes that can assume different identifiers between versions. In this case, the challenge lies in finding a mapping between the sets of blank nodes in the two versions while minimizing the operations needed to convert one version into another.

**Methods:** Within this context, we propose an algorithm, named *ApproxMap*, for mapping bnodes based on extended concepts of rough set theory, which provides a way to measure the proximity of bnodes and map them with closer approximations. Our heuristic method considers various strategies for reducing both the number of comparisons between blank nodes and the delta between the compared versions. The proposed algorithm has a worst-case time complexity of $O(n^2)$.

**Results:** *ApproxMap* showed satisfactory performance in our groups of experiments, as the algorithm that obtained solutions closest to the optimal values. This algorithm succeeded in finding the optimal delta size in 59% of the tests involving optimal values. *ApproxMap* achieved a delta size smaller than or equal to those of existing algorithms in at least 95% of the tested cases.

**Conclusions:** The results show that the proposed algorithm can be successfully applied to versioning RDF documents, such as that produced by software processes with iterative and incremental development. We recommend applying *ApproxMap* in various situations, particularly those involving similar versions and directly connected bnodes.

**Keywords:** Blank nodes; Resource Description Framework (RDF); Mapping; Rough set theory

## Background

In areas such as software engineering, databases, and Web publishing, methods for versioning have already been developed and successfully applied. These methods must be able to calculate the differences (i.e., deltas) between versions to provide efficient storage of subsequent versions.

Particularly in software engineering, versioning algorithms are usually based on a comparison of text lines. However, these methods are not suitable to control versions of structured or semi-structured documents. In this article, we focus specifically on the version control of documents following a Semantic Web standard, the Resource Description Framework (RDF) [1]. We have applied Semantic Web technologies in the software configuration management (SCM) domain [2].

RDF defines a basic data model for writing simple statements about Web objects or resources. It allows the definition of sentences through 'subject-predicate-object' triples; that is, a resource, a property, and a value (which can be a literal or a resource). An RDF triple, like a graph's edge, provides a binary relationship (predicate) that relates a subject to an object. Thus, an RDF document or dataset can be represented by a directed graph [3].

*Correspondence: jmontemor@unifei.edu.br
[1]Federal University of Itajuba - UNIFEI, Campus of Itabira, Rua Irma Ivone Drumond, 200, Distrito Industrial II, Itabira, MG 35903-087, Brazil
[2]Brazilian Aeronautics Institute of Technology - ITA, Praca Marechal Eduardo Gomes, 50, Vila das Acacias, Sao Jose dos Campos, SP 12228-900, Brazil

The conventional line-oriented mechanisms in software engineering are insufficient in the Semantic Web context because their deltas are based on unique serializations, which do not occur naturally in RDF datasets [4]. These bases usually consist of unordered collections of affirmations about resources; however, even when a standard serialization order is imposed (e.g., by sorting), existing comparison tools fail to consider knowledge inferred from schemas associated with RDF datasets [5].

Thus, to obtain the delta between two versions of an RDF dataset, we need to map the nodes in the graphs representing these versions. However, the main problem encountered during calculation of the delta concerns the existence of anonymous nodes (i.e., blank nodes or bnodes) in the RDF graphs. Bnodes represent resources that are not identified by a uniform resource identifier (URI) or literals. In this case, the mapping between bnodes contained in different graph versions directly influences the size of the deltas.

As the scope of identifying bnodes is only local, it is a challenge to find a mapping between bnodes in two versions resulting in the smallest possible delta. Tzitzikas et al. [6] showed that the problem of finding the optimal mapping is NP-hard in the general case and polynomial in the case where bnodes are not directly connected. To illustrate this problem, consider two versions of a dataset as shown in the example proposed by Tzitzikas et al. in Figure 1.

First, we can easily map bnodes '_:3', '_:4', and '_:5' to bnodes '_:8', '_:10', and '_:9', respectively. Then, by mapping bnode pairs '_:1' and '_:6' and '_:2' and '_:7', which seems to be a natural choice, we obtain a delta consisting of four triples. In other words, transforming the first graph into the second requires removing triples '<_:1, friend, _:4>' and '<_:2, friend, _:5>' and adding triples '<_:1, friend, _:5>' and '<_:2, friend, _:4>'. However, if we were to map bnode '_:1' to '_:7' and '_:2' to '_:6', we would have a delta consisting of two triples; that is, triple '<_:1, brother, _:3>' must be removed and triple '<_:2, brother, _:3>' added. The latter mapping is better, owing to the smaller delta size. In the case of directly connected bnodes, we believe that a mapping based on a bottom-up strategy, where nodes in the lower levels are mapped before those in the upper levels, can help reduce the delta size.

During bnode mapping, we need to address inaccuracies between the modified bnodes. To facilitate the handling of this imprecision, we chose to extend some concepts of rough set theory (RST) [7]. RST has already been successfully applied in several areas like artificial intelligence and cognitive sciences. Nicoletti et al. [8] presented the following application examples: creation of machine learning methods, knowledge representation, inductive reasoning, data mining, processing of imperfect or incomplete information, pattern recognition, and discovery of knowledge in databases.

In this context, our approach proposes a heuristic method for mapping blank nodes based on RST. This theory serves as the conceptual basis for the definition of metrics to assist in the choice of bnode pairs, providing the necessary support to map a bnode to the candidate with the closest approximation. Our main objective is to create an algorithm that can be successfully applied in software project versioning.

The remainder of this article is organized as follows: 'Related work' subsection gives an overview of existing work on calculating deltas and mapping bnodes. In 'Problem description' subsection, we formally describe the problem addressed in this work, while 'Rough set theory' discusses some basic concepts of RST. In 'Blank nodes as rough sets', we define a bnode representation model using rough sets, which is necessary for specifying the proposed mapping algorithm in 'The ApproxMap method' section. 'Results and discussion' discusses some

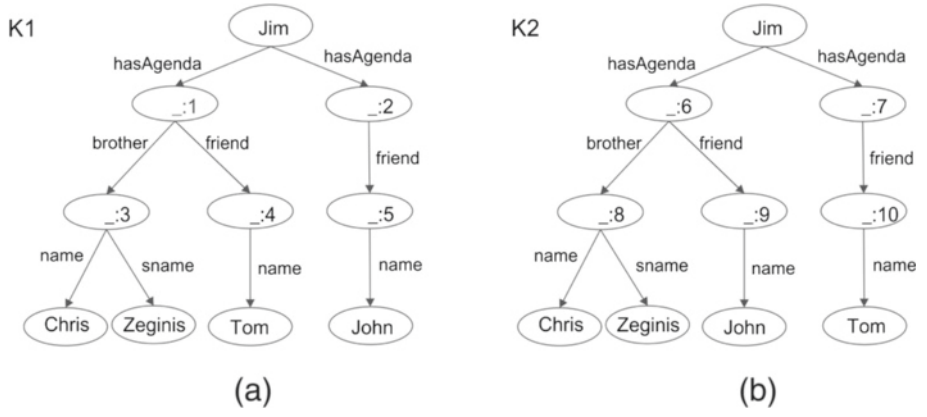


**Figure 1** Versions $K_1$ **(a)** and $K_2$ **(b)** of the same dataset [6].

experimental results, while 'Conclusions' presents our conclusions and recommendations.

**Related work**

Particularly in the software engineering domain, relatively little effort has been made to develop methods for obtaining a better blank node mapping between two versions, by reducing their delta size. Next, we briefly describe some studies on RDF dataset versioning, explaining how they handle blank nodes.

Berners-Lee and Connolly [4] discussed comparing RDF graphs and updating a graph from a calculated set of differences. They emphasized that the order and identification of bnodes can differ arbitrarily with serializations of the same graph. Hence, calculating deltas based on line-oriented approaches is a problem. Computing the differences between two graphs is simple and straightforward if all nodes are named. However, when not all bnodes are named, finding the largest common subgraph becomes an instance of the graph isomorphism problem. The authors further suggested that available solutions for the general isomorphism problem do not appear to be good matches for practical cases. Thus, they proposed an algorithm that produces an RDF difference only for graphs named directly with URIs or indirectly with functional or inverse functional properties. We extend their approach by performing the mapping considering unnamed nodes as well.

Carroll [9] showed that standard algorithms for graph isomorphism can be used to compare RDF graphs. He developed an algorithm considering an iterative vertex classification, used in his RDF toolkit *Jena*, where each anonymous resource is identified based on the statements in which it appears. Thus, bnodes receive identifiers considering their local contexts, which can change between different versions. In our approach, although we do not produce identifiers for bnodes, we also consider the triples in which they appear to classify approximations between bnode pairs.

Noy et al. [10-12] presented an algorithm, called *PromptDiff*, which combines different heuristic matchers to map RDF graphs by comparing structural properties of the ontology versions. New matchers, which may be needed to compare anonymous classes, can easily be added. The authors considered two observations when comparing versions from the same ontology: a large proportion of the frames remain unchanged between versions; and if two frames have the same type and name (or a very similar name), they are almost certainly copies of one another. We follow the first observation, by first mapping equivalent bnodes. We also include some heuristic strategies in the design of our method.

Auer and Herre [13] suggested a framework to support versioning and the evolution of RDF knowledge bases. Their framework is based on atomic changes, including the addition or removal of RDF graphs statements. Atomic changes encompass all statements containing bnodes in a delta, where the graph is atomic if it cannot be split into two nonempty graphs with disjoint blank nodes. In contrast to our approach, because Auer and Herre did not aim to find a mapping between bnodes, there was no commitment to obtain the smallest delta.

Voelkel and Groza [14] showed a versioning approach, called *SemVersion*, which provides structural and semantic versioning for models in RDF/S and OWL. In their approach, bnodes were given unique identifiers in all versions. To identify equal blank nodes across models, they proposed a method for blank node enrichment, where URIs are attached as inverse functional properties to blank nodes. However, this means that blank nodes with different identifiers cannot be mapped, even if they represent the same element in different versions. Moreover, in our approach, we do not add any information to the datasets and do not consider unique identifiers for bnodes in different versions.

Cassidy and Ballantine [15] and Im et al. [16] presented versioning models for RDF repositories. They provided a collaborative annotation facility to develop and share annotations over the Web. Im et al. proposed a version framework for an RDF data model based on relational databases. None of these authors, however, considered blank nodes in their research or defined any method for mapping bnodes, as we do in our approach. These researchers addressed only procedures enabling versioning in RDF repositories.

By considering deltas as sets of change operations, Zeginis et al. [5,17] described various comparison functions, together with the semantics of primitive change operations, and formally analyzed their possible combinations in terms of correctness, minimality, semantic identity, and redundancy properties. Assuming $\mathrm{Add}(t)$ and $\mathrm{Del}(t)$ are, respectively, the straightforward addition and deletion of triple $t$ from set $\mathrm{Triples}(K)$, then, in our approach, we adopt the differential function $\Delta_e$ (where $e$ stands for explicit) for two dataset versions $K$ and $K'$, defined by Zeginis et al. as:

$$\Delta_e(K, K') = \left\{\mathrm{Add}(t) | t \in K' - K\right\} \cup \left\{\mathrm{Del}(t) | t \in K - K'\right\}. \quad (1)$$

Tzitzikas et al. [6] proposed two polynomial time algorithms for mapping bnodes between two knowledge bases. Seeking to reduce the size of the resulting delta, the authors modeled the problem of bnode mapping as an assignment problem and used a Hungarian [18] method, $Alg_{Hung}$, to solve it. This method seeks to find the optimal solution with time complexity $O(n^3)$.

$Alg_{Hung}$ obtains the optimal delta if the considered knowledge bases do not have interconnected bnodes. In the case where the datasets have directly connected bnodes, the authors assume that all neighboring bnodes are equal during mapping. This method cannot be applied to larger knowledge bases owing to its quadratic space requirement in terms of RAM [6].

These authors also proposed a faster signature-based method, called $Alg_{Sign}$, for comparing large knowledge bases with time complexity $O(n \cdot \log n)$. For each bnode, $Alg_{Sign}$ produces a string based on its direct neighborhood as the bnode's signature. Thereafter, the mapping phase compares the generated strings, sorted lexicographically to allow a binary search. The cost of reducing the mapping time is a probable increase in the delta size [6].

Through experiments, Tzitzikas et al. verified that their algorithms obtain deltas with large sizes if the number of directly connected bnodes is high. In this case, once the direct neighborhoods lose their discrimination ability, the delta reduction potential becomes more unstable [6].

Because the number of directly connected bnodes affects the results of both $Alg_{Hung}$ and $Alg_{Sign}$, we proposed a greedy method with a different strategy: neighboring bnodes are treated as different nodes, until they have been mapped in a previous iteration. Our proposal aims to develop a method with lower memory overhead than the $Alg_{Hung}$ algorithm, while reducing the probable increase in delta size when compared with $Alg_{Sign}$.

Research performed before that of Tzitzikas et al. [6] did not seek a mapping that reduces the delta between versions. Tzitzikas et al. were the first to address the bnode mapping problem as an optimization problem, as described in the next section. Accordingly, their work served as the basis for implementing our approach, enabling a comparison between our method and their proposed algorithms.

#### Problem description

In this section, we describe the problem addressed in this article as defined by Tzitzikas et al. [6]. An RDF knowledge base, i.e., an RDF graph, consists of a finite set of RDF triples. Each RDF triple refers to $(s,p,o) \in (W \cup B) \times W \times (W \cup B \cup L)$, where $W$ is an infinite set of URIs, $B$ is an infinite set of blank nodes, and $L$ is an infinite set of literals. Assuming $W_k$, $B_k$, and $L_k$ are sets of URIs, blank nodes, and literals of an RDF $G_k$ graph, respectively, the equivalence between two RDF graphs can be defined as follows:

**Definition 1.** (from [1]) Two RDF graphs $G_1$ and $G_2$ are equivalent if there is a bijection $M$ between the sets of nodes of the two graphs ($N_1$ and $N_2$) such that:

- $M(uri) = uri$, for each $uri \in W_1 \cap N_1$;
- $M(lit) = lit$, for each $lit \in L_1$;
- $M$ maps bnodes to bnodes (i.e., for each $b \in B_1$ it holds that $M(b) \in B_2$); and
- triple $(s,p,o)$ is in $G_1$ if, and only if, triple $(M(s),p,M(o))$ is in $G_2$.

Tzitzikas et al. denoted this equivalence between two graphs $G_1$ and $G_2$ as $G_1 \equiv_M G_2$. Moreover, they also defined the edit distance between two nodes as given in Definition 2. From these two definitions, the equivalence between graphs $G_1$ and $G_2$ can be defined as in Theorem 1.

**Definition 2.** (from [6]) Let $o_1$ and $o_2$ be nodes in $G_1$ and $G_2$, respectively. Suppose a bijection exists between the nodes of these graphs, i.e., function $M : N_1 \longrightarrow N_2$ (obviously $|N_1| = |N_2|$). Then, the edit distance between $o_1$ and $o_2$ over $M$, denoted by $\text{dist}_M(o_1,o_2)$, is the number of additions or deletions of triples required to make the 'direct neighborhoods' of $o_1$ and $o_2$ the same (that is, where $M$-mapped nodes are the same). Formally:

$$\begin{aligned}\text{dist}_M(o_1,o_2) = &\left|\left\{(o_1,p,a) \in G_1 | (o_2,p,M(a)) \notin G_2\right\}\right| \\ &+ \left|\left\{(a,p,o_1) \in G_1 | (M(a),p,o_2) \notin G_2\right\}\right| \\ &+ \left|\left\{(o_2,p,a) \in G_2 | (o_1,p,M^{-1}(a)) \notin G_1\right\}\right| \\ &+ \left|\left\{(a,p,o_2) \in G_2 | (M^{-1}(a),p,o_1) \notin G_1\right\}\right|. \end{aligned} \quad (2)$$

**Theorem 1.** (from [6])

$$G_1 \equiv_M G_2 \Leftrightarrow dist_M(o,M(o)) = 0 \text{ for each } o \in N_1 \quad (3)$$

In the case of versioning, current interest lies in non-equivalent knowledge bases. In this case, it is necessary to find a mapping between bnodes in the two knowledge bases, $B_1$ and $B_2$, that reduces the delta resulting from a comparison thereof.

In this regard, Tzitzikas et al. formulated finding this mapping as an optimization problem: given $n_1 = |B_1|$, $n_2 = |B_2|$, and $n = \min(n_1,n_2)$, the goal is to find the unknown part of bijection $M$. First, $M$ contains the mapping of all URIs and literals of the knowledge bases (according to Definition 1). Assuming that $n = n_1 < n_2$, $\Im$ denotes the set of all possible bijections between $B_1$ and the subset of $B_2$ comprising $n$ elements. Consequently, the set of candidate solutions (i.e., $|\Im|$) is exponential in size. Given the objective of finding a bijection $M \in \Im$ that reduces the size of the delta, they defined the cost of bijection $M$ by Equation 4. From Definition 3, Tzitzikas et al. described the equivalence between two graphs $G_1$ and $G_2$ according to the mapping cost presented in Theorem 2.

$$\text{Cost}(M) = \sum_{b_1 \in B_1} \text{dist}_M(b_1, M(b_1)) \quad (4)$$

**Definition 3.** (from [6]) The best solution (or solutions) is the bijection with the minimal cost. Considering that $arg_M$ returns the set $M \in \Im$ with the minimum cost, we have:

$$M_{\text{sol}} = arg_M \min_{M \in \Im}(\text{Cost}(M)). \tag{5}$$

**Theorem 2.** (from [6])

$$G_1 \equiv_{M_{\text{sol}}} G_2, \textit{ then } \text{Cost}(M_{\text{sol}}) = 0. \tag{6}$$

Therefore, considering the context of this problem described by Tzitzikas et al., we propose a greedy method that seeks to reduce the delta size between two RDF graphs, obtaining an approximate solution to the bijection between the bnodes of these RDF graphs. For this purpose, we define some metrics extending various concepts of RST. In the next section, we present some basic concepts of this theory, which are considered in the design of our algorithm.

### Rough set theory

RST is an extension of set theory, consisting of a mathematical model for uncertainty and imprecision handling, knowledge representation, and rough classification. The main advantage of using RST is that it does not require any preliminary or additional information about the data, such as a probability distribution or membership degree.

In our approach, we adopt RST as the formalism for dealing with imprecision resulting from the comparison of bnode pairs. RST also forms the conceptual basis of defining metrics for measuring the closeness between bnode pairs. Our method aims to map the closest bnode pairs in an attempt to reduce the delta size. Next, we present the main concepts of this theory, extracted from [7,19].

#### *Basic concepts*

Let $U$ be a finite, nonempty, universe set of objects. In set $U$, we can define subsets using the equivalence relation $R$, called the indiscernibility relation. Relation $R$ induces a partition (and consequently, classification) of the objects in $U$. Thus, an approximation space consists of an ordered pair $A = (U, R)$, where given $x, y \in U$, if $xRy$ then $x$ and $y$ are indiscernible in $A$. The equivalence class defined by $x$ is the same as that defined by $y$, i.e., $[x]_R = [y]_R$.

Elementary sets correspond to equivalence classes induced by $R$ in $U$. A partition of $U$ by $R$, denoted by $U/R$, can be viewed as the set $\tilde{R} = U/R = E_1, E_2, \ldots, E_n$, where each $E_i$, with $1 \leq i \leq n$, is an elementary set of $A$. It is assumed that the empty set $\emptyset$ is an elementary set of all approximation spaces $A$. Given an approximation space $A = (U, R)$, let $X \subseteq U$ be any subset of $U$; then, using the following concepts, we can check how well $X$ is represented by the elementary sets of $A$:

- *Lower approximation of X in A* - formed by the union of all elementary sets of $A$ fully contained in $X$, i.e., the largest definable set in $A$ contained in $X$:

$$A_{\text{inf}}(X) = \{x \in U | [x]_R \subseteq X\}. \tag{7}$$

- *Upper approximation of X in A* - formed by the union of all elementary sets of $A$ having a nonempty intersection with $X$, i.e., the smallest definable set in $A$ containing $X$:

$$A_{\text{sup}}(X) = \{x \in U | [x]_R \cap X \neq \emptyset\}. \tag{8}$$

Thus, the lower approximation of $X$ in $A$ contains those elements in $U$ that can definitely be affirmed as belonging to $X$. Furthermore, the upper approximation of $X$ in $A$ covers both those elements that definitely belong to $X$ and those that cannot definitely be excluded from $X$. In many cases, set $X$ may be a finite union of elementary sets, which characterizes $X$ as a definable set in $A$. This implies that $A_{\text{sup}}(X) = A_{\text{inf}}(X) = X$. Besides, based on a rough classification of set $X \subseteq U$, we can identify the following regions in approximation space $A = (U, R)$:

- *Positive region of X in A* - formed by the union of all elementary sets of $U$ fully contained in $X$:

$$\text{pos}\,(X) = A_{\text{inf}}\,(X). \tag{9}$$

- *Negative region of X in A* - formed by the elementary sets of $U$ that have no elements in $X$:

$$\text{neg}\,(X) = U - A_{\text{sup}}(X). \tag{10}$$

- *Doubtful region of X in A* - also called the boundary of $X$, formed by the elementary sets of $U$ that belong to the upper approximation, but do not belong to the lower approximation. The membership of an element of this region to set $X$ is uncertain, based only on the equivalence classes of $A$:

$$\text{duv}\,(X) = A_{\text{sup}}(X) - A_{\text{inf}}\,(X). \tag{11}$$

The positive region has all elements of $U$ that definitely belong to $X$. The negative region comprises all elements that definitely do not belong to $X$. Finally, the doubtful region includes those elements of $U$ whose membership of $X$ cannot definitely be determined. Figure 2 illustrates the main concepts of RST.

#### *Some RST measures*

RST provides several measures (e.g., accuracy and a discriminant index) for checking how well a set $X \in U$ can be represented in approximation space $A = (U, R)$ [7,8,19,20]. In the design of the proposed mapping method, we consider the following RST metrics:

- *Internal measure of X in A*

$$\varpi_{\text{Ainf}}\,(X) = |A_{\text{inf}}\,(X)| \tag{12}$$

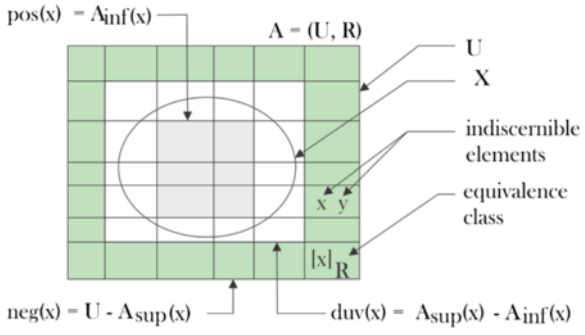


**Figure 2** Regions of *X* in *A*.

- *External measure of X in A*

$$\varpi_{\text{Asup}}(X) = \left|A_{\text{sup}}(X)\right| \tag{13}$$

- *Quality of the lower approximation of X in A*

$$\gamma_{\text{Ainf}}(X) = \frac{\varpi_{\text{Ainf}}(X)}{|U|} = \frac{|A_{\text{inf}}(X)|}{|U|} \tag{14}$$

- *Quality of the upper approximation of X in A*

$$\gamma_{\text{Asup}}(X) = \frac{\varpi_{\text{Asup}}(X)}{|U|} = \frac{\left|A_{\text{sup}}(X)\right|}{|U|} \tag{15}$$

The internal measure is the number of elements in $A$ that definitely belong to $X$, while the external measure indicates the number of elements that could belong to $X$. The metrics for quality of the lower and upper approximations present these measures as percentages of the total number of elements in $A$. In particular, we extended $\gamma_{\text{Ainf}}(X)$ and $\gamma_{\text{Asup}}(X)$ in the design of our mapping algorithm. As a future work, we intend evaluating the adoption of other RST metrics. In the next section, we describe how bnodes can be modeled as approximate sets in an approximation space.

## Methods

We adopted RST in our approach as the basis on which to build a heuristic method to reduce the size of the delta found in the mapping between RDF graphs. To achieve this goal, we must first model the bnodes as sets in an approximation space. The steps required for this transformation are explained below.

### Blank nodes as rough sets

Considering set $B$ containing the bnodes of an RDF graph $G$, Equation 16 defines a subgraph $G_i \subseteq G$ that contains only triples involving a given bnode $b_i \in B$. The negative sign $(-)$ is used to indicate a reverse link in graph $G_i$, i.e., if $b$ is in object '$o$' of triple $(s, p, o)$. Thus, $-W$ is the set consisting of all elements of $W$, preceded by a negative sign.

$$G_i = \left\{(s,p,o) | (s,p,o) \in G \wedge (s = b_i \vee o = b_i)\right\} \tag{16}$$

In addition, *outgoing links* of $b_i$ refer to the links represented by triples in the format $(b_i, p, o) \in G_i$, where $o \neq b_i$. Similarly, we adopt the expression *inbound links* of $b_i$ to refer to triples in the format $(s, p, b_i) \in G_i$, where $s \neq b_i$. Last, we use the symbol '$\sigma$' to denote connections with bnode $b_i$ itself, called $b_i$ recursive links. Thus, to build a set $X_i$ representing bnode $b_i$, we need to transform the triples of $G_i$ using function $S_{b_i} : G_i \rightarrow (W \cup -W) \times (W \cup B \cup L \cup \{'\sigma'\})$:

$$S_{b_i}(s,p,o) = \begin{cases} (p, o), & \text{if } s = b_i \neq o \\ (-p, s), & \text{if } s \neq b_i = o \\ (p, '\sigma'), & \text{if } s = b_i = o. \end{cases} \tag{17}$$

Function $S_{b_i}(s, p, o)$ returns an ordered pair $(l, n)$, where $n$ represents the neighboring node $b_i$ ($s$ or $o$) or '$\sigma$', and $l$ represents the connection or predicate between $b_i$ and $n$. Assuming that $n =$ '$\sigma$', where $S_{b_i}(b_i, p, b_i) = (p, '\sigma')$, the literal '$\sigma$' represents a bnode automatically mapped from the mapping of $b_i$ itself.

In the case of directly connected bnodes, unlike Tzitzikas et al. who considered all bnodes to be the same, our approach considers all unmapped neighboring bnodes to be the same for inbound links and different for outgoing links. Furthermore, we treat 'already mapped' neighbors in the same way as identified nodes (URIs and literals). We can now construct set $X_i$, representing bnode $b_i$, from subgraph $G_i$, corresponding to the image set obtained by applying $S_{b_i}$ to all triples of $G_i$:

$$X_i = S_{b_i}(G_i). \tag{18}$$

Assuming that $B$ corresponds to the bnode set of RDF graph $G$, our method proposes the construction of an approximation space $A = (U, R)$ considering blank nodes $b_i \in B$. Thus, $U$ refers to the set universe obtained from the union of sets $X_i$, representing all considered bnodes $b_i$:

$$U = \bigcup X_i. \tag{19}$$

Besides the set universe, for the construction of approximation space $A = (U, R)$, we also need to define an equivalence relation $R$, to partition the universe into equivalence classes. Given both the set universe $U = \bigcup X_i$ and set intersection $I = \bigcap X_i$, and also two elements $a = (l_a, n_a) \in U$ and $b = (l_b, n_b) \in U$, we define equivalence relation $R$ as:

$$aRb \Leftrightarrow l_a = l_b \wedge ((a, b \in I) \vee (a, b \notin I)). \tag{20}$$

Elements of the same class are indiscernible according to relation $R$. Having defined the approximation space and sets representing bnode pairs in this space, in the next section, we discuss how to extend the RST concepts to provide a measure of the closeness of bnodes.

***Extending the RST concepts***

Given any two approximation sets $X_i$ and $X_j$ in the approximation space $A_{ij} = (U_{ij}, R)$, we observe the following properties for the intersection of their approximations [7]: $A_{\inf}(X_i) \cap A_{\inf}(X_j) = A_{\inf}(X_i \cap X_j)$ and $A_{\sup}(X_i) \cap A_{\sup}(X_j) \supseteq A_{\sup}(X_i \cap X_j)$. For a more accurate analysis of the approximation of $X_i$ and $X_j$ in $A_{ij}$, we can extend the concepts of positive, doubtful, and negative regions, considering the intersections between their approximations:

**Definition 4.** Change regions for $X_i$ and $X_j$ in $A_{ij}$

- *Positive change region* - formed by the union of all elementary sets of $U_{ij}$ contained entirely in both $X_i$ and $X_j$:

$$\text{pos}\left(X_i, X_j\right) = A_{\text{inf}}\left(X_i\right) \cap A_{\text{inf}}\left(X_j\right). \tag{21}$$

- *Negative change region* - formed by elementary sets of $U_{ij}$ that have no elements in $X_i$ or $X_j$:

$$\text{neg}\left(X_i, X_j\right) = U_{ij} - \left(A_{\text{sup}}\left(X_i\right) \cap A_{\text{sup}}\left(X_j\right)\right). \tag{22}$$

- *Doubtful change region* - formed by elementary sets of $U_{ij}$ partially contained in $X_i$ or $X_j$. In this case, $X_i$ or $X_j$, but not both, may integrally contain elementary sets of $U_{ij}$:

$$\text{duv}\left(X_i, X_j\right) = \left(A_{\text{sup}}(X_i) \cap A_{\text{sup}}\left(X_j\right)\right) - \left(A_{\text{inf}}\left(X_i\right) \cap A_{\text{inf}}\left(X_j\right)\right). \tag{23}$$

The positive change region $\text{pos}\left(X_i, X_j\right)$ comprises classes that relate to existing links in both bnodes, with the same neighboring nodes, i.e., these classes contain elements representing equivalent links, considering the mapping between bnodes. Classes contained in the doubtful change region $\text{duv}\left(X_i, X_j\right)$ contain elements representing predicates common to the bnodes, but connected to different neighbors, being considered as similar links. They represent change operations on common predicates of bnodes: rename, extend, or reduce. Finally, the negative change region $\text{neg}\left(X_i, X_j\right)$ consists of classes that are not found in both bnodes. These classes refer to the addition or removal of bnode predicates being considered as independent links.

The change regions may provide a way of measuring the approximation between the two sets representing the bnodes. However, before addressing this issue, we analyze some extreme situations involving these regions to improve the understanding thereof. Initially, considering the case where all elements are in the positive change region, we can rank the bnodes as *equivalent* in $A_{ij}$, because there are no differences between the bnode predicates, i.e., $(b_i \equiv_{A_{ij}} b_j) \Leftrightarrow (A_{\inf}(X_i) \cap A_{\inf}(X_j) = U_{ij})$, where this relationship is denoted by the symbol $\equiv_{A_{ij}}$. Otherwise, if this region is empty, the bnodes have no common connections with the same neighboring nodes (equivalent links), i.e., $A_{\inf}(X_i) \cap A_{\inf}(X_j) = \emptyset$. In this case, analysis of other change regions is necessary.

Regarding the doubtful change region, if all elements meet in this region it means that the bnodes have similar links with different neighboring nodes, i.e., $(A_{\inf}(X_i) \cap A_{\inf}(X_j) = \emptyset) \wedge (A_{\sup}(X_i) \cap A_{\sup}(X_j) = U_{ij})$. If this region is empty, there are no changes in the predicates common to both bnodes, i.e., $(A_{\sup}(X_i) \cap A_{\sup}(X_j)) - (A_{\inf}(X_i) \cap A_{\inf}(X_j)) = \emptyset$. If the positive and/or doubtful regions are not empty and smaller than the universe, we categorize bnodes as *approximated* in $A_{ij}$, represented by the symbol $\approx_{A_{ij}}$, because they have predicates in common, i.e., $(b_i \approx_{A_{ij}} b_j) \Leftrightarrow (\emptyset \neq (A_{\sup}(X_i) \cap A_{\sup}(X_j)) \neq U_{ij})$.

Finally, if all the elements are in the negative change region, we classify the bnodes as *distinct* in $A_{ij}$, represented by $\neq_{A_{ij}}$, because they have independent links, i.e., $(b_i \neq_{A_{ij}} b_j) \Leftrightarrow (A_{\sup}(X_i) \cap A_{\sup}(X_j) = \emptyset)$. On the other hand, if this region is empty, all the connections are common to both bnodes, i.e., $A_{\sup}(X_i) \cap A_{\sup}(X_j) = U_{ij}$.

Therefore, we can evaluate the approximation between bnodes from these change regions. For this purpose, we need to extend the RST measures presented in 'Some RST measures' subsection to measure the approximation between sets $X_i$ and $X_j$ in $A_{ij}$, by considering the intersection of the approximation of these sets:

**Definition 5.** Change measures of $X_i$ and $X_j$ in $A_{ij}$

- *Internal change measure*

$$\varpi_{\text{Ainf}}\left(X_i, X_j\right) = \left|A_{\text{inf}}\left(X_i\right) \cap A_{\text{inf}}\left(X_j\right)\right| \tag{24}$$

- *External change measure*

$$\varpi_{\text{Asup}}\left(X_i, X_j\right) = \left|A_{\text{sup}}\left(X_i\right) \cap A_{\text{sup}}\left(X_j\right)\right| \tag{25}$$

- *Quality of the lower change approximation*

$$\begin{aligned} \gamma_{\text{Ainf}}\left(X_i, X_j\right) &= \frac{\varpi_{\text{Ainf}}\left(X_i, X_j\right)}{|U|} \\ &= \frac{\left|A_{inf}(X_i) \cap A_{inf}(X_j)\right|}{|U|} \end{aligned} \tag{26}$$

- *Quality of the upper change approximation*

$$\begin{aligned} \gamma_{\text{Asup}}\left(X_i, X_j\right) &= \frac{\varpi_{\text{Asup}}\left(X_i, X_j\right)}{|U|} \\ &= \frac{\left|A_{\text{sup}}(X_i) \cap A_{\text{sup}}(X_j)\right|}{|U|} \end{aligned} \tag{27}$$

Based on the measures given in Definition 5, we redefine the approximation between two bnodes $b_i$ and $b_j$ in Definition 6. $\gamma_{\text{Ainf}}(X_i, X_j)$ provides a way of measuring the percentage of identical predicates considering the mapping between $X_i$ and $X_j$, while $\gamma_{\text{Asup}}(X_i, X_j)$ provides a way

of measuring the approximation between the predicates of $X_i$ and $X_j$.

**Definition 6.** Approximation between $b_i$ and $b_j$ in $A_{ij}$

- $(b_i \equiv_{A_{ij}} b_j) \Leftrightarrow (\gamma_{\text{Ainf}}(X_i, X_j) = 1)$;
- $(b_i \approx_{A_{ij}} b_j) \Leftrightarrow (0 < \gamma_{\text{Asup}}(X_i, X_j) < 1)$;
- $(b_i \not\equiv_{A_{ij}} b_j) \Leftrightarrow (\gamma_{\text{Asup}}(X_i, X_j) = 0)$.

#### *Exemplifying the modeling*

To illustrate the construction of sets in an approximation space representing bnodes of RDF graphs, suppose we need to map a blank node modified in two subsequent versions $G_1$ and $G_2$ of a dataset. Figure 3 presents graphs representing the first pair of candidates. Figure 4 shows the positive, negative, and doubtful regions of sets $X_1$ and $X_2$ in the approximation space $A_{12}$, while Figure 5 presents the positive, doubtful, and negative change regions in $A_{12}$.

Now consider another bnode candidate $b_3 \in G_2$ (labeled as '*_:ProductB*'), represented by $X_3$, as shown in Figure 6a; then, Figure 6b presents the change regions for $X_1$ and $X_3$ in $A_{13}$. For this example, we obtain the following values for sets $X_1$, $X_2$, and $X_3$:

- $\varpi_{\text{Ainf}}(X_1, X_2) = 5$;
- $\varpi_{\text{Ainf}}(X_1, X_3) = 3$;
- $\varpi_{\text{Asup}}(X_1, X_2) = 8$;
- $\varpi_{\text{Asup}}(X_1, X_3) = 10$;
- $\gamma_{\text{Ainf}}(X_1, X_2) = 5/10 = 0.5$;
- $\gamma_{\text{Ainf}}(X_1, X_3) = 3/12 = 0.25$;
- $\gamma_{\text{Asup}}(X_1, X_2) = 8/10 = 0.8$;
- $\gamma_{\text{Asup}}(X_1, X_3) = 10/12 \approx 0.83$.

Thus, we have $b_1 \approx_{A_{12}} b_2$ and $b_1 \approx_{A_{13}} b_3$, but as $\gamma_{\text{Ainf}}(X_1, X_2) > \gamma_{\text{Ainf}}(X_1, X_3)$, we prefer the mapping between $b_1$ and $b_2$. We applied metric $\gamma_{\text{Ainf}}(X_i, X_j)$ in the mapping between bnode pairs $b_i$ and $b_j$, with the aim of reducing the delta between the versions. The greater is the value of the lower approximation quality, the higher is the equivalence between the bnode connections. In cases with equal values for $\gamma_{\text{Ainf}}(X_i, X_j)$, we prioritize the pairs providing the greatest value for $\gamma_{\text{Asup}}(X_i, X_j)$, because these are the bnodes with the closest approximations in terms of connections representing the same predicates.

We assume that mapping bnode pairs with higher equivalence or greater approximation between their predicates can reduce the delta size. In the next section, we use the approximation metrics $\gamma_{\text{Ainf}}(X_i, X_j)$ and $\gamma_{\text{Asup}}(X_i, X_j)$ to design the proposed mapping algorithm.

### The ApproxMap method

In this section, we describe the strategies, data structures, and procedures designed to map bnodes in two RDF graphs. We call our mapping algorithm *ApproxMap*, because the project involves an analysis of the approximation between the sets representing the bnodes.

#### *Heuristic strategies*

Our heuristic method considers various strategies for reducing both the number of comparisons between blank nodes and the delta between the compared versions. We adopted the following strategies in the design of our method:

- *Two approximation metrics* - we use metric $\gamma_{\text{Asup}}(X_i, X_j)$ if the candidate pairs have the same $\gamma_{\text{Ainf}}(X_i, X_j)$. A pair with a greater $\gamma_{\text{Asup}}(X_i, X_j)$ has a higher similarity owing to the greater number of common predicates. We consider that mapping pairs with more similar predicates can help in reducing the delta size.
- *Two levels for bnode partitioning* - the first level considers the existing hierarchy between directly connected bnodes, classifying the bnodes into four disjoint sets: roots, leaves, intermediates, and no interconnections. Then, in the second partitioning level, we organize the bnodes according to the number of connections with other nodes, allowing quick access to sets of bnodes with a particular number of links.

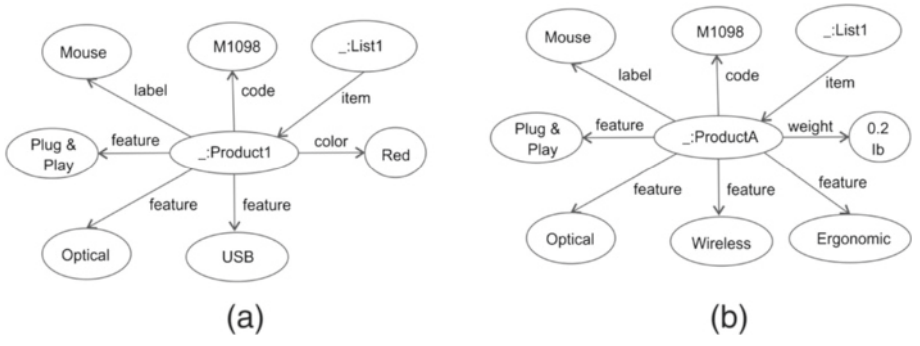


**Figure 3** Simplified graphs of the two versions **(a, b)**.

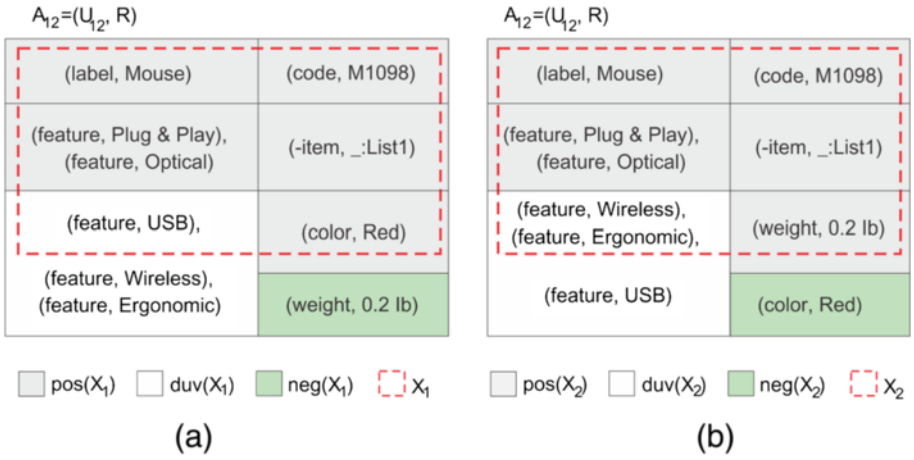


**Figure 4** Regions of the approximation space for $X_1 \subset A_{12}$ **(a)** and $X_2 \subset A_{12}$ **(b)**.

- *Unmapped neighboring bnodes are the same for incoming links but differ for outbound links* - while neighboring bnodes are unmapped, URIs and literals play an important role in distinguishing blank nodes. The strategy adopted by Tzitzikas et al. [6], whereby all neighbors as considered the same, can increase the delta size, if the mapped neighbors differ in the final mapping. Therefore, we aim to mitigate this effect by adopting the strategy described above, which considers the possible impact of different neighbors when computing the delta. With prior mapping of neighboring bnodes, we can find a greater approximation between candidate pairs.
- *Bottom-up approach to map directly connected bnodes* - bnodes in the higher levels are mapped based on prior mappings in the lower levels. We compare each bnode mainly with those in the same hierarchical level, thereby reducing the number of comparisons. Relaxation of the same neighborhood for incoming links is due to this approach.
- *Top-down approximation during bnode mapping* - bnodes are mapped iteratively considering a decreasing approximation in the interval (0.0, 1.0]. We start the mapping of bnodes with the maximum approximation and, in each iteration, we reduce the lower limit for the desired approximation. Using this approach, we are able to reduce the number of comparisons between bnodes if the datasets contain vastly differing numbers of bnode links. This is because we do not need to compare bnode pairs that differ greatly in their numbers of links, thereby preventing an approximation greater than or equal to the desired value.
- *Initial equivalent bnode mapping* - we can reduce the number of comparisons between the remaining bnodes that have not yet been mapped. Moreover, during the mapping of equivalent bnodes, we can also reduce the comparisons by applying filters to select only those bnodes in the same hierarchical level and with the same number of links as the other nodes.

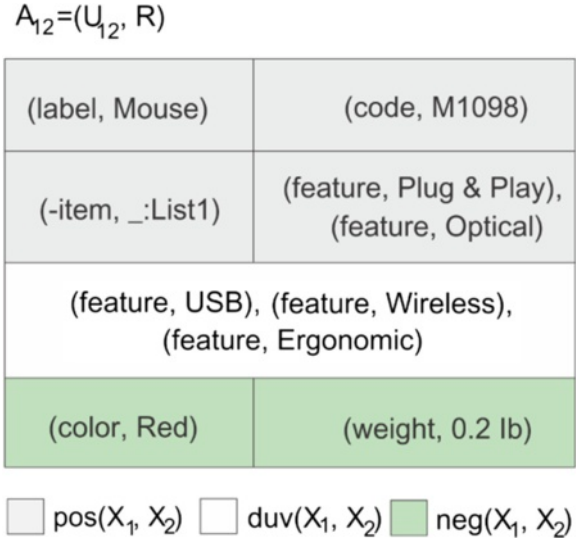


**Figure 5** Change regions in $A_{12}$.

Our heuristic combines all these strategies in an attempt to produce a solution with a reduced delta size during the mapping of blank nodes of two RDF graphs. For this purpose, we use specific data structures, as described in the next section.

### *Data structures*

In the first adopted partitioning level, we store the unmapped bnodes of each graph $G_k$ in the data structure, $TabG_k$, which is partitioned into four disjoint sets:

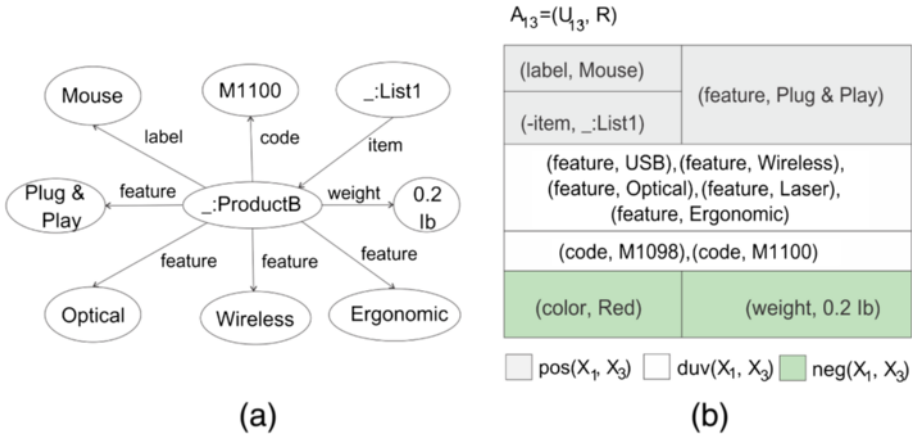


**Figure 6** Simplified graph of the other candidate and the respective change regions in $A_{13}$ **(a, b)**.

roots, leaves, intermediates, or no interconnections. We use the operator '[ ]' to index the partitions of $TabG_k$, where $TabG_k[i]$ denotes partition $i$ of $TabG_k$.

Bnodes without links to other bnodes are placed in $TabG_k[1]$. In the case of directly connected bnodes, the division thereof occurs according to a hierarchical model as shown in Figure 7. $TabG_k[4]$ contains bnodes that are roots in this model; $TabG_k[2]$ stores leaf bnodes; and $TabG_k[3]$ contains bnodes that belong to intermediate layers of this hierarchy, connected to other bnodes by both incoming and outbound links. For simplicity, in Figure 7, we omit the labels of the elements. Moreover, despite the presence of URIs and literals in the figure, the partitioning covers only blank nodes.

Each partition of $TabG_k$ is further partitioned in a second level and indexed by the number of bnode links. This allows us to find bnodes with the same number of predicates quickly, where $TabG_k[i][j]$ returns a reference to the set of bnodes from partition $i$ of $TabG_k$, with $j$ connections with neighboring nodes.

The *ApproxMap* algorithm also makes use of four arrays, with size equal to $|B_k|$, for each graph $G_k$: $alias_k$, $\text{approxInf}_k$, $approxSup_k$, and $M_k$. Considering that $b_i \in B_k$, $alias_k[i]$ stores the bnode currently mapped to $b_i$; $\text{approxInf}_k[i]$ and $approxSup_k[i]$ refer, respectively, to the values of the lower and upper approximations, calculated for $b_i$ and $alias_k[i]$. Similarly, $M_k[i]$ stores the bnode definitely mapped to $b_i$.

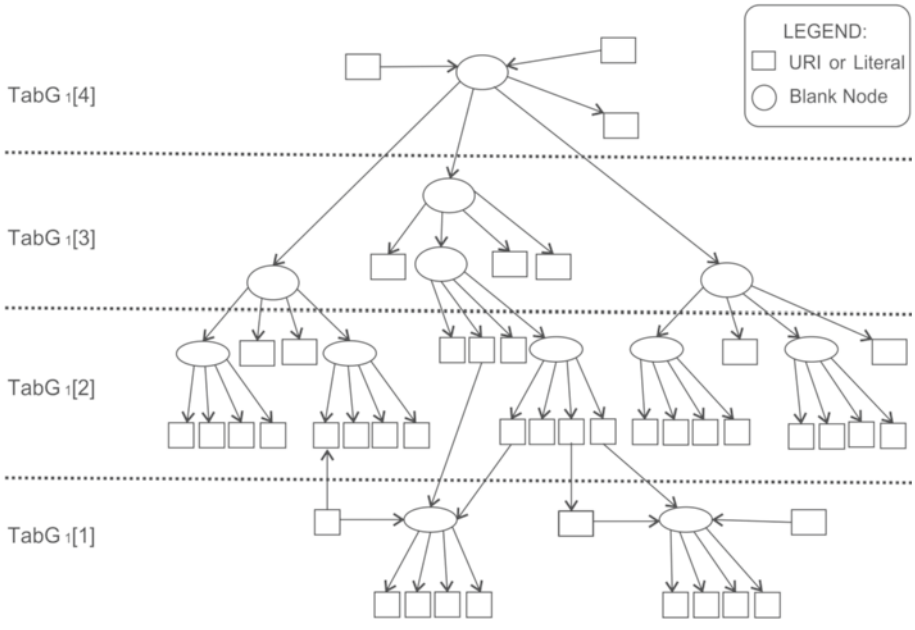


**Figure 7** Example of an RDF graph partitioning.

Before describing the *ApproxMap* method, we need to explain the process of finding bnode pairs with the greatest approximation during the mapping. In the next section, we discuss this process, which uses the data structures mentioned above.

#### *Mapping bnode pairs*

We implemented the mapping of bnodes in two RDF graphs in two phases. In the first phase, as shown in the pseudocode in Figure 8, we look for pairs of unmapped bnodes with the closest approximation. The *FindApproximations* algorithm takes as parameters, indexes $m$ and $n$ referring, respectively, to the desired partitions of $TabG_1$ and $TabG_2$, with $1 \leq m, n \leq 4$, and parameter *approx*, where $0.0 < \text{approx} < 1.0$, which denotes the lower boundary of the current desired approximation.

The algorithm looks for pairs with a value for the quality of lower approximation $\gamma_{\text{Ainf}}(X_i, X_j)$ greater than or equal to the desired value indicated by approx; values below this limit are discarded. Variable $b_m$ stores the current bnode with the closest approximation to $b_i$, while $\text{api}_m$ and $\text{aps}_m$ store, respectively, their lower and upper approximations, calculated by metrics $\gamma_{\text{Ainf}}(X_i, X_j)$ and $\gamma_{\text{Asup}}(X_i, X_j)$.

Considering subgraph $G_i \subseteq G_k$, as defined in Equation 16, $|G_i|$ is the cardinality of $G_i$, i.e., the number of triples or connections of $b_i$. In addition, $\Phi_i$ is the set of possible $p$ values for triples in the form $(s, p, b_i) \in G_i$, and $\Theta_i$ is the set of $p$ values for triples $(b_i, p, o) \in G_i$.

In lines 5 and 6 of the algorithm, we use the values of variables $l_{\text{inf}}$ and $l_{\text{sup}}$ to reduce the comparison space using the top-down approximation approach discussed in the 'Heuristic strategies' subsection. We can only find an approximation greater than or equal to approx in the interval $[l_{\text{inf}}, l_{\text{sup}}]$, considering our second partitioning level. In line 10, a further filtering takes place, whereby

**Algorithm** $FindApproximations(m, n, approx)$

```
1  For each b_i ∈ TabG1[m], do:
2     api_m ← approxInf1[i];
3     aps_m ← approxSup1[i];
4     b_m ← alias1[i];
5     l_inf ← |G_i| × approx;
6     l_sup ← |G_i| / approx;
7     For each l ← l_inf to l_sup, do:
8        Ω ← TabG2[n][l];
9        For each b_j ∈ Ω, do:
10          If (|Φ_i ∩ Φ_j| + |Θ_i ∩ Θ_j| > 0), then:
11             Build X_i, X_j and A_ij(U_ij, R);
12             api ← γ_Ainf(X_i, X_j);
13             aps ← γ_Asup(X_i, X_j);
14             api_2 ← approxInf2[j];
15             aps_2 ← approxSup2[j];
16             If (api > api_m ∧ api > api_2), then:
17                b_m ← b_j;
18                api_m ← api;
19                aps_m ← aps;
20             Else:
21                If (api = api_m ∧ api = api_2) ∧ (aps > aps_m ∧ aps > aps_2), then:
22                   b_m ← b_j;
23                   aps_m ← aps;
24                EndIf.
25             EndIf.
26          EndIf.
27       EndFor.
28    EndFor.
29    If (alias1[i] ≠ b_m), then:
30       alias1[i] ← b_m;
31       approxInf1[i] ← api_m;
32       approxSup1[i] ← aps_m;
33       alias2[m] ← b_i
34       approxInf2[m] ← api_m;
35       approxSup2[m] ← aps_m;
36    EndIf.
37 EndFor.
```

**Figure 8** Pseudocode for *FindApproximations* algorithm.

only bnodes with at least one predicate in common are compared.

After obtaining the lower approximation between $b_i$ and the candidate $b_j$, in line 16, we check whether this new approximation is greater than that previously found. If so, the respective bnodes are marked as candidates for mapping, and any previous pairs are discarded. However, if the new value for $\gamma_{\text{Ainf}}(X_i, X_j)$ is equal to that previously found, we compare the new value of $\gamma_{\text{Asup}}(X_i, X_j)$, as shown in line 22. If this value is greater than the current value, the respective bnodes are also marked as candidates for mapping.

After the first phase, we have pairs of candidates with the greatest approximation for mapping, which is finalized in the second phase. Procedure *MapApproximations*($m$, *approx*), with $1 \leq m \leq 4$, is used to carry out the mapping. Bnodes in $TabG_1[m]$ with an approximation greater than or equal to parameter approx are permanently mapped.

Procedures *FindApproximations* and *MapApproximations* are executed to map similar bnodes. However, we can refine these procedures to filter unmapped bnodes, when looking for equivalent pairs to reduce the search space. Thus, we designed procedure *MapEquivalents*($m$) to map equivalent bnodes in $TabG_1[m]$ and $TabG_2[m]$, where $1 \leq m \leq 4$. This procedure compares only bnodes with exactly the same incoming and outbound predicates. Thus, we permanently map only those bnode pairs with approximations equal to 1.0.

We also developed a procedure to map the remaining bnodes, after termination of the iterations for the adopted top-down approximation strategy. Procedure *MapByOrder*() compares bnodes in the same way as *FindApproximations*. However, the mapping is carried out directly between pairs with the greatest approximation according to the order defined by the partitioning of $TabG_1$, thereby ignoring the possibility of a closer relationship with another bnode pair.

#### *Proposed method*

Finally, we present method *ApproxMap* illustrated in Figure 9, which aims to map the bnodes of two graphs $G_1$ and $G_2$, considering a decreasing approximation in the interval (0.0, 1.0). The mapping occurs between pairs in the same hierarchical level $TabG_1[m]$ and $TabG_2[m]$ considering a fixed step defined by parameter $\eta_1$. To map bnodes of $TabG_1[m]$ and $TabG_2[n]$, with $m \neq n$, we adopted the step defined by $\eta_2$, where $0 < \eta_1 < \eta_2 < 1$. Variable min stores the current desired value (or the lower boundary) for the approximation between bnodes.

The *ApproxMap* starts by mapping equivalent bnodes in $TabG_k[1]$ and $TabG_k[2]$, as shown in lines 1 and 2. During the mapping of $TabG_k[2]$, we consider relaxing the neighboring bnodes for inbound links. This mapping is performed only once, because these bnodes are leaves in the hierarchy and do not depend on previous mappings of other bnodes.

The rest of the algorithm includes a loop, defined between lines 4 and 45, that maps the bnodes using the bottom-up approach discussed in 'Heuristic strategies' subsection, where the mapping of bnodes contained in tables $TabG_k[3]$ and $TabG_k[4]$ depends on previous mappings of bnodes in lower levels of the hierarchy. The

**Algorithm** $ApproxMap(\eta_1, \eta_2)$

```
1  MapEquivalents(1);
2  MapEquivalents(2);
3  approx ← 1;
4  While (approx > 0), do:
5        min ← approx − η2;
6        FindApproximations(1, 4, min);
7        FindApproximations(1, 3, min);
8        FindApproximations(1, 2, min);
9        aux ← approx;
10       While (aux > min), do:
11             aux ← aux − η1;
12             FindApproximations(1, 1, approx);
13             MapApproximations(1, approx);
14       EndWhile.
15       FindApproximations(2, 4, min);
16       FindApproximations(2, 3, min);
17       FindApproximations(2, 1, min);
18       aux ← approx;
19       While (aux > min), do:
20             aux ← aux − η1;
21             FindApproximations(2, 2, approx);
22             MapApproximations(2, approx);
23       EndWhile.
24       MapEquivalents(3);
25       FindApproximations(3, 4, min);
26       FindApproximations(3, 2, min);
27       FindApproximations(3, 1, min);
28       aux ← approx;
29       While (aux > min), do:
30             aux ← aux − η1;
31             FindApproximations(3, 3, approx);
32             MapApproximations(3, approx);
33       EndWhile.
34       MapEquivalents(4);
35       FindApproximations(4, 3, min);
36       FindApproximations(4, 2, min);
37       FindApproximations(4, 1, min);
38       aux ← approx;
39       While (aux > min), do:
40             aux ← aux − η1;
41             FindApproximations(4, 4, approx);
42             MapApproximations(4, approx);
43       EndWhile.
44       approx ← min;
45 EndWhile;
46 MapByOrder( ).
```

**Figure 9** Pseudocode for *ApproxMap* algorithm.

algorithm aims to map $TabG_1[1]$ (lines 6 to 14), $TabG_1[2]$ (lines 15 to 23), $TabG_1[3]$ (lines 24 to 33), and $TabG_1[4]$ (lines 34 to 43) in order.

Thus, for each iteration of the outer loop, the value of *min* is decremented according to step $\eta_2$, as expressed in line 5. This value defines the minimum approximation required to map the bnodes in each partition of $TabG_1$ to the other different partitions in $TabG_2$. In the case of the same partitions, the mapping occurs in the inner loops, taking into account step $\eta_1$, so that the current approximation is decremented in each iteration (lines 11, 20, 30, and 40), until it reaches the limit set in *min*. Just prior to termination of the algorithm, in line 46, the remaining bnodes are mapped after $1/\eta_1$ iterations.

We compared the bnodes of $TabG_1[m]$ $1/\eta_1$ times with the ones in $TabG_2[m]$, and a minor number of $1/\eta_2$ times with those in $TabG_2[n]$, where $m \neq n$. Therefore, during the search for bnode pairs with greater approximations, the outer loop provides the mapping of bnodes that change partitions between versions, while the inner loop provides the mapping of bnodes that remain in the same hierarchical partition for all versions.

#### *Method analysis*

The proposed method models bnodes as approximate sets, based on their classification as equivalent, similar, or distinct predicates in terms of their connections with other nodes. This organization by approximation classes allows the definition of metrics to measure the approximation between bnodes.

Considering the introductory example in Figure 1, algorithms $Alg_{Hung}$ and $Alg_{Sign}$ obtain a mapping resulting in a delta with size 4. Tzitzikas et al. focused on the mapping between pairs (_:1, _:6) and (_:2, _:7) because they considered connected bnodes to be the same, where $\text{dist}_h(1,6) = 0$ and $\text{dist}_h(1,7) = 1$. We emphasize the adoption of both bottom-up and different neighbor strategies in *ApproxMap* while mapping directly connected bnodes. The first iteration of *ApproxMap* results in the mapping of bnode pairs (_:3, _:8), (_:4, _:10), and (_:5, _:9), which have an approximation equal to 1.0. From this initial mapping, our method can map pairs (_:1, _:7) and (_:2, _:6), because $\gamma_{\text{Ainf}}(X_1, X_6) = 0.50$, $\gamma_{\text{Ainf}}(X_1, X_7) = 0.67$, $\gamma_{\text{Ainf}}(X_2, X_6) = 0.67$, and $\gamma_{\text{Ainf}}(X_2, X_7) = 0.34$. The mapping obtained by our method results in a smaller delta size of two triples.

On the other hand, during the *ApproxMap* method design, we assume that reducing the delta for individual bnode pairs also results in a reduction in the global delta size. However, this assumption does not produce the optimal delta in some situations, as illustrated in Figure 10. In this example, we assume that sets $X_1$, $X_2$, $X_3$, and $X_4$ represent, respectively, bnodes '*_:Product1*', '*_:Product2*', '*_:Product3*', and '*_:Product4*'. Thus, we obtain the following approximation measurement: $\gamma_{\text{Ainf}}(X_1, X_3) = 0.50$; $\gamma_{\text{Ainf}}(X_1, X_4) = 0.33$; $\gamma_{\text{Ainf}}(X_2, X_3) = 0.40$; and $\gamma_{\text{Ainf}}(X_2, X_4) = 0.14$.

First, *ApproxMap* maps bnodes '*_:Product1*' and '*_:Product3*', corresponding to the pair with the closest approximation. The closest approximations of both '*_:Product1*' and '*_:Product2*' are to bnode '*_:Product3*'.

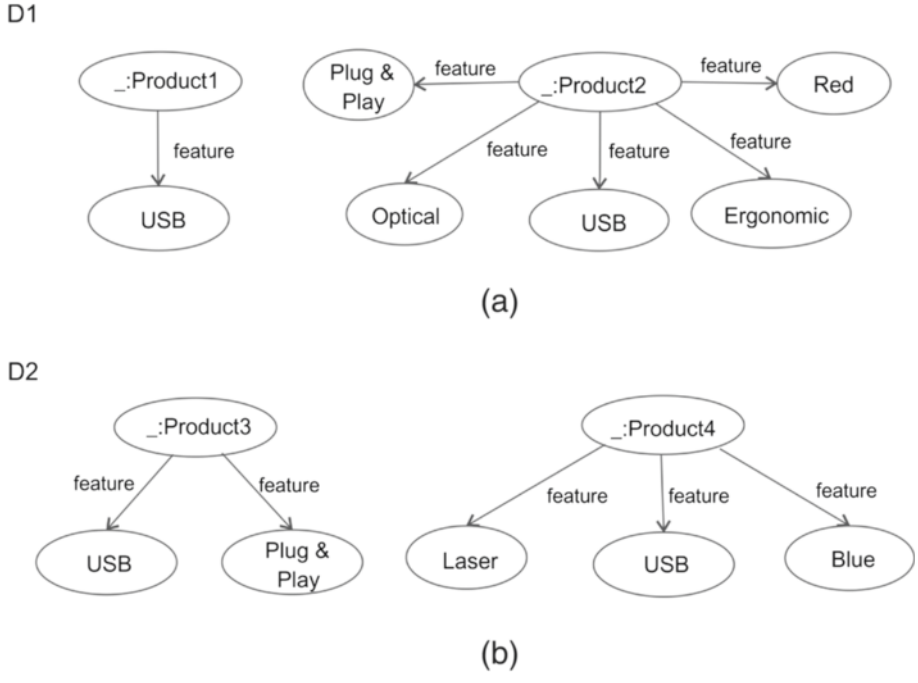


**Figure 10** Sample datasets with small diversity between bnodes **(a,b)**.

However, this mapping represents the lowest cost of transforming some bnode in the first version into '*_:Product3*'. We can change '*_:Product1*' to '*_:Product3*' by including only a single triple. However, we would need to include an additional three triples to transform '*_:Product2*' into '*_:Product3*'.

Therefore, *ApproxMap* also maps the remaining bnodes '*_:Product2*' and '*_:Product4*', resulting in a global delta containing seven triples. However, if we had initially mapped '*_:Product1*' to '*_:Product4*', the resulting delta would have size 5, as is the case using the *Hungarian* algorithm. This occurs because our hypothesis considers only a reduction in delta between individual pairs and not an assessment of the impact of this reduction in terms of the global delta size. Owing to the mapping of remaining bnodes, considering only unmapped bnodes pairs, the *ApproxMap* does not test all mapping possibilities, which can result in obtaining a local optimum.

Moreover, in *ApproxMap*, the mapping occurs in the order defined by the adopted partitioning. We also used some ordered structures during algorithm implementation, optimizing the comparisons between bnodes. The additional cost of insertion is already known for these structures, although this is beyond the scope of this article. This adopted order can affect the delta size, mainly, considering procedure *MapByOrder*. As before, this may occur because our method does not test all mapping possibilities.

We emphasize that great diversity among the bnode predicates in the same version is beneficial for method *ApproxMap*. For a better analysis, let us consider the approximation space $A_{1-4}$, illustrated in Figure 11, and which we constructed from the union of all sets representing bnodes in the two versions in Figure 10. For simplicity, we omit the negative regions of the bnodes in Figure 11.

According to Figure 11, there are few differences between sets of the same version. In particular, $X_1$ is a subset of all other sets, hindering the choice of its best mapping option. The inclusion of different values can contribute to a better choice of bnode pairs, as illustrated in Figure 12.

Figure 13 illustrates the approximation space $A'_{1-4}$, considering all bnodes in the datasets in Figure 12. Now, the bnodes within the same version have greater diversity, allowing a better approximation measure between bnodes if we consider the different versions. In this case, the new values of predicate 'type' leads to a better choice of the candidates, where we have $\gamma_{\text{Ainf}}(X_1, X_3) = 0.25$; $\gamma_{\text{Ainf}}(X_1, X_4) = 0.5$; $\gamma_{\text{Ainf}}(X_2, X_3) = 0.5$; and $\gamma_{\text{Ainf}}(X_2, X_4) = 0.11$. Thus, the new delta obtained from *ApproxMap* contains five triples.

In particular, it may be difficult for *ApproxMap* to choose the best mapping option if there are multiple candidates representing approximately equal sets in an approximation space, i.e., sets with the same lower and upper approximations [7], as exemplified in Figure 14a.

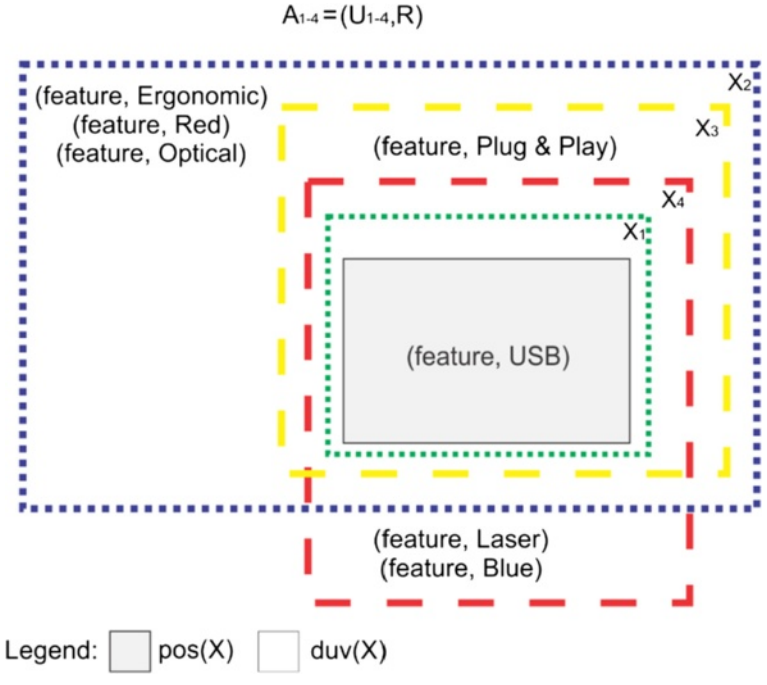


**Figure 11** Approximation space $A_{1-4}$ for bnodes in Figure 10.

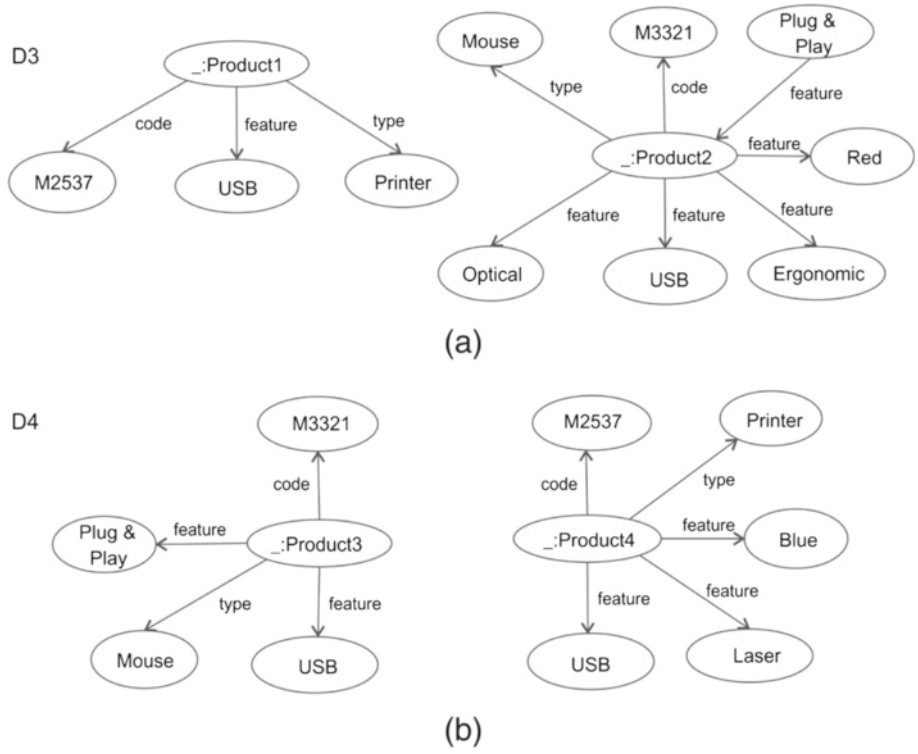


**Figure 12** Sample datasets with greater diversity between bnodes **(a,b)**.

The adopted metrics may be insufficient to distinguish these sets.

Furthermore, in cases involving completely different datasets, *ApproxMap* compares all bnodes in the two datasets during the mapping, resulting in the maximum delta equal to the sum of the triples in the two datasets. We included some optimizations in *ApproxMap*, reducing the cost of comparing distinct pairs, by first checking for the presence of common predicates. The worst-case execution corresponds to a particular case of distinct

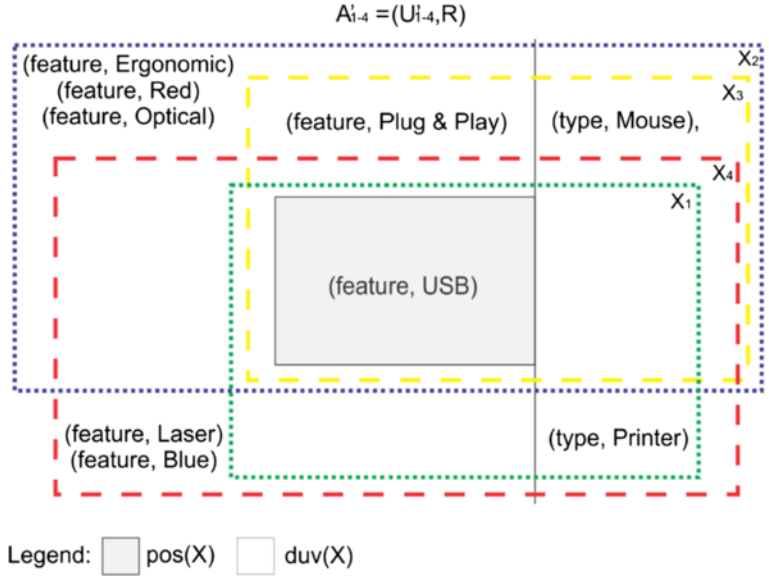


**Figure 13** Approximation space $A'_{1-4}$ for the bnodes in Figure 12.

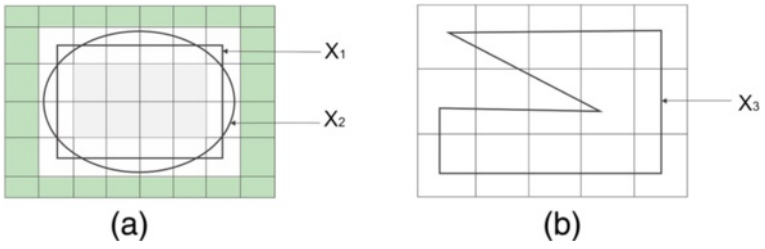


**Figure 14** Approximately equal sets and a dispersed set **(a,b)**.

datasets, where all bnodes have the same predicates. In this case, we obtain dispersed approximate sets representing the bnodes, i.e., sets with an empty lower approximation ($\gamma_{\text{Ainf}}(X_i, X_j) = 0$) and an upper approximation equal to the set universe ($\gamma_{\text{Asup}}(X_i, X_j) = |U|$) [7], as shown in Figure 14b.

We use step $\eta_1$ to control the number of comparisons between bnodes, where the total number is given by $1/\eta_1 \times O(n^2)$. Thus, when $\eta_1$ is considerably smaller than $1/n$, where $n$ gives the smallest number of bnodes in the datasets, in the worst case, the time complexity of the algorithm is $O(n^2)$. Conversely, the best case execution of *ApproMap* occurs with equivalent datasets containing bnodes with varying numbers of connections and without any directly connected bnodes. In this case, we need to compare each bnode with exactly one bnode in the other version. Thus, the complexity of the best case is $\Omega(n)$.

Finally, we intend to apply *ApproxMap* to configuration management of software engineering projects, specifically to version control of RDF datasets. These projects are characterized by the manipulation of data, information, and knowledge in various types and formats, manually constructed based on the modularity principle, where complex elements are divided into smaller parts. Therefore, we expect great diversity between bnodes in the same version, justifying the application of *ApproxMap* in this context.

Because the datasets involved are usually constructed using an incremental development approach, we expect satisfactory performance of *ApproxMap* on similar versions, containing several approximately equivalent bnode pairs, as generally occurs in successive versions of software engineering artifacts. A recommended configuration management practice is to perform version control considering the low percentage of changes between versions. If this does not occur, larger deltas prevent the recovery of intermediate states between successive versions.

**Table 1 Information about real datasets**

| Dataset | $\|B\|$ | $\|G\|$ | $D_a$ | $b_{\text{density}}$ | $b_{\text{len}}$ |
|---|---|---|---|---|---|
| *Swedish* | 522 | 3,670 | 5.47 | 0.00 | 0.00 |
| *Italian* | 6,390 | 49,897 | 3.42 | 0.00 | 0.00 |

As future work, we propose a meticulous analysis of the impact of the adopted metrics and strategies on the mapping. We also intend verifying the applicability of other RST metrics that could provide better approximation measures between bnodes. As a further future work, we propose improving the performance of the algorithm, taking into account execution of some operations in parallel, such as comparison of approximate sets.

## Results and discussion

In analyzing the performance of the *ApproxMap* algorithm, we considered both the delta size calculated from mapping pairs of RDF datasets and the time spent on this task. This allowed comparison of the results and values obtained for the $Alg_{Hung}$ and $Alg_{Sign}$ algorithms, presented by Tzitzikas et al. [6]. All experiments discussed in this section were executed on an Intel Core i7-3537U, 2.0 GHz processor, with 8 GB RAM and running Ubuntu 13.10. To correct any formatting or encoding issues, preprocessing was carried out on certain pairs of datasets.

Three metrics defined by Tzitzikas et al. [6] were used in the analysis of the experiments: $b_{\text{density}}$, $b_{\text{len}}$, and $D_a$. Let $N$ and $B$ denote, respectively, the sets of nodes and blank nodes of graph $G$, where $B \subseteq N$. Further, let $\text{conn}(b)$ denote the set of nodes in $G$ directly attached to $b \in N$. Then, we have $b_{\text{density}} = avg_{b \in B}(|\text{conn}(b) \cap B|/|\text{conn}(b)|)$; $b_{\text{len}}$ refers to the average maximum path length, with

**Table 2 Results of the algorithms applied to real datasets**

| Dataset | | *Swedish* | *Italian* |
|---|---|---|---|
| Delta (triples) | *ApproxMap* 1/10% | 297 | 6 |
| | *ApproxMap* 5/12% | 297 | 6 |
| | *ApproxMap* 5/25% | 297 | 6 |
| | $Alg_{Hung}$ | 297 | 6 |
| | $Alg_{Sign}$ | 423 | 6 |
| Time (ms) | *ApproxMap* 1/10% | 113 | 170 |
| | *ApproxMap* 5/12% | 36 | 158 |
| | *ApproxMap* 5/25% | 34 | 153 |
| | $Alg_{Hung}$ | 4,789 | 456,173 |
| | $Alg_{Sign}$ | 37 | 59 |

**Table 3 Crawled datasets using the load-balancing strategy**

| Instance number | $\|B\|$ | $\|G\|$ | $D_a$ | $b_{\text{density}}$ | $b_{\text{len}}$ |
|---|---|---|---|---|---|
| 1 | 19 | 1,048 | 9.00 | 0.01 | 0.11 |
| 2 | 83 | 11,555 | 7.31 | 0.00 | 0.00 |
| 3 | 361 | 28,208 | 5.93 | 0.00 | 0.00 |
| 4 | 362 | 28,219 | 5.96 | 0.00 | 0.00 |
| 5 | 893 | 15,337 | 4.40 | 0.00 | 0.02 |

vertexes consisting only of bnodes; and $D_a$ corresponds to the average number of bnode triples.

Except for the last experiment, we tested the *ApproxMap* algorithm with three different sets of parameters: $\eta_1 = 0.01$ and $\eta_2 = 0.1$; $\eta_1 = 0.05$ and $\eta_2 = 0.125$; and $\eta_1 = 0.05$ and $\eta_2 = 0.25$ where these tests are denoted, respectively, as *ApproxMap* 1/10%, *ApproxMap* 5/12%, and *ApproxMap* 5/25%. We chose these steps empirically, considering the desired number of iterations. As future work, we propose further analysis of the choice of step values and calibration of *ApproxMap*.

We used the *ApproxMap* 5/12% tests as the baseline for comparison when evaluating the impact of changes in $\eta_1$ and $\eta_2$ on the results. The *ApproxMap* 5/12% test includes 20 iterations ($1/\eta_1$) of the inner loop of the method, comparing each bnode with those in the same hierarchical partition in the second version. In addition, there are eight iterations ($1/\eta_2$) of the outer loop, comparing the bnodes with those in the remaining partitions. The *ApproxMap* 5/25% test was used to verify the impact of an increase in $\eta_2$, reducing the comparisons between distinct partitions for 4 iterations. Finally, we used the *ApproxMap* 1/10% tests to analyze the impact of a reduction in $\eta_1$, increasing the comparisons between the same partitions for 100 iterations. In these tests, we also adjusted $\eta_2$ to better fit $\eta_1$, resulting in ten iterations of the outer loop.

We organized the experiments in three groups based on the type of dataset used in each: real, extracted from the Web (i.e., crawled), or synthetic datasets, as discussed in the following sections. The standard units for delta size and mapping time are, respectively, triples and milliseconds. We used a logarithmic scale for charts showing mapping times of the algorithms, thereby providing better visualization and comparison of the results.

### Real datasets

In the first group of experiments, we used the same real datasets tested by Tzitzikas et al. [6]. Table 1 describes the main features of these datasets, where columns $|B|$ and $|G|$ denote, respectively, the average numbers of bnodes and triples in the version pairs. Measurements for the dataset *Italian* are the same for both files. In the *Swedish* dataset, the coefficient of variation ($cv$) is equal to 2.42%, 2.98%, and 0.19%, for $|G|$, $|B|$, and $D_a$, respectively.

Table 2 gives the results obtained by the algorithms in the first experiment, considering the time to map blank nodes and the delta size calculated from this mapping. In terms of the delta, we obtained the same values for both datasets in all algorithm tests, with the exception of $Alg_{Sign}$. With respect to the execution time, considering the ratio between the time of the algorithms $Alg_{Hung}$ and *ApproxMap* 5/25%, we obtained 141 and 2,982, respectively, for the *Swedish* and *Italian* datasets. Thus, $Alg_{Hung}$ required considerable additional time, particularly for dataset *Italian*.

### Crawled datasets

Owing to the difficulty in finding appropriate real versioned datasets for the experiments, in the second group of experiments, we used an RDF crawler, *LDSpider* [21],

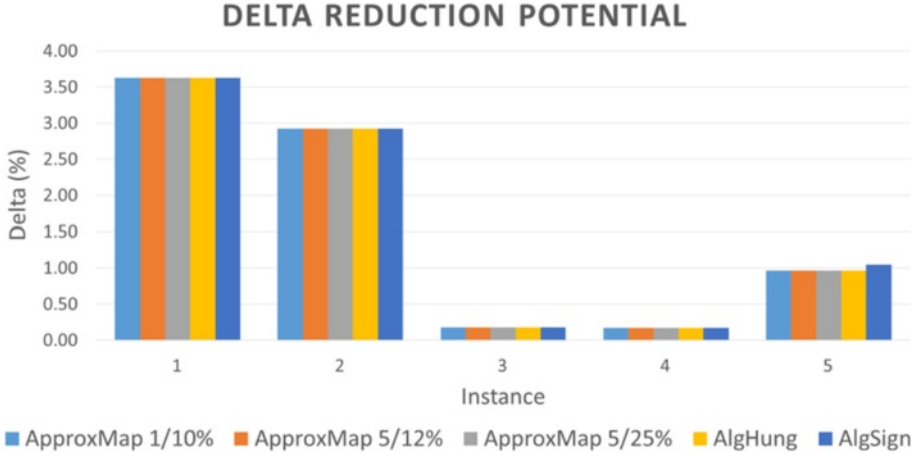


**Figure 15** Delta reduction for load-balancing crawled datasets.

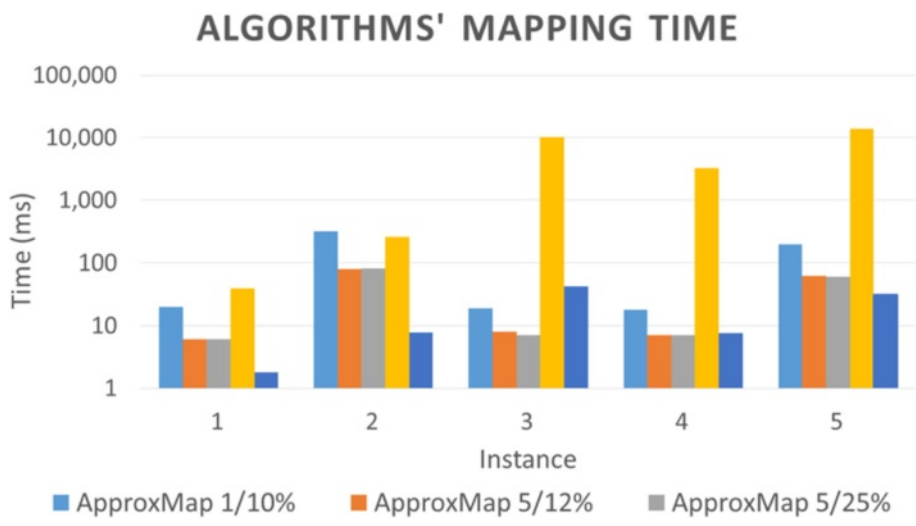


**Figure 16** Execution of load-balancing crawled datasets.

to construct pairs of RDF dataset versions. We extracted some versions from randomly chosen links to common datasets in the linked open data (LOD) cloud [22], such as Dbpedia and DBPL, as well as FOAF Profiles.

We used LDSpider because of its dual crawling strategies [21]: breadth-first and load-balancing. Thus, we executed two experiments based on these strategies, where the maximum number of URIs was limited to generate reasonably sized files for the tests, considering the computational costs of the algorithms. In the first experiment using *LDSpider*, we adopted the load-balancing strategy, with the aim of obtaining pairs of files with approximately the same size. Table 3 gives information about the crawled datasets used. The first column denotes the instance number. All values in Table 3 were identical for both produced versions, with the exception of metric $|G|$ in the second instance, where $cv = 0.73\%$.

Figures 15 and 16 illustrate the results for the datasets described in Table 3. To further analyze the delta reduction potential of the algorithms, the deltas are presented as a percentage of the average number of triples, i.e., $\Delta(G_1, G_2)/|G|$. As seen in Figure 15, all algorithms achieved the same delta reduction on all datasets, except for $Alg_{Sign}$, which showed an increase of 0.08 in the delta reduction potential, in the last instance, compared with the potential of the other algorithms.

For bnode mapping, algorithm $Alg_{Hung}$ was the slowest. Considering the differences between the mapping times of the algorithms presented in Figure 16, compared with *ApproxMap* 5/25%, $Alg_{Hung}$ showed an increase in mapping time between 0.50 and 3.16, on the adopted logarithmic scale. *ApproxMap* 5/25% was faster than $Alg_{Sign}$ in two instances, with the maximum time increase for $Alg_{Sign}$ equal to 0.78. $Alg_{Sign}$ was faster in the remaining instances, with an increase in time for *ApproxMap* 5/25% less than 1.03. Finally, considering the differences between steps $\eta_1$ and $\eta_2$, compared with *ApproxMap* 5/12%, the mapping time for *ApproxMap* 1/10% increased by between 0.38 and 0.60, while *ApproxMap* 5/25% showed a maximum reduction in mapping time of 0.06.

In the second experiment using *LDSpider*, we extracted the first version using the breadth-first strategy and the second using the load-balancing strategy. Once again, we aimed to create files with approximately the same size but with major differences due to the change in strategy. Table 4 shows features of the instances considered in this

**Table 4 Crawled datasets with breadth-first/load-balancing strategy**

| Instance number | $\|B\|$ | | $\|G\|$ | | $D_a$ | | $b_{\text{density}}$ | | $b_{\text{len}}$ | |
|---|---|---|---|---|---|---|---|---|---|---|
| | File 1 | File 2 | File 1 | File 2 | File 1 | File 2 | File 1 | File 2 | File 1 | File 2 |
| 1 | 169 | 19 | 4,355 | 1,048 | 5.73 | 9.00 | 0.21 | 0.01 | 16.26 | 0.11 |
| 2 | 190 | 83 | 11,892 | 11,470 | 5.82 | 7.31 | 0.07 | 0.00 | 1.67 | 0.00 |
| 3 | 1,246 | 893 | 24,364 | 15,337 | 5.13 | 4.40 | 0.10 | 0.00 | 10.88 | 0.02 |
| 4 | 1,963 | 361 | 27,650 | 28,208 | 6.75 | 5.93 | 0.00 | 0.00 | 0.00 | 0.00 |
| 5 | 1,967 | 362 | 28,031 | 28,219 | 6.74 | 5.96 | 0.00 | 0.00 | 0.01 | 0.00 |

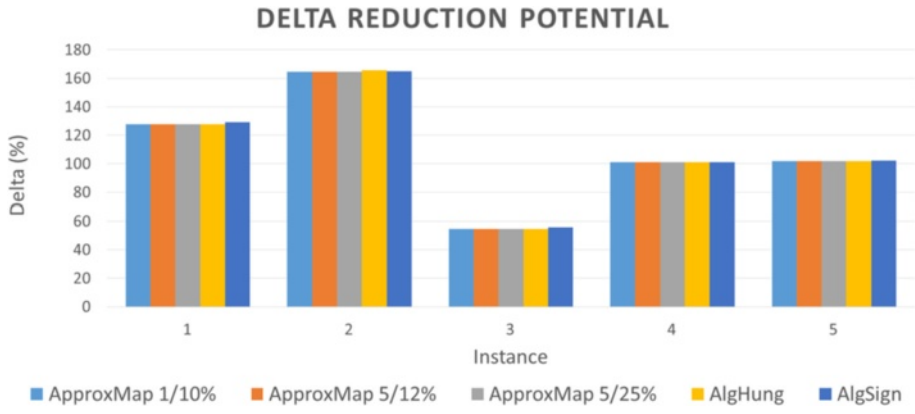


**Figure 17** Delta reduction for breadth-first/load-balancing crawled datasets.

experiment. Detailed information is given in this table, because there are considerable differences between some versions.

Figure 17 shows the delta reduction potential of the algorithms in these tests. $Alg_{Hung}$ showed an increase in its delta reduction percentage of 1.26, while the increase in potential of $Alg_{Sign}$ was less than 1.42, when compared with all tests using *ApproxMap*.

Figure 18 compares the mapping times on a logarithmic scale. $Alg_{Hung}$ once again performed the worst. Compared with *ApproxMap* 5/25%, the mapping times of $Alg_{Hung}$ increased by between 0.11 and 1.92. Compared with $Alg_{Sign}$, the increase in mapping times of *ApproxMap* 5/25% varied between 0.46 and 1.59. Besides, we also verified a time increase for the *ApproxMap* tests; the mapping time of *ApproxMap* 1/10% increased by between 0.42 and 0.60, while that of *ApproxMap* 5/25% decreased by 0.12 , compared with *ApproxMap* 5/12%.

**Synthetic datasets**

In this final group of experiments, to evaluate the algorithms in the mapping of datasets with some specific features, e.g., directly connected bnodes or equivalent datasets, we generated pairs of synthetic datasets for use in the tests.

***Datasets from adapted Univ-Bench artificial generator***

Initially, we considered the datasets used by Tzitzikas et al. [6], the generator of which was based on the Univ-Bench artificial (UBA) data generator [23]. Table 5 lists the features of the dataset pairs tested in this experiment, where all datasets have 240 bnodes. In this table, column $\Delta_{\text{opt}}/|G|$ displays the ratio between the optimal values, represented by $\Delta_{\text{opt}}(G_1, G_2)$, and $|G|$. For the average values shown in this table, we have $cv = 16.17\%$ and $cv = 1.3\%$ to $b_{\text{len}}$, respectively, for instances 4 and 9, and $cv < 0.22\%$ in the other cases.

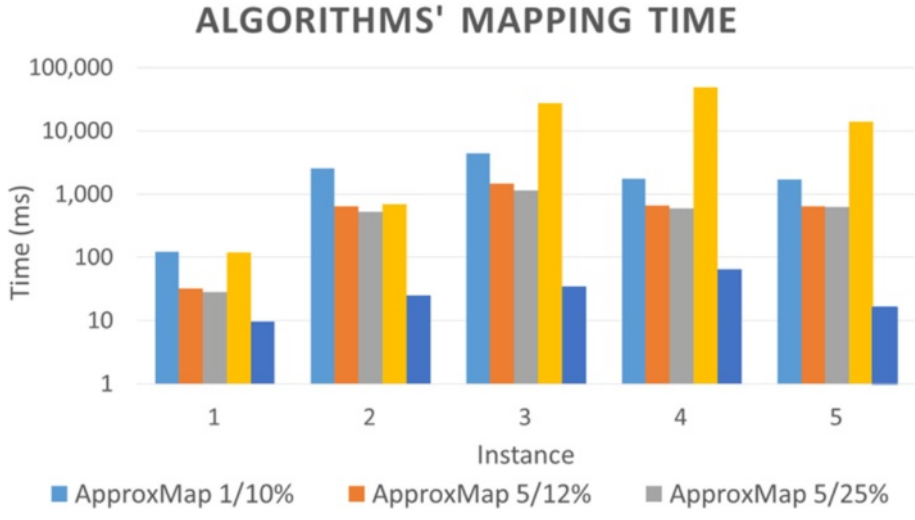


**Figure 18** Execution for breadth-first/load-balancing crawled datasets.

**Table 5 Synthetic datasets generated by Tzitzikas et al. [6]**

| *Instance number* | $\|G\|$ | $D_a$ | $b_{\text{density}}$ | $b_{\text{len}}$ | $\Delta_{\text{opt}}/\|G\|$ (%) |
|---|---|---|---|---|---|
| 1 | 5,846 | 13.4 | 0 | 0 | 1 |
| 2 | 5,025 | 10.5 | 0.1 | 1 | 0.5 |
| 3 | 2,381 | 7 | 0.15 | 1 | 1.5 |
| 4 | 1,628 | 5 | 0.2 | 1 | 1.5 |
| 5 | 1,636 | 5 | 0.2 | 1.15 | 1 |
| 6 | 1,399 | 4 | 0.25 | 1.15 | 1.7 |
| 7 | 919 | 3 | 0.32 | 1.15 | 3.2 |
| 8 | 909 | 3.25 | 0.4 | 1.35 | 2.7 |
| 9 | 1,001 | 3.94 | 0.5 | 21.5 | 2.5 |

Figure 19 shows the delta sizes obtained for the nine version pairs. *ApproxMap* and $Alg_{Hung}$ found the optimal delta in five instances. $Alg_{Hung}$ found a smaller delta than *ApproxMap* 5/25% in instances 4 and 7, with the maximum decrease in delta equal to 0.65. Moreover, in the latter instance, $Alg_{Hung}$ was only surpassed by *ApproxMap* 5/25%. We confirmed the smallest delta reduction potential in instance 8, where *ApproxMap*, $Alg_{Hung}$, and $Alg_{Sign}$ showed increases of 5.34, 7.98, and 8.86, respectively, when compared with the optimal reduction potential.

Figure 20 shows the results for the same nine pairs but considering the second version in reverse order. Contrary to the results of the algorithms proposed by Tzitzikas et al. [6], the bnode order did not affect the results of *ApproxMap*. For the final four instances, compared with the optimal reduction potential, $Alg_{Hung}$ showed an increase varying from 6.72 to 19.64. Similarly, for these instances, the increase in the reduction potential of $Alg_{Sign}$ varied from 8.86 to 20.75, compared with the optimal values.

Figure 21 shows the results for the mapping times of the algorithms on a logarithmic scale. $Alg_{Hung}$ achieved the worst performance, with increased mapping time varying between 1.53 and 2.45, compared with *ApproxMap* 5/25%. In the final three instances, $Alg_{Sign}$ achieved better performance than *ApproxMap* 5/25%, with decreased mapping time of between 0.12 and 0.65. *ApproxMap* 5/25% performed better in the first instance, executing faster than $Alg_{Sign}$ with a decrease in time of 0.98. Finally, compared with *ApproxMap* 5/12%, the mapping time of *ApproxMap* 1/10% showed a maximum increase of 0.52, while that of *ApproxMap* 5/25% decreased by 0.16.

#### *Datasets with directly connected bnodes*

In the final three instances of the nine pairs of synthetic datasets used above, all algorithms obtained higher deltas than the optimal values, when applied to datasets with a marked increase in the number of directly connected bnodes. To assess the influence of $b_{\text{density}}$ on delta size, we conducted a new experiment with identical versions of the datasets. All algorithms found an empty delta for the original order of the datasets. However, considering a reverse order in the second version, Figure 22 shows the results in terms of the deviation from the optimal delta, i.e., $dx = \frac{\Delta(G_1,G_2)-\Delta_{\text{opt}}(G_1,G_2)}{\Delta_{\text{opt}}(G_1,G_2)+1}$, where $\Delta_{\text{opt}}$ denotes the optimal value. With an increase in $b_{\text{density}}$, the reverse ordering once again influenced the results of $Alg_{Hung}$ and $Alg_{Sign}$. These algorithms achieved the same increase in $dx$ in the final three instances, whereas *ApproxMap* produced an empty delta.

To analyze the performance of the algorithms, considering datasets with a higher number of directly connected bnodes, we developed an RDF dataset generator, based on that included in the Berlin SPARQL Benchmark (BSBM) [24]. We used this generator to produce pairs of file versions with an average $b_{\text{density}}$ of 0.34%, and $cv = 7.25\%$. We discuss the experiments using this generator in the next section.

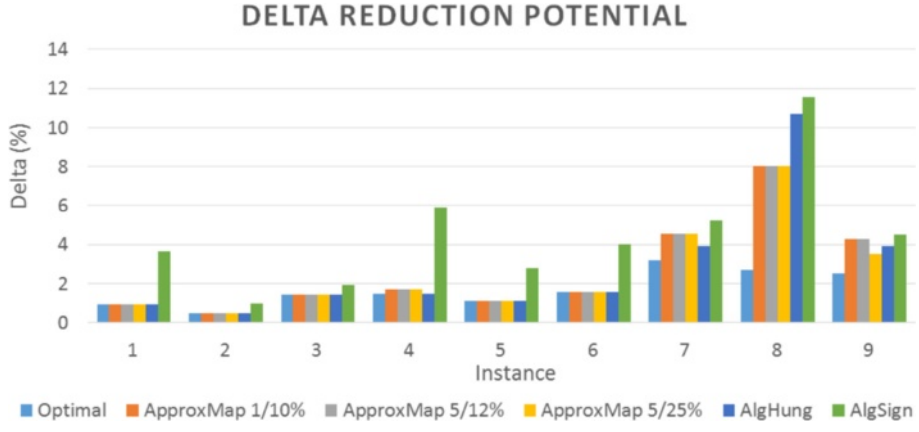


**Figure 19** Delta reduction for nine pairs of synthetic datasets.

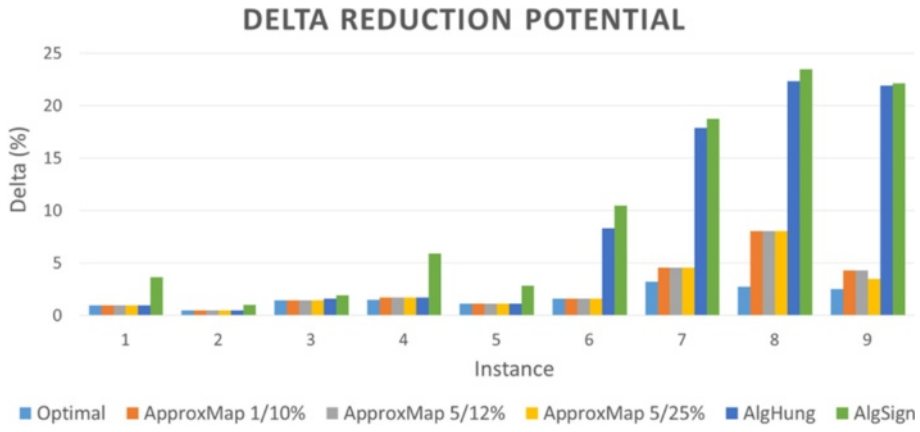


**Figure 20** Delta reduction for the nine pairs with the second version in reverse order.

***Datasets from adapted BSBM generator***

Our adapted generator is capable of producing two versions of an e-commerce portal, which is used by vendors to offer various products and consumers to submit reviews about these products. The versions contain descriptions for five different types of resources, as well as three different types of blank nodes. We determined these quantities empirically to obtain the desired value for $b_{\text{density}}$. Thus, we defined the elements corresponding to products, their types, and characteristics as blank nodes, although the portal also included a hierarchy of product types.

In all experiments, 74.73% of the triples contained bnodes, with the coefficient of variation, $cv = 1.28\%$. The high percentage of bnodes was acceptable because triples without bnodes could be mapped directly and this was not our concern. Moreover, except for the last experiment in which we tested large datasets, we limited the maximum number of bnodes to 2,000, so that the datasets could be tested with all algorithms. We constructed the version pairs in such a way that ensured that changes occurred in isolation. The intersections of sets of equivalent, added, or removed triples between versions were empty, considering all possible bnode mappings. Based on this, we obtained by construction the optimal delta size for the tested pairs.

The adapted generator accepts as input the number of products sold on the e-commerce portal and then, determines the number of other bnodes (product types and characteristics) in terms of this input number. As a result, the values of some metrics were affected by the numbers of bnodes, such as the average maximum path length ($b_{\text{len}}$), due to variations in the product type hierarchy. However, this metric does not affect the computational cost of *ApproxMap*. We dealt with bnode hierarchies using the adopted bottom-up strategy. Similarly, the absence of bnode interconnections ($b_{\text{density}} = 0$) did not affect *ApproxMap* because, in this case, it merely grouped the bnodes in the same hierarchical partition. A meticulous analysis of the impact of these metrics on delta size is suggested as a future work.

In the next sections, we describe the five experiments performed using datasets produced by our adapted generator. These experiments consider increases in the version

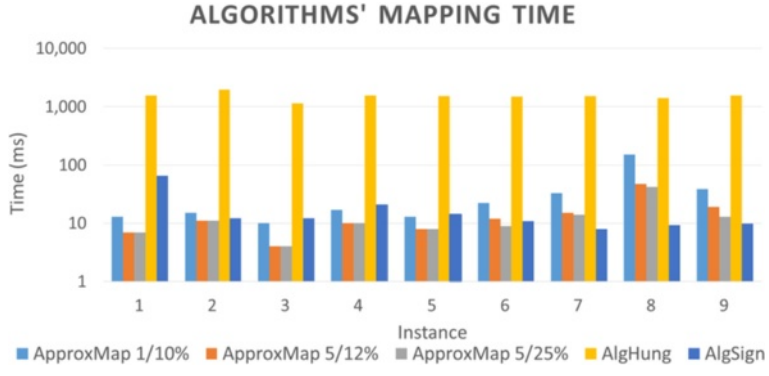


**Figure 21** Results of mapping the synthetic datasets.

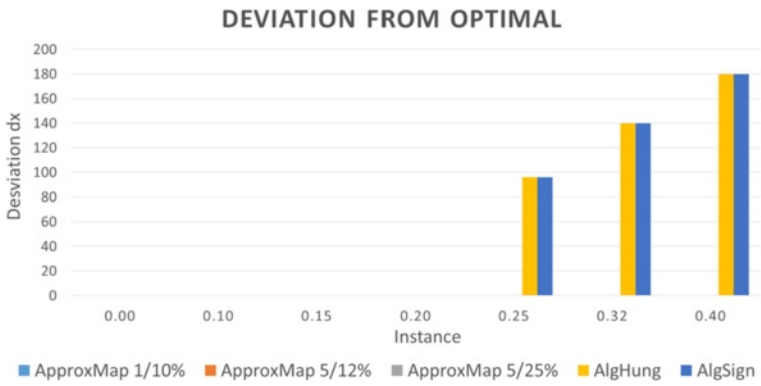


**Figure 22** Deviation from optimal delta for equivalent synthetic datasets.

and delta sizes, identical or different versions, as well as large datasets.

**Changing the size of the datasets** The first experiment using our adapted generator aimed to analyze the impact of an increase in the number of bnodes. For the generation of datasets, we set a fixed ratio of 50% of equivalent elements among pairs, to assess the impact of an increase in version size assuming a moderate delta size.

Table 6 gives the practical information about this experiment involving five version pairs. Considering all the averages shown in this table, we obtained a *cv* smaller than 0.98%, except for values $b_{\text{len}}$, which yields a maximum value of $cv = 4.67\%$. In this table, column $\Delta_{\text{opt}}/|G|$ displays the ratio between the optimal values, represented by $\Delta_{\text{opt}}(G_1, G_2)$, and $|G|$.

Figures 23 and 24 show the results obtained for these datasets, in terms of delta sizes and mapping times of the algorithms, respectively. $Alg_{Sign}$ achieved the worst performance in terms of delta size, with an increase in delta size varying between 33.6 and 36.11. $Alg_{Hung}$ showed an increase in the reduction percentage ranging from 2.6 to 8.88. Finally, the increase in the reduction potential of *ApproxMap* 1/10%, *ApproxMap* 5/12%, and *ApproxMap* 5/25% was, respectively, less than 4.28, 4.53, and 4.7 compared with the optimal reduction potential.

Regarding bnode mapping times, $Alg_{Hung}$ was slower than *ApproxMap* 5/25%, with an increase in time varying between 0.97 and 1.44 on the logarithmic scale. Compared with the $Alg_{Sign}$ algorithm, the increase in time for *ApproxMap* 5/25% varied between 0.65 and 1.70. Finally, the maximum increase in the execution time of *ApproxMap* 1/10% compared with that of *ApproxMap* 5/12% was equal to 0.45. Similarly, *ApproxMap* 5/12% showed an increase smaller than 0.12, compared with *ApproxMap* 5/25%.

**Table 6 Datasets with varying version sizes**

| Instance number | $\|B\|$ | $\|G\|$ | $D_a$ | $b_{\text{density}}$ | $b_{\text{len}}$ | $\Delta_{\text{opt}}/\|G\|$ (%) |
|---|---|---|---|---|---|---|
| 1 | 400 | 3,390 | 7.29 | 0.29 | 60.42 | 53.39 |
| 2 | 800 | 7,000 | 7.78 | 0.33 | 127.34 | 53.02 |
| 3 | 1,200 | 10,806 | 8.30 | 0.37 | 212.71 | 52.74 |
| 4 | 1,600 | 14,061 | 7.88 | 0.34 | 200.95 | 52.49 |
| 5 | 2,000 | 17,541 | 7.86 | 0.34 | 270.09 | 52.71 |

**Changing delta size** While the first experiment using our generator considered varying numbers of bnodes, the second experiment investigated variations in the differences between version pairs. In this experiment, we adopted datasets with a fixed number of 2,000 bnodes, varying the percentage of different elements between 15% and 90% in fixed steps of 15%. With the choice of these percentages, the six pairs generated in this experiment were grouped in doubles to represent the following change levels between versions: low, medium, and high. Table 7 gives information about these pairs. Considering all the averages shown in this table, we obtained a *cv* of less than 2.22%.

To facilitate impact analysis of increasing delta sizes, Figure 25 shows the results of the algorithms, in terms of the percentage difference between the deltas found and the optimal values in terms of these optimal values, i.e., $(\Delta(G_1, G_2) - \Delta_{\text{opt}}(G_1, G_2))/\Delta_{\text{opt}}(G_1, G_2)$.

As before, $Alg_{Sign}$ performed the worst in terms of delta size, with the distance to the optimal delta varying between 51.61 and 88.3. $Alg_{Hung}$ showed a distance to the optimal ranging from 11.45 to 21.28, while *ApproxMap* 1/10% showed one varying between 1.36 and 7.87. For *ApproxMap* 5/12% and *ApproxMap* 5/25%, the distances to the optimal varied from 0.85 to 8.84 and from 5 to 9.39, respectively.

Moreover, Figure 26 shows the results for the mapping times of the algorithms. The time increase for $Alg_{Hung}$ varied between 1.32 and 1.47, compared with *ApproxMap* 5/25%. But, when compared with $Alg_{Sign}$,

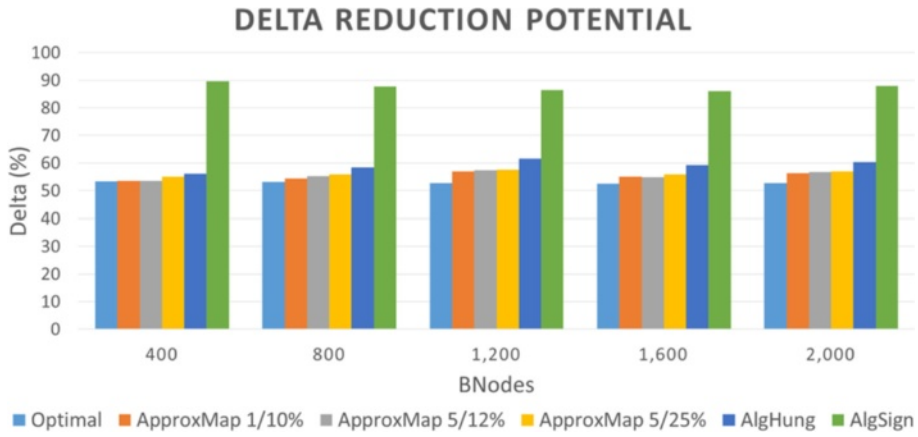


**Figure 23** Results for datasets with varying version sizes.

*ApproxMap* 5/25% required an increased time ranging from 1.54 to 1.92, on the logarithmic scale. Finally, considering *ApproxMap* 5/12%, we observed a time increase less than 0.5 for *ApproxMap* 1/10%, and a time decrease less than 0.13 for *ApproxMap* 5/25%.

**Identical datasets** For a better analysis of the algorithms behavior, the next two experiments considered extreme cases, with the datasets either identical or completely different. In the first case, we compared the second version of the datasets from the first experiment using our adapted generator, with a version created by application of the delta in the first version, i.e., $G_2^{'} = G_1 + \Delta$. With this, we validated the deltas previously found by *ApproxMap*. No differences were found by any of the algorithms for the identical datasets, even when considering the second version in reverse order.

Figure 27 shows the time spent comparing the dataset pairs. Compared with *ApproxMap* 5/25%, $Alg_{Hung}$ required an increased time ranging from 1.24 to 1.51, while $Alg_{Sign}$ required decreased time varying between 0.97 and 1.76. Compared with *ApproxMap* 5/12%, the increase in execution time of *ApproxMap* 1/10% was less than 0.1, while that for *ApproxMap* 5/25% time was less than 0.04, on the adopted logarithmic scale.

**Different datasets** In this experiment, we also based the generation of different datasets on the second versions of the datasets produced in the first experiment using our generator. In this case, we changed all the elements included in the triples and obtained deltas equal to $|G_1| + |G_2|$. Figure 28 shows the results for these pairs, with mapping times expressed on a logarithmic scale.

Compared with *ApproxMap* 5/25%, $Alg_{Hung}$ required increased time varying between 1.4 and 2.21, while the reduced time requirement of $Alg_{Sign}$ varied from 0.94 to 1.92. Compared with the time requirement of *ApproxMap* 5/12%, *ApproxMap* 1/10% showed an

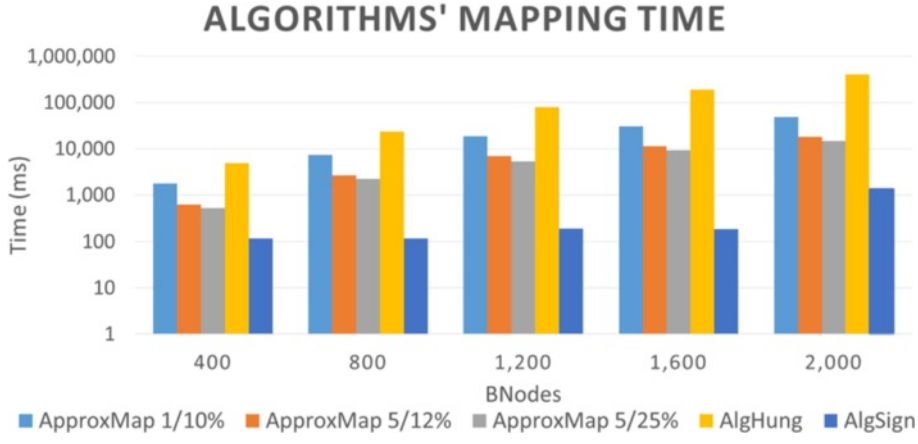


**Figure 24** Execution times for datasets with varying version sizes.

**Table 7 Datasets with varying delta sizes**

| Instance number | $\|G\|$ | $D_a$ | $b_{\text{density}}$ | $b_{\text{len}}$ | $\Delta_{\text{opt}}/\|G\|$ (%) |
|---|---|---|---|---|---|
| 1 | 17,895 | 8.47 | 0.38 | 231.60 | 15.86 |
| 2 | 17,639 | 7.99 | 0.35 | 304.20 | 31.73 |
| 3 | 17,868 | 8.19 | 0.37 | 326.40 | 47.34 |
| 4 | 17,841 | 8.13 | 0.36 | 265.72 | 62.91 |
| 5 | 18,014 | 8.28 | 0.37 | 340.61 | 78.35 |
| 6 | 17,695 | 7.95 | 0.34 | 328.19 | 94.06 |

increase ranging from 0.4 to 0.44, while the time reduction for *ApproxMap* 5/25% varied between 0.11 and 0.15, on the logarithmic scale.

**Large datasets** Finally, the last experiment considered the behavior of *ApproMap* and $Alg_{Sign}$ when mapping large datasets. We could not test $Alg_{Hung}$ in this experiment, owing to its high computational cost. With the aim of reducing the number of comparisons between bnodes, we adopted steps $\eta_1 = 0.2$ and $\eta_2 = 0.5$ for *ApproxMap*, which is referred to as *ApproxMap* 20/50%. Thus, with this choice of steps, there are five iterations comparing bnodes in the same hierarchical partitions and only two iterations comparing bnodes in different partitions.

In the construction of the dataset pairs, the number of bnodes varied between 20,000 and 100,000, in fixed steps of 20,000. We created these five pairs with an average number of triples ($|G|$) of 183,732; 356,176; 562,828; 754,524; and 958,038 assuming a maximum value of $cv = 0.37\%$. We created these datasets with 25% different elements, with the average size of the optimal delta equal to 26.41% of the triples with $cv = 0.11\%$. The adopted values in this experiment were chosen empirically with the aim of reducing the mapping times of the algorithms. We did not test the algorithms with datasets larger than those generated in this experiment, owing to the computational cost of *ApproxMap*. However, the considered instances were sufficient to evaluate the behavior of the algorithm with large datasets. Moreover, the construction of a large dataset is not common practice in the application context of our method, that is, software development projects, with stimulated techniques such as modularization.

Figures 29 and 30 show, respectively, the distance between the results found and the optimal delta size and the algorithms' mapping times. As observed previously, $Alg_{Sign}$ underperformed in terms of delta size, achieving a distance to the optimal delta ranging from 81.87 to 110.87. Furthermore, for *ApproxMap* 20/50%, the distance to the optimal delta varied from 16.76 to 27.33. Regarding time cost, the increased time required by *ApproxMap* relative to that of $Alg_{Sign}$ varied from 2.86 to 3.12 on the logarithmic scale.

### Analysis of results

Satisfactory results of *ApproxMap* in the experiments confirm our hypothesis that mapping bnode pairs with the highest approximation can assist in reducing the delta size. Considering the tests where optimal delta values are known, *ApproxMap* obtained the optimal delta size in 59% of the tests. $Alg_{Hung}$ and $Alg_{Sign}$ found optimal solutions, respectively, in 50% and 30% of the test cases.

Considering all experiments, except the final one with large datasets, *ApproxMap* found a delta size equal to that of $Alg_{Hung}$ in 55% of the tests and smaller than that of $Alg_{Hung}$ in 40% of the cases, except in the tests with steps 5/25%, where the delta found by *ApproxMap* was smaller in 41% of the test cases. Compared with $Alg_{Sign}$, *ApproxMap* obtained a smaller delta in 67% of the cases and the same delta in 33% of the cases. In the experiment with large datasets, *ApproxMap* performed better than $Alg_Sign$ in all cases. Moreover, when compared to $Alg_{Sign}$, $Alg_{Hung}$ found the same delta in 38% of the tests and a smaller delta in 60% of the tested cases.

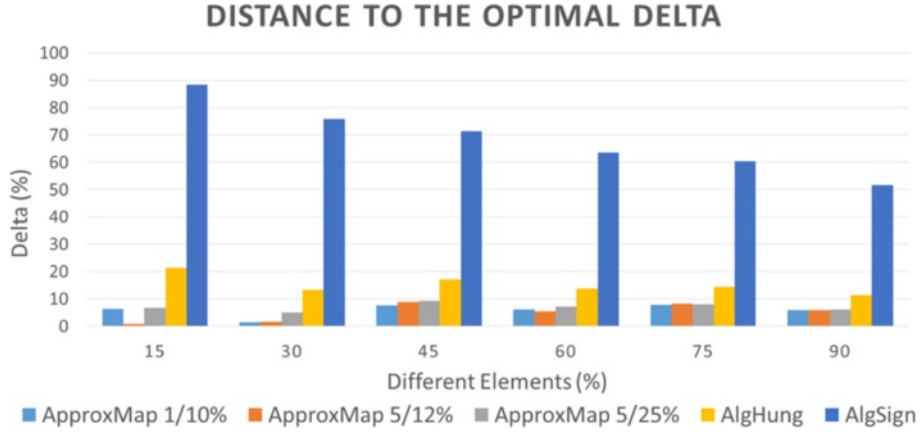


**Figure 25** Results for datasets with varying delta sizes.

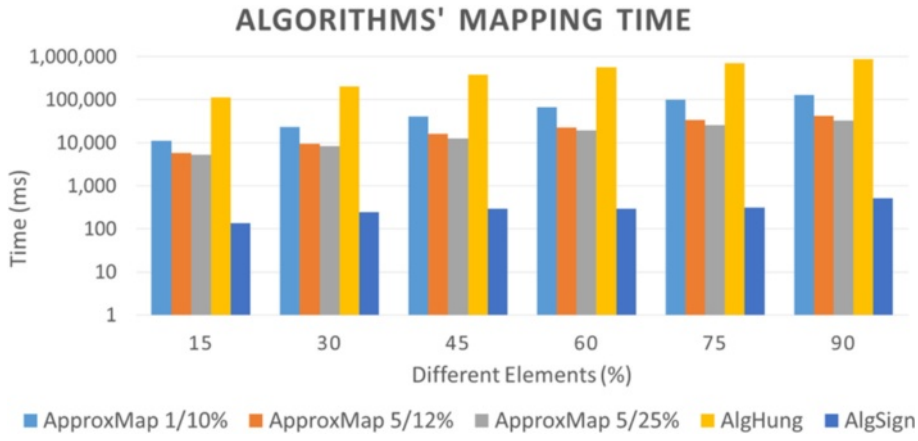


**Figure 26** Execution times for datasets with varying delta sizes.

Regarding mapping time, *ApproxMap* was faster than $Alg_{Hung}$ in 84% of the tests and slower in the remaining 16%, except in the tests with steps 1/10%, where it was outperformed in 21% of the tested instances. *ApproxMap* was faster than $Alg_{Sign}$ in 14%, 21%, and 24% of the tests with steps 1/10%, 5/12%, and 5/25%, respectively, and outperformed in the other cases. In the experiment with large datasets, $Alg_{Sign}$ was faster than *ApproxMap* in all tests. $Alg_{Sign}$ was also faster than $Alg_{Hung}$ in all the tests conducted with these algorithms.

Based on the experimental results, the empirically defined values for parameters $\eta_1$ and $\eta_2$ are considered to be satisfactory. Considering the tests with steps 5/12% as our reference, the decrease in $\eta_1$ from 5% to 1% (steps 1/10%) caused a reduction in the delta size in 12% of the tests, while we obtained the same delta in 81% of the cases. However, the consequent increase in mapping time was confirmed in 98% of the cases, while it remained the same in the other 2% of cases. On the other hand, with the increase in $\eta_2$ from 12.5% to 25% (steps 5/25%), a delta increase occurred in 17% of the tests, while we obtained the same delta in 78% of cases. However, a consequent reduction in mapping time occurred in 64% of the instances, while the time remained the same in 28% of the tested cases.

Furthermore, considering the impact of interconnected bnodes in the experiments, in cases without directly connected bnodes (with $b_{\text{density}} = 0$), *ApproxMap* had the same delta size as $Alg_{Hung}$ in all the tests. But, in cases where $b_{\text{density}} > 0$, *ApproxMap* had the same delta size in 47% of the cases, and a smaller size than that in $Alg_{Hung}$ in 47% of the tests, with the exception of tests with steps 5/25%, where the size was smaller in 49% of the tested instances.

On the other hand, analyzing the algorithms' performance in the experiments with equivalent pairs, *ApproxMap* was faster than $Alg_{Hung}$ in all tests. When considering the ratio between the time spent by these algorithms, the

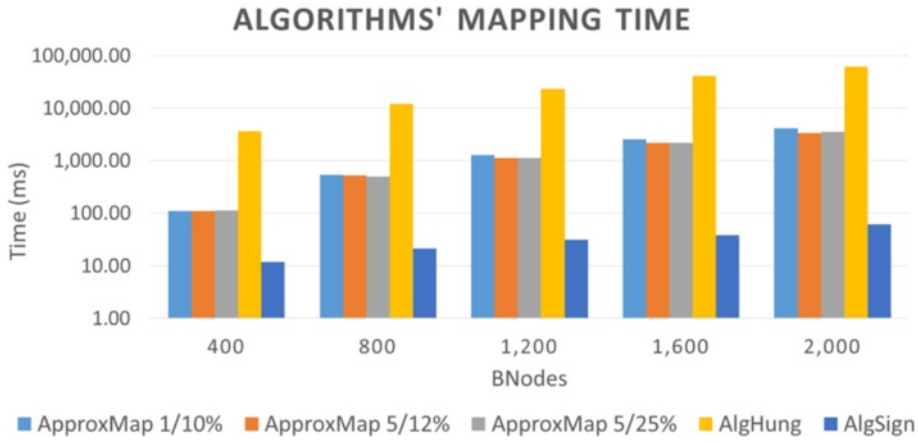


**Figure 27** Execution times for identical datasets.

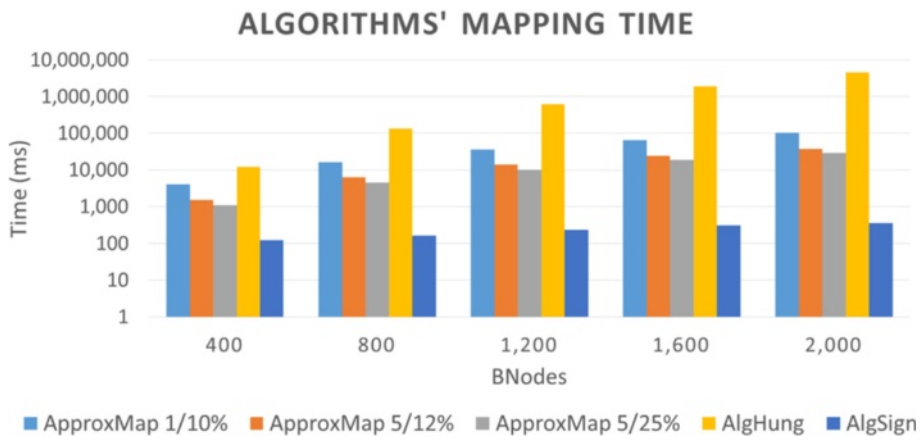


**Figure 28** Execution times for different datasets.

mapping time of $Alg_{Hung}$ was up to 283 times greater than that of *ApproxMap* 5/25%. We also emphasize the results for the real dataset *Italian*, whose delta contained no triples with bnodes. In this case, $Alg_{Hung}$ required a greater mapping time than *ApproxMap* 5/25% with a ratio of 2,982. However, $Alg_{Hung}$ yielded a nonempty delta in 25% of the tests with equivalent datasets but with the datasets in reverse order. The bnode order did not affect *ApproxMap* in the experiments, because it imposes an internal ordering.

Based on these results, we can state that the *ApproxMap* method obtained satisfactory performance in the experiments, and its application is recommended in the versioning of RDF datasets. We intend to apply this algorithm in the design of an SCM method, as part of an integrated environment of tools for software engineering projects, based on the Semantic Web standards. Moreover, we emphasize the satisfactory performance of *ApproxMap* in mapping datasets with large numbers of equivalent elements. Thus, we recommend its application for version control following the recommended practices for SCM, considering a low percentage of changes between versions.

## Conclusions

This paper aimed to develop a heuristic method for mapping blank nodes. The proposed method, called *ApproxMap*, applied extended concepts of RST, presented by Pawlak [7], in the handling of imprecision in bnode mapping. RST provided the necessary support to obtain a mapping between bnodes, seeking closer approximations between bnodes of the considered versions. The proposed modeling of blank nodes as approximate sets in an approximation space is an important contribution of this article. This modeling can be reused in other research domains involving blank node mapping.

In our method, we determined the number of comparisons between bnodes as parameter $\eta_1$. Considering

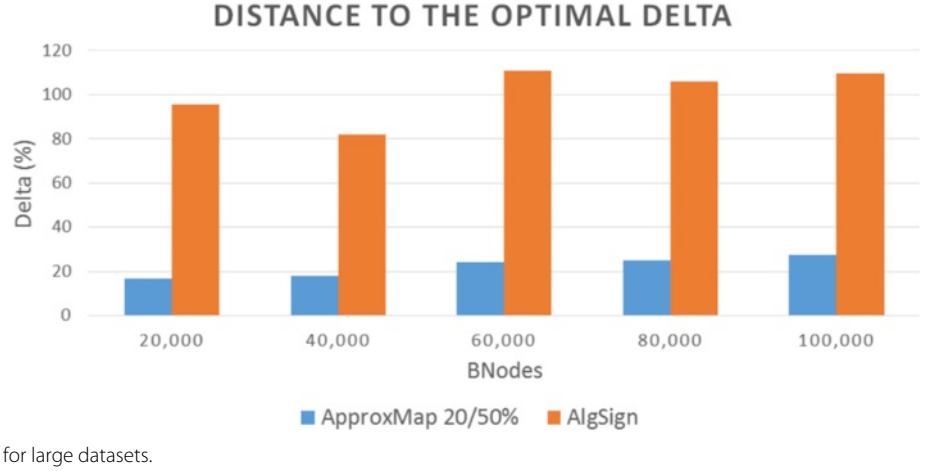


**Figure 29** Results for large datasets.

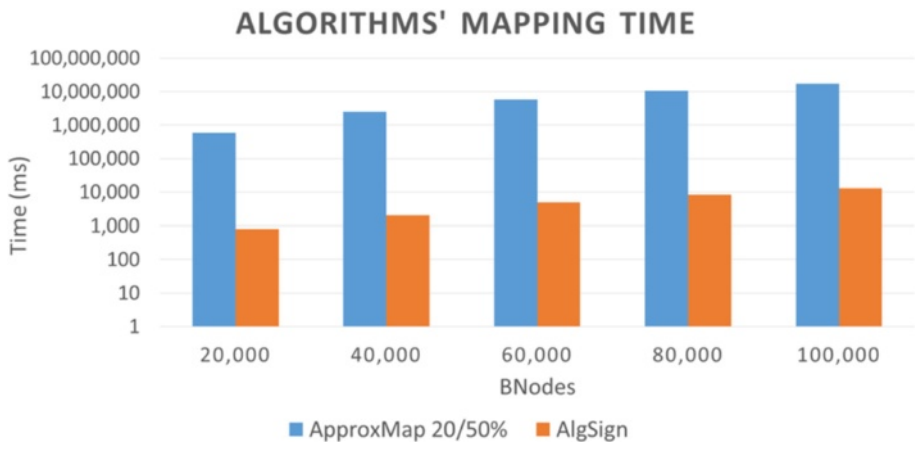


**Figure 30** Execution times for large datasets.

small values for the ratio $1/\eta_1$, the proposed algorithm has worst-case time complexity of $O(n^2)$, involving two completely different datasets, whose bnodes have the same predicates.

*ApproxMap* showed satisfactory performance in our groups of experiments, as the algorithm that obtained solutions closest to the optimal values. This algorithm succeeded in finding the optimal delta size in 59% of the tests involving optimal values. Considering all tests with different values for parameters $\eta_1$ and $\eta_2$, *ApproxMap* achieved a delta size smaller than or equal to those of $Alg_{Hung}$ and $Alg_{Sign}$, respectively, in at least 95% and 100% of the tested cases. Regarding mapping time, *ApproxMap* was faster than $Alg_{Hung}$ in at least 79% of the instances and slower than $Alg_{Sign}$ in at least 76% of the tests.

Despite its mapping time being greater than that of $Alg_{Sign}$, which has a time cost of $n \cdot \log n$, we recommend applying *ApproxMap* in various situations, particularly those involving similar versions and directly connected bnodes. Great diversity between the bnodes in the same version is beneficial for *ApproxMap*. Thus, our algorithm can be successfully applied in RDF dataset versioning, such as that produced by software processes with iterative and incremental development.

As future work, we propose the creation of a parallel version of the *ApproxMap* algorithm to reduce the time required to compare bnodes of the two RDF bases. Furthermore, we propose a meticulous analysis of the appropriate choice of input steps $\eta_1$ and $\eta_2$, and of the impact of the adopted metrics and strategies on delta size. Besides, we also intend investigating other RST metrics.

**Competing interests**
The authors declare that they have no competing interests.

**Authors' contributions**
JAM developed algorithms, performed experiments, and drafted the manuscript. AMC participated in design and coordination of the study. Both authors read and approved the final manuscript.

**Acknowledgements**
Many thanks to Christina Lantzaki and Yannis Tzitzikas for their help in the execution of tests using the $Alg_{Hung}$ and $Alg_{Sign}$ algorithms and also for making their synthetic datasets available. We also thank the reviewers for their help in improving the article.

Received: 21 October 2013 Accepted: 16 October 2014
Published online: 28 April 2015

**References**

1. Klyne G, Carroll JJ, McBride B (2014) RDF 1.1 concepts and abstract syntax. World Wide Web Consortium, Recommendation. http://www.w3.org/TR/rdf11-concepts
2. Monte-Mor JA, Cunha AM (2014) Galo: a semantic method for software configuration management. In: Information Technology: New Generations (ITNG), 2014 11th International Conference On. pp 33–39
3. Antoniou G, van Hatrmelen F (2004) A Semantic Web prime. The MIT Press, London, England. p. 238
4. Lee TB, Connolly D (2001) Delta: an ontology for the distribution of differences between RDF graphs. Technical report, W3C. http://www.w3.org/DesignIssues/Diff
5. Zeginis D, Tzitzikas Y, Christophides V (2011) On computing deltas of RDF/s knowledge bases. ACM Trans Web 5(3):14–11436
6. Tzitzikas Y, Lantzaki C, Zeginis D (2012) Blank node matching and RDF/s comparison functions. In: Proceedings of the 11th International Conference on The Semantic Web - Volume Part I. ISWC'12. Springer, Berlin, Heidelberg. pp 591–607
7. Pawlak Z (1982) Rough sets. Int J Comput Inform Sci 11:341–356
8. do Carmo Nicoletti M, Uchôa JQ, Baptistini MTZ (2001) Rough relation properties. Int J Appl Math Comput Sci 11(3):621–635
9. Carroll JJ (2002) Matching RDF graphs. In: Proceedings of the First International Semantic Web Conference on The Semantic Web. ISWC '02. Springer, London, UK. pp 5–15
10. Noy NF, Kunnatur H, Klein M, Musen MA (2004) Tracking changes during ontology evolution. In: ISWC2004, Proceeding of the 3rd International Semantic Web Conference, Hiroshima, Japan, November 7-11, 2004. Springer, Berlin, Heidelberg. pp 259–273
11. Noy NF, Musen MA (2002) Promptdiff: a fixed-point algorithm for comparing ontology versions. In: Eighteenth National Conference on Artificial Intelligence. American Association for Artificial Intelligence, Menlo Park, CA, USA. pp 744–750
12. Noy NF, Musen MA (2004) Ontology versioning in an ontology management framework. IEEE Intell Syst 19(4):6–13
13. Auer S, Herre H (2006) A versioning and evolution framework for RDF knowledge bases. In: Proceedings of the 6th International Andrei Ershov Memorial Conference on Perspectives of Systems Informatics. PSI'06. Springer, Berlin, Heidelberg. pp 55–69

14. Völkel M, Groza T (2006) SemVersion: An RDF-based ontology versioning system. In: Nunes MB (ed). Proceedings of IADIS International Conference on WWW/Internet (IADIS 2006), Murcia, Spain. pp 195–202
15. Cassidy S, Ballantine J (2007) Version control for RDF triple stores. In: Filipe J, Shishkov B, Helfert M (eds). ICSOFT 2007, Proceedings of the Second International Conference on Software and Data Technologies, Volume ISDM/EHST/DC, Barcelona, Spain, July 22-25, 2007. INSTICC Press, Setubal, Portugal. pp 5–12
16. Im D-H, Lee S-W, Kim H-J (2012) A version management framework for RDF triple stores. Int J Softw Eng Knowl Eng 22(1):85–106
17. Zeginis D, Tzitzikas Y, Christophides V (2007) On the foundations of computing deltas between RDF models. In: Proceedings of the 6th International The Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference. ISWC'07/ASWC'07. Springer, Berlin, Heidelberg. pp 637–651
18. Kuhn HW (1955) The Hungarian method for the assignment problem. Naval Res. Logist. Quart 2:83–97
19. Uchôa JQ (1998) Representação e indução de conhecimento usando teoria de conjuntos aproximados. Master's thesis, Universidade Federal de São Carlos, São Carlos, Brasil
20. Pawlak Z, Skowron A (2007) Rough sets: some extensions. Inform Sci 177(1):28–40
21. Isele R, Umbrich J, Bizer C, Harth A (2010) Ldspider: An open-source crawling framework for the web of linked data. In: Polleres A, Chen H (eds). ISWC Posters & Demos. CEUR Workshop Proceedings. CEUR-WS.org Vol. 658
22. Bizer C, Heath T, Berners-Lee T (2009) Linked data - the story so far. Int J Semantic Web Inf Syst 5(3):1–22
23. Guo Y, Pan Z, Heflin J (2005) LUBM: a benchmark for owl knowledge base systems. Web Semant 3(2-3):158–182
24. Bizer C, Schultz A (2009) The Berlin SPARQL benchmark. Int J Semantic Web Inform Syst 5(2):1–24