RDF Graph Alignment with Bisimulation

Peter Buneman[1] Sławek Staworko[1,2,3]
[1]University of Edinburgh [2]University of Lille [3]LINKS, INRIA Nord-Europe
{opb,sstawork}@inf.ed.ac.uk

ABSTRACT

We investigate approaches to the aligningment of two RDF triple stores inspired by the classical notion of graph bisimulation and relate this to metrics that describe the accuracy of the alignment. Alignment of RDF stores is essential in the understanding of evolving, curated ontologies. One needs to align nodes in sucessive versions that are given "blank" names; moreover, a new version may switch terminology from one external ontology to another. This means that some further alignment of the identifiers of the two external ontologies is useful.

We first describe a form of bisimulation based on node colorings, in which the colors define equivalence classes, we then go on to describe an estimate of the accuracy of a coloring based on (a) the similarity of literal nodes and (b) on the "structural" similarity of internal nodes (URIs and blank nodes).

The effectiveness of these method is tested on two evolving data sets: an ontology described in OWL and triple store derived from an evolving relational database. Both of these are curated resources for biologists.

1. INTRODUCTION

Identifying references to the same real-life entity is one of the most fundamental concerns in databases. It plays an important if not crucial role in virtually all non-trivial data processing tasks from computing join of two tables to removing duplicate entries in data cleaning [15] to combining data objects in multiple databases in data integration [3]. This problem comes in a number of flavors depending on the type of data used to identify the entity represented by a given data object. Ideally, as in the case of a well-designed stand-alone database, a consistent system of unique identifiers supports the linkage of objects in a manner that is reliable and efficient. However, independent databases may use different and often incompatible schemes of unique identifiers. Consequently, linking their contents may require using other methods, based on data values and the structure of the databases to match corresponding identifiers [8].

In this paper we study an instance of this problem that arises in the context of evolving RDF graphs: for two consecutive versions of an RDF graph we wish to construct an *alignment* that connects pairs of nodes in the two versions that represent the same entity. RDF is essentially an edge-labeled graph that uses *URIs* (Unique Resource Identifiers) as nodes and edge labels but also has *blank* nodes, which are not persistent identifiers, as well as *literal* nodes, which store (unique) data strings. Because of the varied types of nodes aligning two RDF graphs presents a number of interesting challenges. While it is reasonable to assume that two nodes labeled with the same URI represent the same entity, the converse is not necessarily true. Indeed, the same entity may be represented in different versions with different identifiers, for instance, as a result of changing the scheme of attributing URIs. Even more problematic are blank nodes, which although discouraged are often misused when using reification for purpose of representing data structures such as lists and records [5]. Because blank nodes are not persistent identifiers, we require methods to establish an identity for a blank node based on a description by its neighborhood in the graph. This however is a nontrivial task because both the data values and the connections may undergo modifications in the subsequent version of the RDF graph. Finally, constructing an alignment between two RDF graphs presents a significant computational challenge: RDF graphs tend to be large, which quickly renders infeasible any method that attempts to perform pairwise comparison between *all* pairs of nodes of the two graphs.

We investigate a number of methods of aligning RDF graphs inspired by the classical notion of bisimulation for graphs. In essence, two nodes are bisimilar if they cannot be distinguished from each other by structural comparison of their outbound neighborhoods in which the nodes reachable from the bisimilar nodes are also bisimilar. What makes bisimulation particularly interesting is its computational properties: it is well-known that bisimulation can be computed in sub-quadratic time [13] but the basic partition refinement approach, while having quadratic worst time complexity scales well in practice [16], and we have chosen it as a basis of RDF alignment algorithms.

Example 1. Consider corrections in an evolving RDF graph presented in Figure 1 containing personal information of one of the authors of this paper. The first name is changed from the diminutive Sławek to its legal variant Sławomir, and an erroneous middle name Paweł is removed. Also, the URI

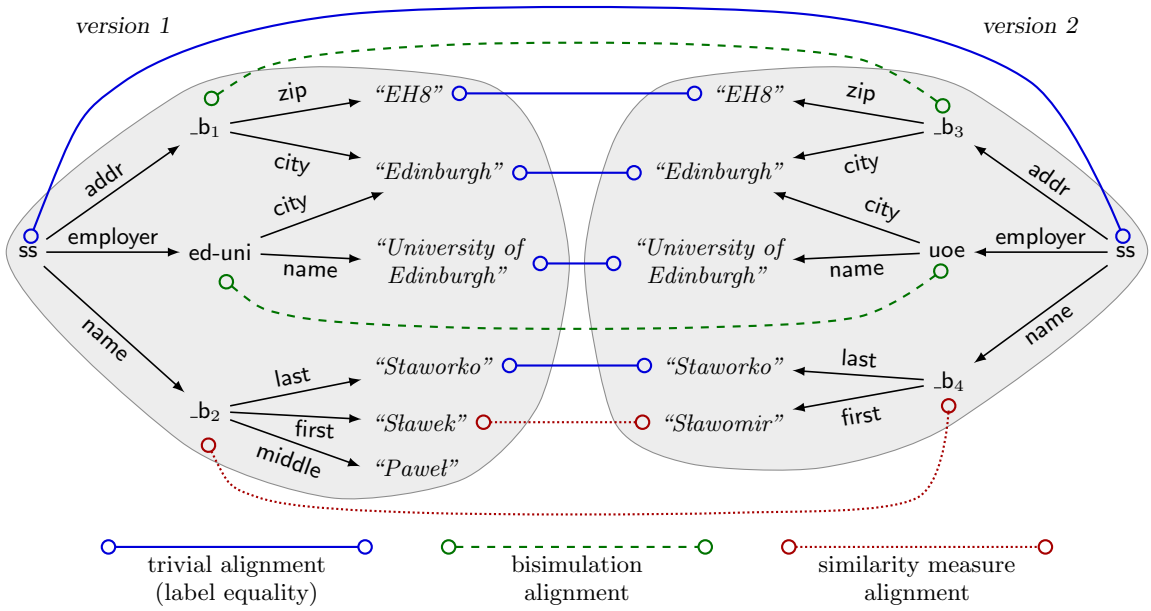

Figure 1: Alignment methods on two consecutive versions of an evolving RDF graph: uris are typeset in sanserif, *"literals"* are in italics surrounded by quotes, and blank nodes are _b.

representing the University of Edinburgh is changed from ed-uni to uoe. Note that a majority of literals and one URI, ss, can be trivially aligned by testing label equality. However, this simple method does not work for the address information even though it does not change. Here, address is structured as a record represented with a blank node and blank nodes are labeled with *local* identifiers that distinguish them only in a single version. Bisimulation aligns the blank nodes _b1 and _b3 because they represent a record with the same information structured in the same manner. Similarly, bisimulation aligns the nodes ed-uni and uoe. However, bisimulation requires strict similarity in the data values and the structure of the graph, and therefore, cannot handle edit changes in the data values and the structure. Consequently, bisimulation does not align the nodes _b2 and _b4 even though there is a significant evidence that they represent the same entity (the name of the same person.) □

Aligning nodes _b2 and _b4 (Figure 1) calls for similarity methods, and in we propose a natural similarity measure based on the string edit distance on literal nodes and the graph edit distance for non-literal nodes. While this method can align the nodes _b2 and _b4, it suffers from high complexity, which springs from the sizes of the input RDF graphs and the combinatorial nature of the edit distance problem: indeed, the lower complexity bound for the edit distance is quadratic for strings [19, 6] and cubic for graphs [7].

To overcome this obstacle, we investigate extending the bisimulation approach to account for such edits. While bisimulation defines a partition of nodes into clusters of indistinguishable nodes, we propose an approach that defines a *weighted* partition, where every node still belongs to exactly one cluster but is additionally attributed with a *confidence* value. Intuitively, the confidence value captures the distance of the node from the center of the cluster, which can be used to approximate the relative distance between two nodes in the same cluster. The limitation of the membership of a node to exactly one cluster has both positive and negative consequences. While it enables a scalable method for constructing weighted partitions, the weighted partition only approximates the goal similarity measure and the resulting alignment may be incomplete. Our experiments show, however, that the trade-off is generally positive: with a diligent application of a number of optimizations we obtain a relatively scalable method for RDF alignment that fails to align correctly relatively few pairs of nodes. We can also show that any pair aligned with this method is also aligned with the similarity method we wish to approximate.

The main contributions of the present paper are summarized as follows:

1. We formalize the problem of RDF graph alignment and present a methodology of aligning RDF graphs with partitions of the nodes.
2. We propose RDF alignment methods based on the standard notion of bisimulation for graphs that allow to handle blank nodes and changes in ontology (URI naming schemes).
3. We propose a natural measure of node similarity that yields an RDF alignment method robust under editing operations and extend the bisimulation approach to approximate the proposed similarity method.
4. We evaluate the accuracy and effectiveness of these methods on widely used databases presented in RDF.

Related Work. The similarity measure we define bears some resemblance to the similarity flooding approach [12] with an important difference on on how similarities are propagated: when defining the similarity of two nodes, the similarity flooding takes a weighted average over the Cartesian product of sets of outgoing edges of the two nodes while our approach identifies the optimal matching among the outgoing edges. We believe this approach to be more appropriate

in the context of evolving graphs and incorporates edit operations on the edges. Still, the inherent high complexity of both methods limits their scalability, and the aim of our research is development of scalable methods for identifying similar objects.

There is extensive work on entity resolution in the context of relational databases, however a comparison with that work is problematic. Because we are using RDF and because we are placing predicates on the same footing as other URIs the problem we are setting ourselves is equivalent to finding an alignment between two versions in which one (a) changes all the table names and column names and (b) changes all the key values. All that is kept are are the non-key data values the foreign key constraints.

There is also extensive literature on graph alignment [2]. Constructing an alignment between two graphs is virtually equivalent to constructing their *delta* [20], a description of changes occurring between the two graphs. The existing research [14] focuses mainly on on reporting high-level changes identifying ontology changes (`rdfs:type`) and compactly representing the delta whereas we treat RDF as a stand-alone data representation system and identify low-level changes occurring on the atomic level of nodes and their labels. The ability of identifying ontology changes may potentially allow both directions to reinforce each other. Handling blank nodes in the context of change detection is known to be very challenging (graph isomorphism) and a method of label-invention have been proposed [17]. This method works under the assumption that the blank nodes do not form cycles and can be seen as an adaptation of existing XML archiving techniques [1]. Our work generalises this method: we handle cycles, editing operations, and can even identify ontology changes.

Organization. The paper is organized as follows. In Section 2 we define basic notions. In Section 3 we and state the problem of RDF graph alignment and present a number of alignment methods based inspired by bisimulation. In Section 4 we present a natural measure for node similarity for RDF alignment and we propose its approximation based on an extension of bisimulation robust under editing operations. Section 5 contains experimental evaluation of the proposed methods. We present the conclusions of our study and discuss directions of future research in Section 6. Because of space restrictions we omit proofs, some formal definitions, and numerical values of our experiments; they can be found in the appendix of the complete version available at `http://homepages.inf.ed.ac.uk/sstawork/vldb16.pdf`.

2. PRELIMINARIES

In this section we define the data model for RDF graphs, formalize the notion of partition, and define the notion of bisimulation for RDF graphs.

2.1 Data model

RDF graphs are typically represented as sets of triples of URIs, literals, and blank nodes. Because we are dealing with two graphs that may contain the same URI we need a more general model that uses node identifiers and treats the URIs and literals as labels: we assume an enumerable set of node identifiers $\mathcal{N}$ and a set of labels $\mathcal{I} = \mathcal{U} \cup \mathcal{L} \cup \{\_\mathsf{b}\}$, which consists of URI labels $\mathcal{U}$, literal values $\mathcal{L}$, and a special *blank* value $\_\mathsf{b}$ used to label blank nodes. We assume that $\mathcal{U}$ and $\mathcal{L}$ are disjoint and neither contains $\_\mathsf{b}$.

Definition 1. A (*triple*) *graph* is a tuple $G = (N_G, E_G, \ell_G)$, where $N_G \subseteq \mathcal{N}$ is a finite set of nodes, $E_G \subseteq N_G \times N_G \times N_G$ is a set node triples (edges), and $\ell_G : N_G \to \mathcal{I}$ is a node labeling function. □

The URIs of G, $URIs(G)$ are those nodes $n \in G$ for which $\ell(n) \in \mathcal{U}$. $Literals(G)$ and $Blanks(G)$ are defined similarly.

We now define an *RDF graph* (e.g., one of the two versions we are trying to align) as a triple graph in which no two nodes have the same URI or literal label and the labels agree with the usual RDF conventions (literal labels only occur as objects and predicates cannot have blank labels.) Figure 2 shows an example of an RDF graph in which nodes are identified by their label and blank-labeled nodes are decorated with a subscript.

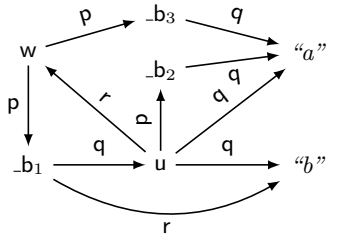

Figure 2: An RDF graph: URIs are typeset in sanserif, *"literals"* are in italics surrounded by quotes, and blank nodes are _b.

Using node identifiers that are independent of labels allows us to take two versions of the same RDF graph with possibly overlapping labels and combine them without confusing nodes with the same label. Graphs $G_1 = (N_1, E_1, \ell_1)$ and $G_2 = (N_2, E_2, \ell_2)$ are *disjoint* if $N_1 \cap N_2 = \emptyset$. Their *disjoint union* is $G_1 \uplus G_2 = (N_1 \cup N_2, E_1 \cup E_2, \ell_1 \cup \ell_2)$.

2.2 Partitions

We align two versions of an RDF graph using equivalence relations represented by a partitions of the node set of the combined graph. For our purposes we assign every node a unique color, and the equivalence classes of a partition are the sets of nodes with the same color.

Formally, we assume an enumerable set of *colors* $\mathcal{C}$, which allows both node labels and node identifiers to be used as colors as well as other structures that we can build from these. A *partition* of a graph G is a function $\lambda : N_G \to \mathcal{C}$ that assigns a color to every node of G. We use $\mathcal{C}^G$ for the set of all partitions of G. Throughout this paper, we only work with partitions of the same graph and we normally assume the graph to be known from the context. Note that the node labeling function ℓ_G is also a partition of G, which groups nodes by their labels, and in particular, places all blank nodes in the same equivalence class.

A partition λ defines an equivalence relation on the nodes of the graph, $R_\lambda = \{(n, m) \in N_G \times N_G \mid \lambda(n) = \lambda(m)\}$. A partition λ_1 is *finer* than a partition λ_2 if $R_{\lambda_1} \subseteq R_{\lambda_2}$. Two partitions λ_1 and λ_2 are *equivalent*, in symbols $\lambda_1 \equiv \lambda_2$, if $R_{\lambda_1} = R_{\lambda_2}$.

2.3 Bisimulation

Bisimulation is often defined on edge-labeled graphs. While RDF graphs are often drawn as such graphs with a triple

(s, p, o) represented as an edge (s, o) labeled with p, the label p is itself a node, and should participate in the bisimulation relation. We therefore adapt the definition of bisimulation to triple graphs by treating them as graphs in which the triple (s, p, o) is represented as an unlabeled edge connecting the node s to the *pair* (p, o) and define the *outbound neighborhood* of a node n in G is:

$$out_G(n) = \{(p, o) \mid (n, p, o) \in E_G\}.$$

Definition 2. A binary relation $R \subseteq N_G \times N_G$ is a *simulation* on a graph $G = (N_G, E_G, \ell_G)$ if for every $(n, m) \in R$ we have $\ell_G(n) = \ell_G(m)$ and for any $(p, o) \in out_G(n)$ there is $(p', o') \in out_G(m)$ such that $(p, p') \in R$ and $(o, o') \in R$. R is a *bisimulation* on G if both R and R^{-1} are simulations on G. Two nodes n and m of G are *bisimilar* if there is a bisimulation R on G such that $(n, m) \in R$. □

In the graph in Figure 2 the nodes _:b2 and _:b3 are bisimilar. Bisimulation identifies pairs of nodes that are indistinguishable by means of exploration of their outbound neighborhood, or intuitively, nodes having the same contents, as it is the case with the nodes _:b1 and _:b3 in the graph in Figure 1.

The identity relation on the nodes of a graph is always a bisimulation. If R_1 and R_2 are bisimulations on G, so is their union $R_1 \cup R_2$ is. Since all bisimulations on G are subsets of the finite Cartesian product $N_G \times N_G$, there exists a unique maximal bisimulation, $\mathsf{Bisim}(G)$, on G. The maximal bisimulation on a graph is an equivalence relation on nodes of the graph and thus defines a partition.

3. RDF GRAPH ALIGNMENT

Throughout most of the development we fix a single *combined* graph $G = (N_G, E_G, \ell_G)$, which (see Section 2.1) is the disjoint union of the source graph and target graph we want to align. Figure 3 shows such a union whose evolution can be described as replacing the equivalent blank nodes _:b2 and _:b3 with a single blank node _:b4 and renaming the URI u to v. While the blank node _:b1 has not been modified, in the second graph it has a different identifier _:b5.

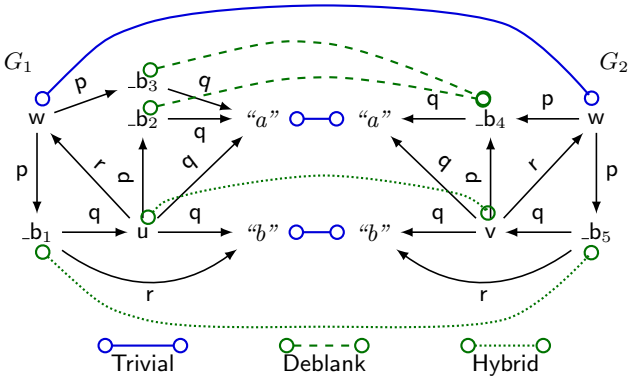

Figure 3: Progressive alignment of two RDF graphs.

3.1 Alignment by partition

Aligning two graphs consists of identifying pairs of corresponding nodes. We do not require, however, this to be a 1-to-1 correspondence, as a node of one graph may have a number of possible matches in the other. This allows us to model uncertainty of the correspondence between nodes; and even when the correspondence is free of uncertainty, we can represent redundancy in graphs such as the equivalent blank nodes _:b2 and _:b3 in the graph in Figure 3. Given a partition λ we can simply define an alignment of $G_1 = (N_1, E_1, \ell_1)$ and $G_2 = (N_2, E_2, \ell_2)$ as

$$Align(\lambda) = \{(n, m) \in N_1 \times N_2 \mid \lambda(n) = \lambda(m)\}.$$

Such alignments are precisely those binary relations that have the *crossover property*. An alignment A of G_1 and G_2 has this property if whenever $(n, m) \in A$, $(n, m') \in A$, and $(n', m) \in A$, then also $(n', m') \in A$.

An example of an alignment defined with a partition is the *trivial alignment* that uses label equality on non-blank nodes, which can be defined with the following partition of G ($n \in N_G$):

$$\lambda_{\mathsf{Trivial}}(n) = \begin{cases} \ell_G(n) & \text{if } n \text{ is a non-blank node,} \\ n & \text{if } n \text{ is a blank node.} \end{cases}$$

Indeed, $\lambda_{\mathsf{Trivial}}$ aligns only non-blank nodes with the same label as illustrated in Figure 3.

The alignment methods we propose in this paper work progressively. The *unaligned* nodes in N_1 are those which λ does not associate with a member of N_2: $Unaligned_1(\lambda) = \{n \in N_1 \mid \nexists m \in N_2 . \lambda(n) = \lambda(m)\}$. $Unaligned_2(\lambda)$ is defined similarly, and $Unaligned(\lambda) = Unaligned_1(\lambda) \cup Unaligned_2(\lambda)$.

3.2 Partition refinement

As a first step we employ *partition refinement* technique to improve on trivial alignment with bisimulation.

Definition 3. A (*one-step*) *partition refinement* is a function Λ that maps one partition of G to another partition of G such that $\Lambda(\lambda)$ is finer than λ and $\Lambda(\lambda_1) \equiv \Lambda(\lambda_2)$ whenever $\lambda_1 \equiv \lambda_2$. □

The first condition is natural and requires the process to be monotone; the second condition allows only those refinement functions that are independent of the representation of the partition. The refinement function is applied iteratively to a given initial partition until the process stabilizes i.e., further applications of the function yield an equivalent partition. Taken together, these two conditions guarantee termination.

Definition 4. The *refinement* $\Lambda^*(\lambda)$ of λ w.r.t. Λ is $\Lambda^n(\lambda)$ where n is minimal such that $\Lambda^n(\lambda) \equiv \Lambda^{n+1}(\lambda)$ □

$\Lambda^*(\lambda)$ is – to within recoloring – a fixpoint Λ on λ. We incorporate bisimulation by coloring a node with the combined colors of its outbound node pairs:

$$recolor_\lambda(n) = (\lambda(n), \{(\lambda(p), \lambda(o)) \mid (p, o) \in out_G(n)\}), \quad (1)$$

where λ is the current partition. The inclusion of the original color of n is to ensure that the procedure yields progressively finer partitions. We use this function on a selected subset of nodes without changing the color of the other nodes. Formally, the (one step) *bisimulation partition refinement* $BisimRefine_X(\lambda)$ on $X \subseteq N_G$ is the partition defined as

$$\lambda'(n) = \begin{cases} recolor_\lambda(n) & \text{if } n \in X, \\ \lambda(n) & \text{otherwise.} \end{cases} \quad (2)$$

This partition refinement captures bisimulation when applied to all nodes with the node labeling function ℓ_G defining the initial partition.

Proposition 1 *For any graph $G = (N_G, E_G, \ell_G)$, the partition $\lambda_{\mathsf{Bisim}} = BisimRefine^*_{N_G}(\ell_G)$ captures the maximal bisimulation on G i.e., $Align(\lambda_{\mathsf{Bisim}}) = \mathsf{Bisim}(G)$.*

The color assigned to a node is essentially a derivation tree rooted at the node, and because of the recursive nature of the bisimulation process, it can be compactly presented as a DAG and implemented with a simple hashing technique.

Example 2. Figure 4 shows the fixpoint computation of λ_{Bisim} on the graph in Figure 2, where colors are presented using derivation trees. Note that a node with no outgoing edges, in particular a literal or a URI used solely as a predicate, essentially maintains the same color through all iterations of the process. For clarity we use only the original color of such nodes and illustrate the refinement process only on nodes whose color changes. We use (derivation)

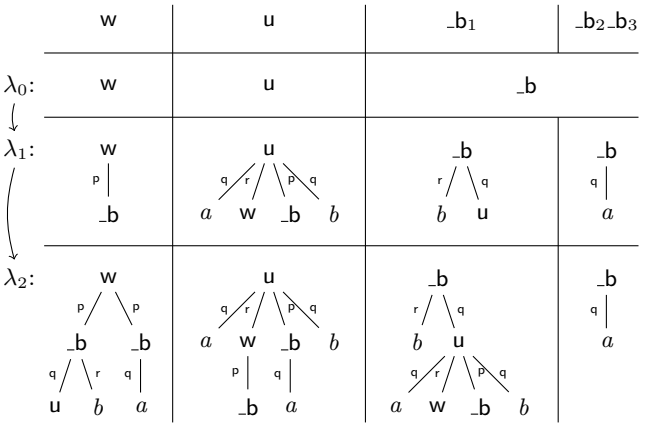

Figure 4: Fixpoint computation of bisimulation partition on RDF graph from Figure 2.

trees to represent the colors and point out that every iteration essentially unfolds by one level the tree from the previous iteration. Depending on the node the derivation tree is rooted at the unfolding may yield different results, and consequently, different derivation trees may be assigned to nodes that previously had the same derivation tree. For instance, the nodes $\_\mathsf{b}_1$, $\_\mathsf{b}_2$, and $\_\mathsf{b}_3$ all have initially (λ_0) the same tree, but after the first iteration they are split into two separate classes. Since the partition λ_2 is the same as the previous partition λ_1, the end result is λ_1. □

3.3 Deblanking alignment

Deblanking alignment improves on trivial alignment by using bisimulation on blank nodes:

$$\lambda_{\mathsf{Deblank}} = BisimRefine^*_{Blanks(G)}(\ell_G).$$

Intuitively, the bisimulation partition refinement assigns to every blank node a color that characterizes it by its contents (the URIs and data values reachable from the node). Two blank nodes are aligned if they have the same contents. The deblanking partition defines an equivalence relation that is similar to bisimulation and captures the essence of the deblanking process (described in the full version).

Example 3. Figure 3 shows the final colors of blank nodes of the graphs in Figure 3 obtained during the iterative refinement of deblanking alignment. Derivation trees are used for colors; in the case of deblanking alignment the trees are unfolded only at blank nodes. The unfolding halts at URIs and literals, and in particular, the derivation trees of URIs and literals consist of a root node alone. The colors of the blank nodes are as follows:

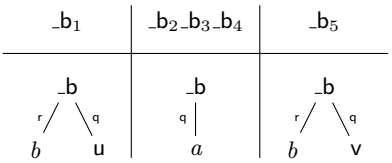

Figure 5: Colors of blank nodes in Deblank.

As a result both the nodes $\_\mathsf{b}_2$ and $\_\mathsf{b}_3$ are aligned to $\_\mathsf{b}_4$. On the other hand, the node $\_\mathsf{b}_1$ is not aligned to $\_\mathsf{b}_5$ because their colors differ. □

The use of *BisimRefine* determines the identity of a blank node solely on its contents i.e., the identity of nodes reachable with the outgoing edges. In general, however, the proposed framework could easily accommodate approaches that consider the incoming edges or only a selected subset of edges, such as those determined by the type of a node.

3.4 Hybrid alignment

Hybrid alignment improves on deblanking alignment by applying bisimulation to unaligned URI nodes. Deblanking alignment colors those nodes with their URI label; however, the bisimulation refinement function incorporates this label in every iteration, and consequently, an unaligned URI cannot be aligned in this fashion to another unaligned node with a different URI label. We also note that aligning URI nodes with different labels could permit aligning previously unaligned blank nodes whose color in the deblanking alignment might incorporate the different URI labels. Therefore, we begin by modifying the deblanking partition by resetting the color of unaligned URI and blank nodes to the neutral blank color: essentially, we place all unaligned non-literal nodes in the same cluster and then use bisimulation refinement to define their identity. Formally, for a set of nodes $X \subseteq N_G$ we define an auxiliary function that blanks their colors in the given partition: $Blank(\lambda, X) = \lambda'$, where

$$\lambda'(n) = \begin{cases} \_\mathsf{b} & \text{if } n \in X, \\ \lambda(n) & \text{otherwise.} \end{cases} \tag{3}$$

We also identify unaligned non-literal nodes:

$$UN(\lambda) = Unaligned(\lambda) \setminus Literals(G) \tag{4}$$

We define the hybrid partitioning as follows:

$$\lambda_{\mathsf{Hybrid}} = BisimRefine^*_{UN(\lambda_{\mathsf{Deblank}})}(Blank(\lambda_{\mathsf{Deblank}}, UN(\lambda_{\mathsf{Deblank}}))).$$

Using $\lambda_{\mathsf{Trivial}}$ instead of $\lambda_{\mathsf{Deblank}}$ above yields the same result.

Example 4. In Figure 4 we present the final colors of unaligned nodes (by Deblank) of the graphs in Figure 3 obtained during the iterative refinement procedure of the hybrid alignment. Again, we represent the colors as trees but note that technically speaking they are no longer derivation trees because for unaligned nodes we use the blank label rather than the label of the node (cf. colors of u and v). Also,

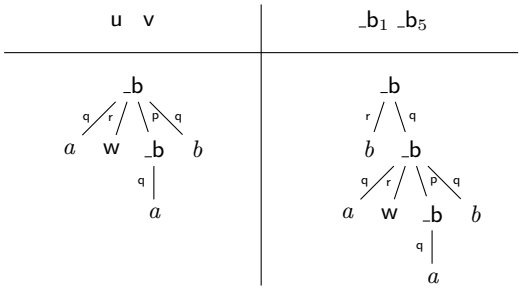

Figure 6: Colors of selected nodes in Hybrid.

the depth of the trees may be greater than the number of iterations of the refinement process because for aligned nodes colors from the deblanking alignments are used, as it is the case with the colors of nodes $\_b_2$, $\_b_3$, and $\_b_4$. Naturally, the final colors of nodes u and v coincide and therefore these two nodes are aligned by Hybrid. Similarly, Hybrid aligns the blank nodes $\_b_1$ and $\_b_5$. □

Finally, we point out that because the constructed alignments have been defined by improving one on another, the corresponding alignments create a proper hierarchy:

$$Align(\lambda_{\mathsf{Trivial}}) \subseteq Align(\lambda_{\mathsf{Deblank}}) \subseteq Align(\lambda_{\mathsf{Hybrid}}).$$

4. SIMILARITY ALIGNMENT

In this section we outline a method of further refining the bisimulation-based Hybrid alignment with pairs of similar nodes as identified with a distance function. More precisely, we define two distance functions on nodes: σ_{Edit}, which defines an alignment robust under editing operations but is computationally expensive, and $\sigma_{\mathsf{Overlap}}$, which approximates σ_{Edit} and scales well in practice. We continue to work with a single combined graph $G = (N_G, E_G, \ell_G)$ that is the disjoint union of the source graph $G_1 = (N_1, E_1, \ell_1)$ and the target graph $G_2 = (N_2, E_2, \ell_2)$.

4.1 Node distance functions

We investigate a natural manner of aligning nodes using the standard notion of a distance function $\sigma : N_1 \times N_2 \to [0, 1]$ in which similar nodes have a low value of σ. Although we do not require σ to satisfy the triangle equality, we will only work with metrics as they are sometimes used to represent distances in a graph. When we wish to combine (add) distance values, in order that the result is again in $[0, 1]$, we use an infix addition operator $\oplus : [0, 1] \times [0, 1] \to [0, 1]$. This operator can have a number of natural definitions, the only requirement being compatibility with the triangle inequality: $\sigma(n, z) \oplus \sigma(z, m) \leq \sigma(n, m)$ for any nodes n, m, z. We shall use a rudimentary definition of this operator: $x \oplus y = \min\{x + y, 1\}$ for $x, y \in [0, 1]$.

The alignment defined by a node distance function σ is additionally parameterized by a threshold value $\theta \in [0, 1]$:

$$Align_\theta(\sigma) = \{(n, m) \in N_1 \times N_2 \mid \sigma(n, m) \leq \theta\}.$$

Alignments captured with distance functions do not necessarily have the cross-over property, and are more expressive than alignments captured with partitions; but for every partition there exists a distance function and threshold that defines the same alignment.

4.2 Edit distance alignment

We define a natural node distance function σ_{Edit} that attempts to address two important aspects of evolving RDF data sets: 1) editing changes in the literal values, 2) editing changes in the structure of the graph. In essence, σ_{Edit} attempts to refine the Hybrid alignment by incorporating string edit distance on literal values and graph edit distance on non-literal nodes while iteratively propagating the distances throughout the graph. We omit its formal definition and illustrate its workings on an example below.

Example 5. Consider the two RDF graphs G_1 and G_2 presented in Figure 7, where we present the distance between pairs of nodes. For clarity, we indicate the distances between closest pairs of nodes. Because σ_{Edit} refines the Hybrid

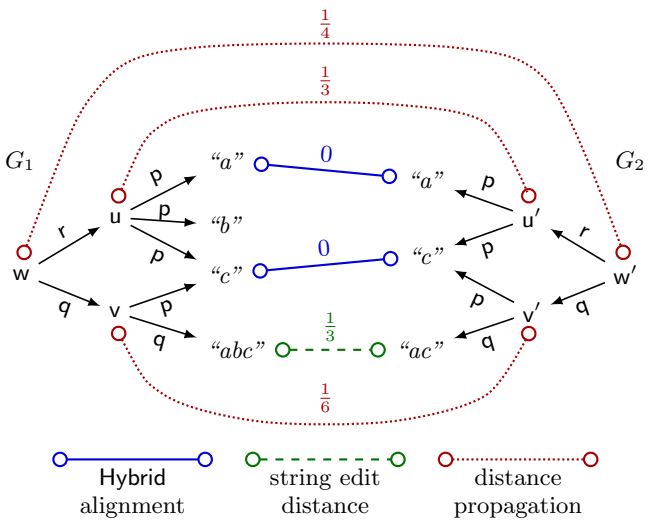

Figure 7: Alignment with the distance function σ_{Edit}.

alignment, the distance between any pair of aligned nodes aligned by the Hybrid partition is 0, as it is the case with the trivially aligned literal nodes *"c"* or with the trivially aligned URIs used as edge labels. On pairs of unaligned literal nodes we use a string edit distance. For instance, the distance between the nodes *"abc"* and *"ac"* is $\frac{1}{3}$ because they differ by the presence of b and the length of both is bounded by 3. Note that σ_{Edit} is 1 on any other pair that involves at least one node aligned by the Hybrid partitioning. For example, the distance between *"a"* and *"ac"* is 1 even though the normalized edit distance is $\frac{1}{2}$.

On unaligned non-literal nodes we propagate the distance established on other nodes. For example, the distance between v and v′ is the average of the distances between the pairs of nodes on the edges with corresponding labels. Because a node may have more than one edge with a given label, this process consists of finding a matching that maximizes this average. Furthermore, when the numbers of edges with a given label are different the matching accounts for the differences in a manner consistent with graph edit distance. The distance between u and u′ is $\frac{1}{3}$ because the main difference is in the presence of the outgoing edge to a node labeled with *"b"* and the size of their outbound neighborhood is bounded by 3. Finally, an optimal matching is found using the Hungarian algorithm [9]. □

The main obstacle to the practical use of σ_{Edit} is the significant computational cost of constructing σ_{Edit}: we need to materialize a matrix whose size is quadratic in the size of the input graphs, which makes this approach scale poorly. Furthermore, lower bounds on computing edit distance on strings [19] and graphs [7] suggest that the limitations on practical use of σ_{Edit} may be fundamental. These observations motivate us to investigate a heuristic approach that approximates σ_{Edit} and has better computational properties.

4.3 Weighted partitions

We begin with a simple intuition: for aligning nodes with σ_{Edit} we do not necessarily need to know the distance between every pair of nodes but only wish to find pairs of nodes that are close to each other. In the context of alignment of evolving RDF, it is natural to expect the number of such pairs to be relatively low and more manageable. Ideally, the alignment is a one-to-one correspondence between the source and the target nodes, which translates to a linear number of pairs of close nodes. We contrast it with the unlikely case of a complete bipartite alignment, where every source node is aligned to every target node, which yields a quadratic number of pairs of close nodes and has space requirements on a par with materializing σ_{Edit}. Encouraged by the very good performance of the basic refinement algorithm, we investigate a generalization of bisimulation geared towards clustering nodes that are in close proximity.

We begin by generalizing partitions by assigning to every node of a cluster the distance from the center of the cluster. By using the triangle inequality, the distance from the center allows us to estimate the relative distance between any pair of nodes in the same cluster. Formally, a *weighted partition* of a graph G is a pair $\xi = (\lambda, \omega)$, where $\lambda : N_G \to \mathcal{C}$ is a partition of G and $\omega : N_G \to [0, 1]$ is a *weight* function. A weighted partition $\xi = (\lambda, \omega)$ defines the following distance function (for $n, m \in N_G$)

$$\sigma_\xi(n, m) = \begin{cases} \omega(n) \oplus \omega(m) & \text{if } \lambda(n) = \lambda(m), \\ 1 & \text{otherwise.} \end{cases} \tag{5}$$

Naturally, the alignment defined by the weighted partition ξ w.r.t. the threshold value $\theta \in [0, 1]$ is

$$\mathit{Align}_\theta(\xi) = \{(n, m) \in N_1 \times N_2 \mid \lambda(n) = \lambda(m),\ \omega(n) \oplus \omega(m) < \theta\}.$$

We propose a method for constructing a weighted partition that approximates σ_{Edit} and thus produces an alignment that approximates $\mathit{Align}_\theta(\sigma_{\mathsf{Edit}})$.

Example 6. Figure 8 presents a weighted partition of the graph from Figure 7 that captures the essence of the alignment defined with σ_{Edit}. For instance, the distance between the nodes *"abc"* and *"ac"* is $\frac{2}{9} \oplus \frac{1}{9} = \frac{1}{3}$ and the distance between nodes the nodes w and w' is $\frac{2}{9} \oplus \frac{1}{36} = \frac{1}{4}$. However, the two node distance functions are not equal: for the nodes u and v' the weighted partition defines distance 1 because those nodes are in different clusters while σ_{Edit} gives this pair a lower value of $\frac{1}{3}$ because one outgoing edge can be matched. □

We view the weight function ω of a weighted partition $\xi = (\lambda, \omega)$ only as a measure of uncertainty that a given node n has been correctly assigned to its the cluster (labeled) $\lambda(n)$. The node n belongs to the cluster $\lambda(n)$ even for the extreme weight value $\omega(n) = 1$ and the weight value is only taken

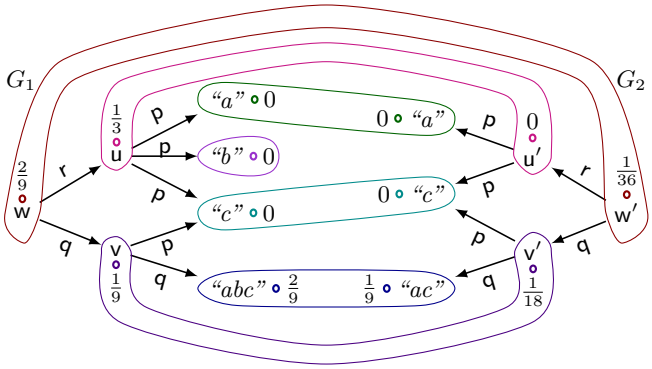

Figure 8: Weighted partition approximating σ_{Edit}.

under consideration during the construction of the alignment. While the precise threshold value θ identifies the sets of nodes unaligned by $\mathit{Align}_\theta(\xi)$, for simplicity of notations we use the same definitions of unaligned nodes as for standard partitions: a node of one graph is unaligned if it belongs to a class with no nodes of the opposite graph. We remark that our methods can be easily extended to incorporate the threshold value in identifying unaligned nodes.

4.4 Enrichment

Our approach can be described as a simple iterative procedure: start with an initial weighted partition and while there exist previously unaligned but close and easily identifiable pairs of nodes, enrich the partition correspondingly and propagate this information to other unaligned nodes. The exact method used to identify pairs of close nodes typically depends on the precise nature of data and later on we propose one generic method. Here, we present a general approach of enriching a given weighted partition with newly discovered pairs of close nodes.

We assume a weighted partition $\xi = (\lambda, \omega)$ of G, and the newly discovered pairs of close nodes are given in the form of a weighted bipartite graph $H = (A, B, M, d)$, where $A \subseteq \mathit{Unaligned}_1(\xi)$ is a subset of unaligned source nodes, $B \subseteq \mathit{Unaligned}_2(\xi)$ is a subset of unaligned target nodes, $M \subseteq A \times B$ is the set of newly discovered close pairs of nodes, and $d : M \to [0, 1]$ is a distance function on those pairs of nodes. We view H as an undirected graph and w.l.o.g. assume that no node in H is isolated i.e., every node in H is connected to at least one node (isolated nodes can be removed from consideration). For two arbitrary nodes v and w of H, we use $d^*(v, w)$ the length of the shortest path connecting v and w calculated using $\oplus$, and 1 if v and w are not connected. A number of ways of enriching ξ with H can be envisioned, and we use a rather simple one because in practice H will have a very sparse structure (close to a one-to-one correspondence).

In the first step we decompose H into a maximal set of disjoint connected components $\mathcal{X} = \{X_1, \ldots, X_k\}$, where two nodes belong to the same component if and only if they are connected. Because we work with H with no isolated nodes, every component X_i contains both nodes from A and B. To incorporate these components into the weighted partition we need to assign to every element of each component a weight value that is consistent with the distances in H i.e., we need to define a function $w : \bigcup \mathcal{X} \to [0, 1]$ such that

for any X_i, any $a \in A \cap X_i$, and any $b \in B \cap X_i$ we have $d^*(a,b) \leq w(a) \oplus w(b)$. We propose a simple approach where for every source node in a component we take the half of the maximum distance to any target node in the same component and *vice versa*. Now, the *enrichment* of ξ by H is a weighted coloring $\mathsf{Enrich}(\xi, H) = (\lambda', \omega')$, where

$$\lambda'(n) = \begin{cases} X_i & \text{if } n \in A \cup B \text{ and } n \in X_i, \\ \lambda(n) & \text{otherwise} \end{cases}$$

$$\omega'(n) = \begin{cases} w(n) & \text{if } n \in A \cup B, \\ \omega(n) & \text{otherwise.} \end{cases}$$

4.5 Propagation

Once the newly discovered pairs of close nodes have been incorporated into the weighted coloring, we propagate this new information in a manner inspired by the coloring refinement procedure that allows to identify further close nodes.

The weight of the new color will be an average of the weights of the colors of outbound nodes that constitute the new color:

$$\mathit{reweight}_\omega(n) = \bigoplus \left\{ \frac{\omega(p) \oplus \omega(o)}{|\mathit{out}_G(n)|} \,\middle|\, (p,o) \in \mathit{out}_G(n) \right\},$$

where ω is the current weight function of a weighted partition. This function is defined only for nodes having one or more outgoing edges; for a node $n \in N_G$ with no outgoing edges we set $\mathit{reweight}_\omega(n) = \omega(n)$. Analogously to the refinement procedure, we recolor only a selected subset of (previously unaligned) nodes. We define one-step refinement of a weighted coloring $\xi = (\lambda, \omega)$ on a set of nodes $X \subseteq N_G$ as $\mathit{BisimRefine}_X(\xi) = (\lambda', \omega')$ where λ' is defined as for non-weighted partitions in (2), and

$$\omega'(n) = \begin{cases} \mathit{reweight}_\omega(n), & \text{if } n \in X, \\ \omega(n) & \text{otherwise.} \end{cases}$$

We shall apply this refinement operation iteratively until the partition no longer changes and the weights stabilize i.e., the weight assigned to any node changes by less than some fixed small value $\epsilon > 0$. The property that ensures stabilization is that the initial weights of the nodes in X will all be 0, and will only increase during the refinement process We use $\mathit{BisimRefine}^*_X(\xi)$ to denote the weighted partition obtained with sufficient iterations of *BisimRefine* to ensure a fix-point partition (cf. Definition 4) and stabilization of the weight value function. The exact definition is presented in appendix of the complete version of the paper.

Because we use propagation extensively we introduce a convenient shorthand $\mathsf{Propagate}(\xi)$ that propagates the alignment information in a weighted partition $\xi = (\lambda, \omega)$ to unaligned nodes. The set of unaligned non-literal nodes $\mathit{UN}(\xi)$ is defined as for non-weighted partitions (4) and we extend the blank-out operation to weighted partitions: for a set of (unaligned) nodes $X \subseteq N_G$ let $\mathit{Blank}(\xi, X) = (\lambda', \omega')$, where λ' is defined as for non-weighted partitions (3) and (for $n \in N_G$)

$$\omega'(n) = \begin{cases} 0 & \text{if } n \in X, \\ \omega(n) & \text{otherwise.} \end{cases}$$

Finally, we define

$$\mathsf{Propagate}(\xi) = \mathit{BisimRefine}^*_{\mathit{UN}(\xi)}(\mathit{Blank}(\xi, \mathit{UN}(\xi))).$$

There is a natural relationship between the propagation and the hybrid partition obtained with the coloring refinement algorithm: $\mathsf{Propagate}((\lambda_{\mathsf{Trivial}}, \mathbb{0})) = \mathsf{Propagate}((\lambda_{\mathsf{Deblank}}, \mathbb{0})) = (\lambda_{\mathsf{Hybrid}}, \mathbb{0})$, where $\mathbb{0}$ is a constant weight function that assigns 0 to every node.

4.6 Overlap heuristic

Similar nodes by are identified with a heuristic based on the overlap measure combined with inverted indexes and identification of least frequent elements as outlined on Algorithm 1. Recall that the overlap measure between two sets of objects O_1 and O_2 is defined as the fraction of elements in common over the number of all elements:

$$\mathit{overlap}(O_1, O_2) = \frac{|O_1 \cap O_2|}{|O_1 \cup O_2|},$$

This similarity measure has a natural distance counterpart that measures the fraction of elements present in exactly one of the sets ($\div$ is the symmetric difference operator):

$$\mathit{diff}(O_1, O_2) = \frac{|O_1 \div O_2|}{|O_1 \cup O_2|} = 1 - \mathit{overlap}(O_1, O_2),$$

Note that $\mathit{diff}(X, X) = 0$ but since the above formula is valid only if one of the sets O_1 and O_2 is nonempty, we set $\mathit{diff}(\emptyset, \emptyset) = 0$ and $\mathit{overlap}(\emptyset, \emptyset) = 1$.

Algorithm 1 Overlap heuristic

function OverlapMatch($A, B, \theta, \mathit{char}, \mathit{dist}$)
Input: A, B – two (disjoint) sets of nodes
$\theta \in [0,1]$ – similarity threshold value
$\mathit{char} : A \cup B \to \mathcal{P}(\mathcal{O})$ – node characterizing function
$\sigma : A \times B \to [0,1]$ – similarity measure
Output: (A, B, M, w) – weighted bipartite graph

```
1:  O := ⋃{char(n) | n ∈ B}
2:  Inv := hashtable()
3:  freq := hashtable()
4:  for o ∈ O do
5:      Inv[o] := {n ∈ B | o ∈ char(n)}
6:      freq[o] := |Inv[o]|
7:  M := ∅
8:  w := hashtable()
9:  for n ∈ A do
10:     C := ∅
11:     (o_1, ..., o_k) := sort(char(n), freq)    //freq[o_i] ≤ freq[o_{i+1}]
12:     for i ∈ {1, ..., ⌈k * θ⌉} do
13:         for m ∈ Inv[o_i] do
14:             if overlap(char(n), char(m)) ≥ θ then
15:                 C := C ∪ {m}
16:     for m ∈ C do
17:         if σ(n, m) < θ then
18:             M := M ∪ {(n, m)}
19:             w(n, m) := σ(n, m)
20: return (A, B, M, w)
```

Our approach (Algorithm 1) identifies candidate pairs of nodes by representing a node n with a set of objects $\mathit{char}(n)$ that characterize n in a manner that exhibits a high coincidence between $\sigma^{\mathit{dist}}(n,m) < \theta$ and $\mathit{diff}(\mathit{char}(n), \mathit{char}(m)) < \theta$. Intuitively, the more similar two nodes are the more objects they have in common. We use inverted indexes to identify pairs of nodes that have the same object in common and we use frequency counts to use the less frequent,

and thus more discriminating, objects when identifying the set C of potentially close nodes. Additionally, we use the threshold value θ to inspect only a fraction of all objects $char(n)$ characterizing the node n since this fraction must contain objects of any node m that has overlap above θ. Every candidate pair is then tested with a distance function σ that filters out the wrong candidates (this function needs not have the same definition as σ^{dist}).

4.7 Overlap alignment

We use the overlap heuristic to construct a weighted partition ξ_{Overlap} (Algorithm 2) and the corresponding *overlap* alignment. It defines a distance measure $\sigma_{\mathsf{Overlap}}$ that for the purposes of alignment closely captures the edit distance σ_{Edit}. First, literal nodes are characterized with the function *split* that takes the label of the literal node and splits it into a set of words and the similarity measure $\sigma_{Literals}$ is defined in the same way as σ_{Dist} on literal nodes. Then, for a given weighted partition $\xi = (\lambda, \omega)$, non-literal nodes are characterized with the set of colors of their outgoing edges

$$out\text{-}color_\xi(n) = \{(\lambda(p), \lambda(o)) \mid (p, o) \in out_G(n)\}.$$

The distance function on non-literals σ^ξ_{NL} is defined in a manner that captures the weight of the optimal (Hungarian algorithm) matching among the outgoing edges of two nodes given that only the weight function is at our disposal. For $n \in N_1$ and $m \in N_2$ the value of $\sigma^{NL}_\xi(n, m)$ is defined as

$$\bigoplus \left\{ \frac{\sigma_\xi(p_1, p_2) \oplus \sigma_\xi(o_1, o_2)}{f} \,\middle|\, ((p_1, o_1), (p_2, o_2)) \in M \right\} \oplus \frac{R}{f},$$

where σ_ξ is the distance on nodes induced by ξ as defined in (5), $f = \max\{|out\text{-}color_\xi(n)|, |out\text{-}color_\xi(m)|\}$, M is a binary relation coupling the outgoing edges of n and m with the same color and having the same position in the list of outgoing edges with the same colors ordered by their weight, and R is the number of outgoing edges of n and m that are not coupled in M (when one node has more outgoing edges of a given color than the other node). Interestingly, computing M and R can be easily done without the use of the Hungarian algorithm. The overlap heuristic is applied iteratively until no further close pair of non-literal nodes can be found.

Algorithm 2 Overlap weighted partition

function $\mathsf{Overlap}(G, \theta)$
Input: $G = G_1 \uplus G_2$ – combined graph
Parameter: $\theta \in [0, 1]$ – similarity threshold value
1: $\xi_0 := (\lambda_{\mathsf{Hybrid}}, \mathbb{0})$
2: $A_0 := Unaligned_1(\xi_0) \cap Literals(G_1)$
3: $B_0 := Unaligned_2(\xi_0) \cap Literals(G_2)$
4: $H_0 := \mathsf{OverlapMatch}(A_0, B_0, \theta, split, \sigma_{Literals})$
5: $i := 0$
6: **do**
7: $\quad i := i + 1$
8: $\quad \xi_i := \mathsf{Propagate}(\mathsf{Enrich}(\xi_{i-1}, H_{i-1}))$
9: $\quad A_i := Unaligned_1(\xi_i) \setminus Literals(G_1)$
10: $\quad B_i := Unaligned_2(\xi_i) \setminus Literals(G_2)$
11: $\quad H_i := \mathsf{OverlapMatch}(A_i, B_i, \theta, out\text{-}color_{\xi_i}, \sigma^{NL}_{\xi_i})$
12: **until** H_i has no edges
13: **return** ξ_i

The fundamental result is that the overlap alignment only aligns pairs of nodes that are similar and is stated below.

Theorem 1 *Let $\xi_{\mathsf{Overlap}} = (\lambda_{\mathsf{Overlap}}, \omega_{\mathsf{Overlap}})$ be the overlap alignment of $G_1 = (N_1, E_1, \ell_1)$ and $G_2 = (N_2, E_2, \ell_2)$. Then, for any $n \in N_1$ and $m \in N_2$, if $\lambda_{\mathsf{Overlap}}(n) = \lambda_{\mathsf{Overlap}}(m)$, then $\sigma_{\mathsf{Edit}}(n, m) \leq \omega_{\mathsf{Overlap}}(n) * \omega_{\mathsf{Overlap}}(m)$.*

5. EXPERIMENTAL RESULTS

In this section we report on experimental evaluation of the proposed solutions on two practical data sets: EFO – Experimental Factor Ontology [11] supported by the European Bioinformatics Institute and GtoPdb – The Guide to Pharmacology database [4] supported by the International Union of Pharmacologists and the British Pharmacological Society. A brief word about the two databases. EFO provides a systematic description of many experimental variables available in other databases and combines parts of several biological ontologies. It is expressed in OWL, which is in turn reasonably directly expressed in RDF. GtoPdb is a relational database that contains curated information from hundreds of experts about drugs in clinical use and some experimental drugs, together with information on the cellular targets of the drugs and their mechanisms of action in the body. We converted GtoPdb into RDF using a standard (W3C recommended) approach [18].

Both databases are evolving; new versions are released every few months. Both databases are usually viewed through a Web interface and despite their internal representions have a similar general nature consisting of classification hierarchies along with a rich annotation. However their representation in RDF is very different. For example, in EFO the notion of a subclass is directly represented, while in GtoPdb it is inferred from the relational database.

5.1 Experimental Factor Ontology (EFO)

In Figure 9 we present node and edge counts of ten versions of the Experimental Factor Ontology (versions 2.34 through 2.44; 2.40 not accessible). We point out that lit-

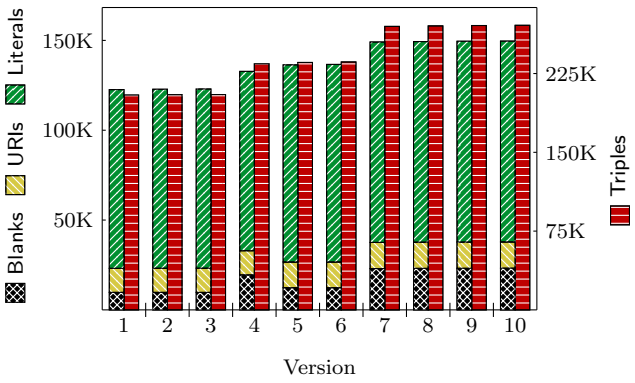

Figure 9: EFO dataset versions.

erals comprise over 75% of the contents of every version. While the number of URIs is generally proportional to the total number of nodes (approx. 10%), the number of blank nodes fluctuates quite significantly 7–15%. After a closer inspection we found that the fluctuations are due to duplication (bisimilar blank nodes) and normalized counts of blank nodes do not fluctuate but grow steadily.

We analyzed the alignments obtained with the presented methods between any pair of versions of EFO. We also measured the number of aligned triples – the results are virtually

the same if we measure the number of aligned nodes. For the trivial and deblanking alignment in Figure 10 we report the ratio of the number of aligned edges to the total number of edges in both graphs (edges using precisely the same identifiers are counted precisely once). The diagonal of the

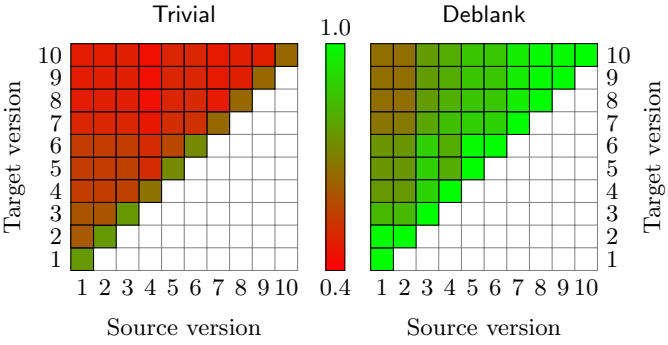

Figure 10: Trivial and Deblank alignments (EFO).

matrix is the result of self-alignment, the alignment of a version with itself, and ideally we wish it to be a complete alignment with ratio equal to 1, as it is for the deblanking alignment. The ratios for trivial alignment are significantly worse because of the impact of blank nodes that are not aligned. Overall, we observe an expected descending gradient from the diagonal towards the upper left point of the matrix, except for version 3 due to fluctuations in the number of blanks. This gradient has a natural explanation: the further apart the two aligned versions are, the more significant changes they have undergone, and consequently, less edges can be aligned.

The relative improvement offered by the hybrid and overlap alignments is subtle, and to highlight it in Figure 11 we show the absolute number of edges that are additionally aligned by the hybrid alignment (compared with the deblanking alignment) and the overlap alignment (compared with the hybrid alignment). In both cases, the improve-

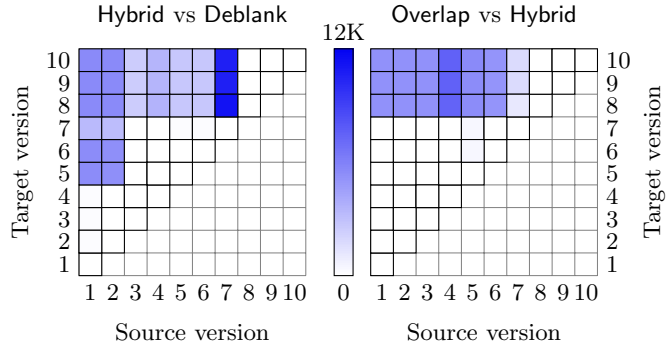

Figure 11: Hybrid and Overlap alignments (EFO).

ments come mainly from ontology changes manifested by change of URI prefix e.g., `http://purl.org/obo/owl/` to `http://purl.obolibrary.org/obo/`. This process can be quite straightforward e.g., a large number of URIs using old prefix in version 7 is replaced by URIs with new prefix in version 8. This change also involves changes in the contents of the affected nodes, which are captured with the overlap alignment. Ontology change may take more time with URIs disappearing in between: a number of URIs using the old prefix in the first two versions are removed in version 3, and then reappear in version 5 with the new prefix.

Because our methods focus on the outgoing neighborhood of a node, they make errors by incorrectly aligning URIs that are used as predicates only: these URIs typically are present as subject in one triple that declares the type of the URI (and uses `rdf:type` as predicate). The number of such incorrectly aligned predicates is relatively small (< 15). A better solution would identify URIs that are predominantly used as predicates and use a different refinement process, for instance, one that incorporates the colors of the subject and the object in any triple that uses the given predicate.

Finally, we found the quality of the hybrid and overlap alignments to be overall satisfying: very few URIs undergoing changes are missed and no URIs are aligned in error. Unfortunately we cannot precisely evaluated it because we lack the appropriate ground truth for the EFO dataset and we present a more detailed discussion in the appendix of the complete paper. In the following subsection, we run experiments on a dataset for which the ground truth is easily obtained.

5.2 Guide to Pharmacology database (GtoPdb)

We used 10 versions of the GtoPdb relational database, which we exported to RDF following the W3C Direct Mapping recommendation [18] using the D2RQ platform. The mapping works as follows: 1) every tuple is identified by a URI which is constituted from a given URI prefix, the table name table, and the attribute values of the primary key, 2) value (non-referential) attributes are translated to triples consisting of the tuple URI, the attribute name and a literal for the attribute value, 3) referential attributes are translated to triples pointing to the URI of the referred tuple. While this experimental setting has been designed to evaluate the hybrid and overlap alignments, we believe it captures a common situation in which a relational database is exported to RDF at different times by different services using similar export schemes (e.g., the default W3C Direct Mapping configuration).

To focus our study on the hybrid and overlap alignments, we export every version with a different URI prefix. Because there are no common URIs and no blank nodes, the trivial and deblanking alignments align no non-literal nodes. However, since the URI prefixes are known to us and the key values in the GtoPdb are generally persistent (the same entity does not change its key over different versions), we are able to identify a precise alignment between any pair of versions that will serve as ground truth (GtoPdb). For example, the calcitonin ligand is identified in all versions as ligand 685. In version 1 this is given a URI `http://gtopdb.org/ver1/ligand685` and in version 2 `http://gtopdb.org/ver2/ligand685`.

Node and edge counts are shown in Figure 12. These graphs do not have any blank nodes, and the number of literals is slightly larger than the number of URIs.

In Figure 13 we show the number of aligned nodes in all pairs of consecutive versions by the hybrid and overlap alignment together with the number of nodes aligned by ground truth as well as the total number of nodes (Total) present in both versions. All counts are free of duplicates: any two URIs coming from two versions but representing the same tuple are counted as one. Comparing the values of Total and GtoPdb allows us characterize the degree of relative change

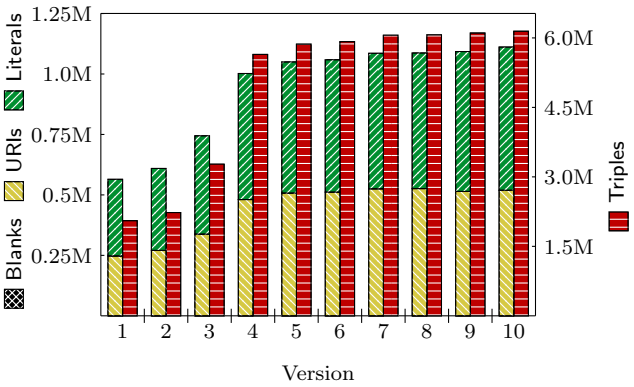

Figure 12: GtoPdb dataset versions.

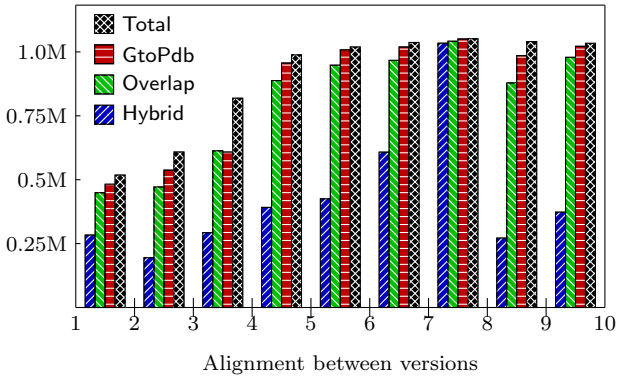

Figure 13: Alignments (GtoPdb).

between versions. In particular between versions 3 and 4 these two values are most different, which indicates a large number of changes (mainly insertions of new nodes). On the other hand, the changes are minute between versions 7 and 8. In general, the value of the overlap alignment are significantly closer than those of the hybrid alignment. The results suggest that the hybrid alignment is sensitive to changes as they propagate throughout the graph, while overlap may better handle changes.

We can now use the ground truth in (GtoPdb) to substantiate these observations and to evaluate the precision of the alignments. In the ground truth a node is aligned to at most one other node, while the overlap and hybrid alignment may map a node to multiple nodes. Consequently, for every alignment we identify the numbers of: exact matches – any node that is aligned to the same set of nodes as the ground truth, inclusive matches – any node that is aligned to a set of nodes that properly includes the node indicated by the ground truth, missing matches – any node that is mapped to a set of nodes that does not include the node indicated by the ground truth, and false matches – any node that is aligned to a nonempty set of nodes while the ground truth does not align the node to any node. We present the result in Figure 14. Clearly, the results confirm that the overlap significantly outperforms the hybrid alignment. We point out that the relative change between versions (as we read it by comparing the values GtoPdb and Total in Figure 13) is not a good indicator of the performance of the hybrid alignment e.g., the hybrid exhibits better precision when aligning versions 3 and 4, where the relative change is significant, than it does when aligning versions 5 and 6,

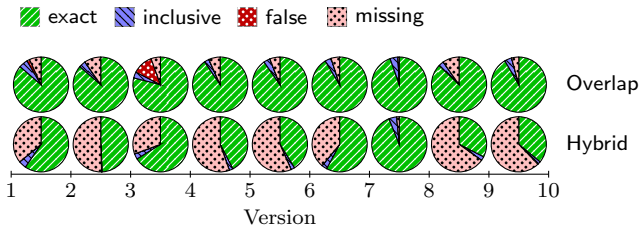

Figure 14: Alignment precision (GtoPdb).

where the relative change is smaller. Interestingly, for the overlap alignment there is a dependence between the relative change between two versions and the precision of the overlap alignment. In particular, the overlap alignment between versions 3 and 4 has the worst precision overall and even aligns incorrectly a significant number of nodes. Our investigations of why nodes are falsely aligned indicate that it mainly happens to nodes that are inserted and deleted between the two versions and that the main reason of false alignment of a node is the number of previously existing nodes present in its outbound neighbourhood. For instance, out of 177K inserted URIs 31K are falsely aligned, and in case of the falsely aligned URIs on average only 9% of outbound nodes are newly inserted nodes while in case of inserted nodes that are correctly unaligned this average is higher and reaches 31%. In Figure 15 we further investigate how the precision can be controlled with the threshold value used by the overlap alignment (between versions 3 and 4) The findings are as expected: the lower the threshold value

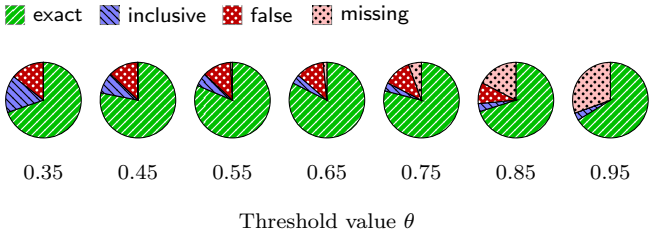

Figure 15: Overlap alignment between versions 3 and 4 (GtoPdb) for different threshold values.

the higher number of false and inclusive matches while the number of missing matches decreases with the value of the threshold. The number of exact matches reaches maximal value at threshold equal 0.65.

5.3 DBpedia

To evaluate scalability of our methods, we report running times on a subset of DBpedia containing category information (including hierarchical information and wikipedia article categorization). We used versions 3.0 through 3.5, run our experiments on a MacBook Pro with 2.3 GHz Intel Core i7, 16 GB RAM, and 512 GB SSD, our (single-thread) implementation was in Python 2.7. The RDF graphs progressively grow from 2.6M nodes and 7.6M triples to 4.2M nodes and 13.7M triples. The performance of our methods fluctuates mainly due to the varying number of overlapping nodes between two consecutive versions. The general trend appears proportional to the size of the input graphs. Furthermore, the execution times are in line with those presented in [16], which suggest that our methods should scale to larger datasets, using methods such as MapReduce.

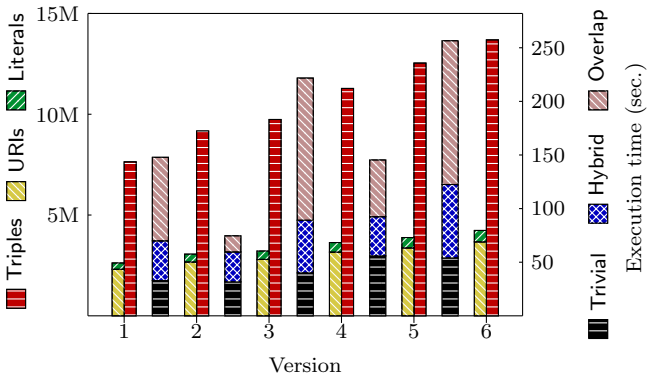

Figure 16: Evaluation time on a subset of DBpedia

6. CONCLUSIONS AND FUTURE WORK

We have presented an approach of identifying nodes corresponding to the same entity in different versions of the same graph, a task whose importance has recently been identified [10]. Our approach is based on the classical notion of bisimulation, which essentially defines the identity of a node based on the identity of its outbound neighborhood. This approach is particularly suited to align the nodes of two graphs that follow the same structure, and evolving RDF is one such real-life scenario. We have also presented a generalization of the basic bisimulation technique that produces weighted partitions, which allows to approximate similarity measures on nodes without incurring the high complexity of computing similarity measures, a matter of obvious importance when handling large RDF graphs.

While our methods are relatively straightforward, they have been designed with simplicity and possible extensions in mind. In the future, we would like to explore variants of our approach where only selected parts of the outbound neighborhood are used, for instance specified by a notion of a key for graph, possibly allowing to align nodes of graphs following different structure, or even a broader context of the node involving its inbound neighborhood and the triple where the node is used as predicate, possibly allowing to better align them. Our experiments show, however, that the presented methods perform very well in the scenario of evolving RDF database.

An interesting question arises: can the (constructed) alignments be used to construct compact representations of all versions of an RDF database? One way of approaching this would be to decorate triples with intervals that represent versions where the triple was present. Our preliminary observations suggest that triples tend to enter and leave with their subject. with its subject, and moving the interval information where possible to the subject nodes could offer further improvements on space requirements.

Acknowledgements We are grateful to Simon Jupp and Tony Burdett for discussions on the EFO database and to Joanna Sharman and Jamie Davies for the GtoPdb data. The referees also made many useful comments. This work was funded by the EU DIACHRON project, the EPSRC SOCIAM project and NSF IIS 1302212: Citing Structured and Evolving Data.

7. REFERENCES

[1] P. Buneman, S. Khanna, K. Tajima, and W.-C. Tan. Archiving scientific data. *ACM Transactions on Database Systems (TODS)*, 29(1):2–42, March 2004.

[2] Frank Emmert-Streib, Matthias Dehmer, and Yongtang Shi. Fifty years of graph matching, network alignment and network comparison. *Information Sciences*, 346:180–197, 2016.

[3] A. Y. Halevy, A. Rajaraman, and J. J. Ordille. Data integration: The teenage years. In *International Conference on Very Large Data Bases (VLDB)*, pages 9–16, 2006.

[4] A. J. Harmar et al. IUPHAR-DB: the IUPHAR database of G protein-coupled receptors and ion channels. *Nucleic acids research*, 37(suppl 1):D680–D685, 2009.

[5] A. Hogan, M. Arenas, A. Mallea, and A. Polleres. Everything you always wanted to know about blank nodes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 27:42–69, 2014.

[6] X. Huang. A lower bound for the edit-distance problem under arbitrary cost function. *Information Processing Letters*, 27(6):319–321, 1988.

[7] D. Justice and A. Hero. A binary linear programming formulation of the graph edit distance. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(8):1200–1214, 2006.

[8] H. Köpcke and E. Rahm. Frameworks for entity matching: A comparison. *Data and Knowledge Engineering*, 69(2):197–210, 2010.

[9] H. W. Kuhn. The hungarian method for the assignment problem. *Naval Research Logistics (NRL)*, 52(1):7–21, 2005.

[10] Christina Lantzaki and Yannis Tzitzikas. Tasks that require, or can benefit from, matching blank nodes. *CoRR*, abs/1410.8536, 2014.

[11] J. Malone et al. Modeling sample variables with an experimental factor ontology. *Bioinformatics*, 26(8):1112–1118, 2010.

[12] S. Melnik, H. Garcia-Molina, and E. Rahm. Similarity flooding: A versatile graph matching algorithm and its application to schema matching. In *International Conference on Data Engineering (ICDE)*, pages 117–128, 2002.

[13] R. Paige and R. E. Tarjan. Three partition refinement algorithms. *SIAM Journal on Computing*, 16(6):973–989, 1987.

[14] V. Papavasileiou, G. Flouris, I. Fundulaki, D. Kotzinos, and V. Christophides. High-level change detection in RDF(S) KBs. *ACM Transactions on Database Systems (TODS)*, 38(1):1, 2013.

[15] Erhard Rahm and Hong Hai Do. Data cleaning: Problems and current approaches. *IEEE Data Engineering Bulletin*, 23(4):3–13, 2000.

[16] A. Schätzle, A. Neu, G. Lausen, and M. Przyjaciel-Zablocki. Large-scale bisimulation of RDF graphs. In *International Workshop on Semantic Web Information Management (SWIM)*, page 1. ACM, 2013.

[17] Y. Tzitzikas, C. Lantzaki, and D. Zeginis. Blank node matching and RDF/S comparison functions. In *International Semantic Web Conference (ISWC)*, pages 591–607. Springer, 2012.

[18] W3C. A direct mapping of relational data to RDF, 2012. `http://www.w3.org/TR/rdb-direct-mapping/`.

[19] C.-K. Wong and A. K. Chandra. Bounds for the string editing problem. *Journal of the ACM*, 23(1):13–16, 1976.

[20] D. Zeginis, Y. Tzitzikas, and V. Christophides. On computing deltas of RDF/S knowledge bases. *ACM Transactions on the Web (TWEB)*, 5(3), 2011.

APPENDIX

A. DEFINITIONS AND PROOFS

Throughout this section we work with a fixed combined graph $G = (N_G, E_G, \ell_G)$ that is the disjoint union of the source graph $G_1 = (N_1, E_1, \ell_1)$ and the target graph $G_2 = (N_2, E_2, \ell_2)$.

Edit distance similarity measure. We use a variant of string edit distance that decomposes strings into sets of words and measures the overlap between the sets.[1] Here, we assume a function *split* that maps literal values into a (nonempty) set of atomic objects and for two (nonempty) sets X and Y we define $overlap(X, Y) = |X \cap Y|/|X \cup Y|$. Now, for $n \in N_1$ and $m \in N_2$

$$\sigma^{(0)}_{\mathsf{Edit}}(n,m) = \begin{cases} 0 & \text{if } \lambda_{\mathsf{Hybrid}}(n) = \lambda_{\mathsf{Hybrid}}(m), \\ overlap(split(\ell_1(n)), split(\ell_2(m))) & \text{if } n \text{ and } m \text{ are both literals unaligned by } \lambda_{\mathsf{Hybrid}}, \\ 0 & \text{if } n \text{ and } m \text{ are both non-literals unaligned by } \lambda_{\mathsf{Hybrid}}, \\ 1 & \text{otherwise.} \end{cases}$$

We point out that $\sigma^{(0)}_{\mathsf{Edit}}$ returns 0 for any pair of unaligned nodes, thus renders them identical in a manner analogous to the workings of the hybrid partitioning.

Next, we iteratively lift the similarity on unaligned literals to other unaligned nodes. Essentially, for two unaligned non-literal nodes n and m we take the maximum score of a matching between the elements of the outbound neighborhoods of n and m. Recall that the outbound neighborhood of a node in a graph is a set consisting of pairs of predicate and object of triples in which the given node is present. Formally, a *matching* between $O_1 \subseteq N_1 \times N_1$ and $O_2 \subseteq N_2 \times N_2$ is a relation $M \subseteq O_1 \times O_2$ such that for every $o_1 \in O_1$ there is at most one $o_2 \in O_2$ such that $(o_1, o_2) \in M$ and vice versa. By $Matchings(O_1, O_2)$ we denote the set of all matchings between O_1 and O_2. The *score* of a matching M w.r.t. a weight function $w : N_1 \times N_2 \to [0, 1]$ is

$$score(M, w) = \bigoplus_{((n_1,n_2),(m_1,m_2)) \in M} \frac{w(n_1, m_1) \oplus w(n_2, m_2)}{\max\{|O_1|, |O_2|\}}.$$

Note that $0 \leq score(M, w) \leq 1$ and we point out that this value can be efficiently computed using the Hungarian algorithm. Also, we assume that O_1 and O_2 are nonempty and point out that the score is used only on outbound neighborhoods of pairs of unaligned non-literal nodes and if for two non-literal nodes their outbound neighborhoods are empty, they are aligned by the hybrid coloring scheme.

Now, given $\sigma^{(i-1)}_{\mathsf{Edit}}$ for $i > 0$ we define $\sigma^{(i)}_{\mathsf{Edit}}$ for $n \in N_1$ and $m \in N_2$ as follows: if n and m are both non-literals unaligned by $\lambda_{\mathsf{Hybrid}}$, then

$$\sigma^{(i)}_{\mathsf{Edit}}(n,m) = \max\{score(M, \sigma^{(i-1)}_{\mathsf{Edit}}) \mid M \in Matchings(out_{G_1}(n), out_{G_2}(m))\}; \tag{6}$$

otherwise $\sigma^{(i)}_{\mathsf{Edit}}(n,m) = \sigma^{(i-1)}_{\mathsf{Edit}}(n,m)$. We point out that the values of $\sigma^{(i)}_{\mathsf{Edit}}(n,m)$ are always positive and may only increase as i increases, and therefore, they converge. At every iteration we compute the cumulative change:

$$\delta_i = \Delta(\sigma^{(i-1)}_{\mathsf{Edit}}, \sigma^{i}_{\mathsf{Edit}}), \qquad \text{where} \qquad \Delta(\sigma, \sigma') = \textstyle\sum_{n \in N_1} \sum_{m \in N_2} |\sigma(n,m) - \sigma'(n,m)|.$$

We repeat the process until δ_i falls below a certain small value $\epsilon > 0$ and by σ_{Edit} we denote $\sigma^{(i)}_{\mathsf{Edit}}$ for such i.

Weighted partition refinement. We formally define the fix-point of a weighted partition refinement.

Definition 5. A *weighted partition refinement* is a function Ξ that maps a weighted partition of a graph G to another weighted partition of the same graph G. We say that Ξ *works well* on $\xi = (\lambda, \omega)$ if when applied $xi' = \Xi(\xi) = (\lambda', \omega')$, the following conditions are satisfied: λ' is finer than λ, for every $n \in N_G$ we have $\omega'(n) \geq \omega(n)$, and Ξ also works well on ξ'.

The above notion ensures stabilisation of the iterative refinement process and it is easy to see that for any weighted partition ξ of G and for any $X \subseteq Unaligned(\xi)$ the weighted refinement $BisimRefine_X$ works well on $Blank(\xi, X)$ (because the only weights that change are initially set to 0 and only grow in subsequent applications of $BisimRefine_X$). To define formally stabilization, we identify the *relative weight change* between ξ and ξ' as $\Delta(\xi, \xi') = \sum_{n \in N} |\xi(n) - \xi'(n)|$ and assume a fixed value $\varepsilon > 0$.

Definition 6. Assume Ξ works well on ξ. The *refinement* $\Xi^*(\xi)$ of ξ w.r.t. Ξ is $\Xi^n(\xi)$, where n is the smallest n such that $\lambda_{n-1} \equiv \lambda_n$ and $\Delta(\xi_{n-1}, \xi_n) < \varepsilon$ with $\Xi^i(\xi) = \xi_i = (\lambda_i, \omega_i)$.

Overlap alignment. We begin by formally defining the edit distance σ^{NL}_{ξ} applied on non-literal nodes. For ease of notation, we assume both the graph G and the weighted partition ξ of G to be fixed.

[1] This formulation is in fact better suited to the data we run our experiments, and for instance, for literals *"14 November 1999"* and *"15 November 1995"* the (weighted) standard edit distance is $\frac{1}{8}$ while the overlap edit distance is $\frac{1}{3}$.

First, for a node $n \in N_G$ and any two colors $c, d \in \mathcal{C}$ we identify the outgoing edges of n whose predicate and object have the colors c and d respectively

$$\mathit{Colored\text{-}Out}_n(c,d) = \{(p,o) \in \mathit{out}_G(n) \mid \lambda(p) = c,\ \lambda(o) = d\}$$

and then sort this set based on the combined weight of the object and predicate

$$S_n(c,d) = ((p_1,o_1),\ldots,(o_k,p_k)),$$

such that $(p_i,o_i) \in \mathit{Colored\text{-}Out}_n$ for all i and $\omega(p_1) \oplus \omega(o_1) \leq \ldots \leq \omega(p_k) \oplus \omega(o_k)$. For any $n \in N_1$ and $m \in N_2$ the coupling $M \subseteq \mathit{out\text{-}color}_G(n) \times \mathit{out\text{-}color}_G(m)$ contains $((p_i,o_i),(p'_i,o'_i))$ for $(p_i,o_i) \in S_n(c,d)$, $(p'_i,o'_i) \in S_m(c,d)$, for $i \leq \min(|S_n(c,d)|,|S_m(c,d)|)$, for any two colors $c,d \in \mathcal{C}$ (the number of pairs of colors needs to be considered is bounded by the sizes of the outbound neighborhoods of n and m). The reminder value is

$$R = \sum_{c,d \in \mathcal{C}} ||S_n(c,d)| - |S_m(c,d)||.$$

We make the following easy to prove claim.

Lemma 1 *Let ξ be a weighted partition of G and σ_ξ the node distance function defined by ξ (5). Take any $n \in N_1$ and $m \in N_2$ that are both unaligned i.e., $n,m \in \mathit{Unaligned}(\xi)$ and let d be the maximum score over all matchings as defined in (6) w.r.t. σ_ξ. Then $\sigma_\xi^{NL}(n,m) = d$.*

Proof of Theorem 1 Let's first recall Theorem 1: **Theorem 1** *Let $\xi_{\mathsf{Overlap}} = (\lambda_{\mathsf{Overlap}}, \omega_{\mathsf{Overlap}})$ be the overlap alignment of G_1 and G_2. Then, for any $n \in N_1$ and $m \in N_2$, if $\lambda_{\mathsf{Overlap}}(n) = \lambda_{\mathsf{Overlap}}(m)$, then $\sigma_{\mathsf{Edit}}(n,m) \leq \omega_{\mathsf{Overlap}}(n) * \omega_{\mathsf{Overlap}}(m)$.* PROOF

The proof is by compound induction by the number of iteration of Algorithm 2 and inner iterations the weighted partition refinement. The initial case is trivial (the claim holds for $\xi_0 = (\lambda_{\mathsf{Hybrid}}, \mathbb{0})$). For the outer inductive step, it suffices to note that the overlap heuristics verifies that the candidates pairs are indeed appropriately close (thanks to Lemma 1). The inner induction on similarity propagation through refinement *BisimRefine* is proven simply by observing that the weights may only decrease with iterations of the weighted refinement.

B. EXPERIMENTAL RESULTS

B.1 Experimental Factor Ontology (EFO)

Version	Nodes	URIs	Literals	Blanks		Triples
				Raw	Normal	
1	122618	13368 (10.90%)	99500 (81.15%)	9750 (7.95%)	9712	204518
2	122806	13421 (10.93%)	99635 (81.13%)	9750 (7.94%)	9712	204823
3	122956	13463 (10.95%)	99754 (81.13%)	9739 (7.92%)	9701	205004
4	132747	13487 (10.16%)	99779 (75.16%)	19481 (14.68%)	11497	234329
5	136392	14311 (10.49%)	109721 (80.45%)	12360 (9.06%)	12322	235556
6	136626	14309 (10.47%)	109957 (80.48%)	12360 (9.05%)	12322	236040
7	149106	14543 (9.75%)	111548 (74.81%)	23015 (15.44%)	14564	269883
8	149351	14571 (9.76%)	111670 (74.77%)	23110 (15.47%)	14609	270366
9	149528	14612 (9.77%)	111763 (74.74%)	23153 (15.48%)	14599	270724
10	149605	14625 (9.78%)	111763 (74.71%)	23217 (15.52%)	14643	270926

Table 1: EFO dataset versions.

Target version \ Source version	1	2	3	4	5	6	7	8	9	10
10	0.471	0.472	0.472	0.436	0.494	0.495	0.461	0.477	0.478	0.647
9	0.471	0.472	0.473	0.436	0.494	0.496	0.461	0.478	0.647	–
8	0.472	0.473	0.473	0.437	0.495	0.496	0.462	0.647	–	–
7	0.491	0.492	0.493	0.455	0.514	0.515	0.648	–	–	–
6	0.538	0.540	0.550	0.504	0.572	0.729	–	–	–	–
5	0.539	0.541	0.551	0.505	0.728	–	–	–	–	–
4	0.544	0.545	0.559	0.675	–	–	–	–	–	–
3	0.599	0.600	0.763	–	–	–	–	–	–	–
2	0.615	0.763	–	–	–	–	–	–	–	–
1	0.762	–	–	–	–	–	–	–	–	–

Table 2: Trivial alignment (EFO).

Quality evaluation. To asses the quality of the hybrid and overlap alignments, we devised an experiment, in which we first use those alignments to identify pairs of old and new URI prefixes (there are 3 such pairs), and then check the number of URIs

Target version	Source version									
10	0.659	0.660	0.768	0.812	0.866	0.869	0.960	0.998	0.999	1.000
9	0.659	0.661	0.769	0.812	0.866	0.869	0.961	0.998	1.000	–
8	0.660	0.661	0.770	0.819	0.867	0.870	0.963	1.000	–	–
7	0.682	0.683	0.793	0.853	0.891	0.894	1.000	–	–	–
6	0.754	0.755	0.889	0.812	0.997	1.000	–	–	–	–
5	0.755	0.757	0.891	0.814	1.000	–	–	–	–	–
4	0.762	0.763	0.902	1.000	–	–	–	–	–	–
3	0.841	0.843	1.000	–	–	–	–	–	–	–
2	0.998	1.000	–	–	–	–	–	–	–	–
1	1.000	–	–	–	–	–	–	–	–	–
	1	2	3	4	5	6	7	8	9	10

Table 3: Deblank **alignment (EFO).**

Target version	Source version									
10	5613	5523	2448	3598	2640	2642	10643	0	0	0
9	5619	5523	2448	3598	2640	2642	10642	0	0	–
8	5619	5523	2448	3713	2640	2642	11275	0	–	–
7	3245	3149	103	21	137	139	0	–	–	–
6	5474	5354	0	84	0	0	–	–	–	–
5	5472	5352	0	84	0	–	–	–	–	–
4	180	84	0	0	–	–	–	–	–	–
3	120	0	0	–	–	–	–	–	–	–
2	120	0	–	–	–	–	–	–	–	–
1	0	–	–	–	–	–	–	–	–	–
	1	2	3	4	5	6	7	8	9	10

Table 4: Hybrid **alignment (EFO).**

that undergo such change and compare it to the number of the URIs detected by our alignment methods. The results are generally positive. When aligning a version with the subsequent one, the renaming occurs only in three instances: between version 1 and 2, where all 14 URIs are correctly aligned, 2) between versions 4 and 5 where a single URIs undergoes the change but is not aligned, and 3) between versions 7 and 8, where out of 628 altered URIs only 5 are not aligned. Never is any URI using old prefix aligned to an URI that does not use the new prefix, which suggests that our methods are suitable to identify prefix changes. The number of URIs that are not aligned grows with the span of versions that are aligned: it reaches 85 out of 643 URIs when aligning version 1 and 10. The overlap alignment used the default threshold value $\theta = 0.75$ and lowering it to 0.65 has significantly improved the results: the most drastic improvement was the alignment between between versions 1 and 10, where the number of unaligned nodes has dropped from 85 to 39. We study more thoroughly the effect of the threshold value on the quality of the alignment in the next set of experiments.

We note we note that our implementation of an alignment based on the similarity measure σ_{Edit} has produced results that compared to the overlap alignment have only a slightly higher number of aligned nodes but also a higher number of erroneously aligned nodes. Furthermore, we were able to run σ_{Edit} only on a few pairs of subsequent versions: on all others it has failed to terminate after several days of computation.

B.2 Guide to Pharmacology database (GtoPdb)

Target version	Source version									
10	5233	5224	5222	7517	5522	5068	1760	5	5	0
9	5230	5221	5219	7514	5519	5065	1757	2	0	–
8	5230	5221	5219	7403	5519	5065	1129	0	–	–
7	15	7	3	3	454	0	0	–	–	–
6	7	9	3	3	454	0	–	–	–	–
5	7	7	3	3	0	–	–	–	–	–
4	12	4	0	0	–	–	–	–	–	–
3	6	6	0	–	–	–	–	–	–	–
2	2	0	–	–	–	–	–	–	–	–
1	0	–	–	–	–	–	–	–	–	–
	1	2	3	4	5	6	7	8	9	10

Table 5: Overlap **alignment (EFO).**

Version	Nodes	URIs	Literals	Triples
1	564448	247627	316821	2054576
2	609347	270379	338968	2229910
3	744464	337799	406665	3278459
4	1002010	480891	521119	5643624
5	1049975	507568	542407	5865307
6	1058981	511558	547423	5919830
7	1085547	525375	560172	6057875
8	1086923	526267	560656	6065688
9	1092833	514380	578453	6106081
10	1111393	519184	592209	6144210

Table 6: GtoPdb dataset versions.

Alignment between	Trivial	Deblank	Hybrid	Overlap	GtoPdb	Total
1 – 2	1	1	283368	449064	482207	518005
2 – 3	1	1	194833	471607	537385	608177
3 – 4	1	1	293076	613203	608083	818689
4 – 5	1	1	391594	887966	956639	988458
5 – 6	1	1	425330	948015	1008587	1019125
6 – 7	1	1	607819	966924	1019355	1036932
7 – 8	1	1	1034117	1041729	1050661	1051641
8 – 9	1	1	271450	878966	985195	1040646
9 – 10	1	1	372955	978778	1021913	1033563

Note: all versions share a common predicate
`http://www.w3.org/1999/02/22-rdf-syntax-ns#type`,
which explains the values 1 in the Trivial and Deblank columns.

Table 7: Alignments (GtoPdb).

Alignment between	Exact	Inclusive	Missing	False	Total
1 – 2	165064 (59.61%)	12390 (4.47%)	99429 (35.91%)	19 (0.01%)	276902
2 – 3	166881 (49.16%)	1244 (0.37%)	171304 (50.46%)	56 (0.02%)	339485
3 – 4	340542 (66.17%)	16514 (3.21%)	157533 (30.61%)	59 (0.01%)	514648
4 – 5	216312 (42.40%)	11202 (2.20%)	282491 (55.38%)	134 (0.03%)	510139
5 – 6	212005 (41.18%)	11080 (2.15%)	291668 (56.65%)	79 (0.02%)	514832
6 – 7	304738 (57.80%)	16701 (3.17%)	205784 (39.03%)	32 (0.01%)	527255
7 – 8	494184 (93.90%)	23855 (4.53%)	8272 (1.57%)	0 (0.00%)	526311
8 – 9	178294 (32.53%)	12820 (2.34%)	356880 (65.12%)	55 (0.01%)	548049
9 – 10	187956 (35.97%)	9784 (1.87%)	324753 (62.14%)	114 (0.02%)	522607

Table 8: Hybrid alignment precision (GtoPdb).

Alignment between	Exact	Inclusive	Missing	False	Total
1 – 2	238993 (86.31%)	14370 (5.19%)	18893 (6.82%)	4646 (1.68%)	276902
2 – 3	293106 (86.34%)	13361 (3.94%)	32933 (9.70%)	85 (0.03%)	339485
3 – 4	406307 (78.95%)	18192 (3.53%)	28343 (5.51%)	61806 (12.01%)	514648
4 – 5	461598 (90.48%)	13927 (2.73%)	34202 (6.70%)	412 (0.08%)	510139
5 – 6	464890 (90.30%)	19456 (3.78%)	30353 (5.90%)	133 (0.03%)	514832
6 – 7	481471 (91.32%)	19451 (3.69%)	26255 (4.98%)	78 (0.01%)	527255
7 – 8	497787 (94.58%)	24058 (4.57%)	4466 (0.85%)	0 (0.00%)	526311
8 – 9	478083 (87.23%)	16729 (3.05%)	53069 (9.68%)	168 (0.03%)	548049
9 – 10	478737 (91.61%)	21787 (4.17%)	21798 (4.17%)	285 (0.05%)	522607

Table 9: Overlap alignment precision (GtoPdb).

Threshold value θ	Exact	Inclusive	Missing	False	Total
0.35	359002 (69.76%)	81954 (15.92%)	183 (0.04%)	73509 (14.28%)	514648
0.45	401974 (78.11%)	45968 (8.93%)	1113 (0.22%)	65593 (12.75%)	514648
0.55	418481 (81.31%)	29020 (5.64%)	3955 (0.77%)	63192 (12.28%)	514648
0.65	424327 (82.45%)	19855 (3.86%)	7923 (1.54%)	62543 (12.15%)	514648
0.75	406307 (78.95%)	18192 (3.53%)	28343 (5.51%)	61806 (12.01%)	514648
0.85	360967 (70.14%)	17391 (3.38%)	89699 (17.43%)	46591 (9.05%)	514648
0.95	340811 (66.22%)	16514 (3.21%)	157264 (30.56%)	59 (0.01%)	514648

Table 10: Alignment between version 3 and 4 for different threshold value.

Version	1	2	3	4	5	6
URIs	2.31G	2.67G	2.80G	3.16G	3.37G	3.67G
Literals	0.313G	0.390G	0.416G	0.474G	0.514G	0.565G
Triples	7.64G	9.17G	9.74G	11.28G	12.54G	13.69G

Table 11: Sizes of DBpedia (categories) dataset (G is $*10^6$).

Alignement	1–2	2–3	3–4	4–5	5–6
Trivial	32.907	31.663	40.045	55.877	54.011
Hybrid	69.969	59.594	89.140	92.467	122.480
Overlap	147.832	74.630	221.744	145.370	256.561

Table 12: Execution times for DBpedia (in seconds).