View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by University of Strathclyde Institutional Repository

University of Strathclyde Glasgow

Strathprints Institutional Repository

Al Azwari, Sana and Wilson, John N. (2015) Consistent RDF updates with correct dense deltas. In: Data Science. Springer, pp. 74-86. ISBN 978-3-319-20423-9 , http://dx.doi.org/10.1007/978-3-319-20424-6_8

This version is available at http://strathprints.strath.ac.uk/53083/

Strathprints is designed to allow users to access the research output of the University of Strathclyde. Unless otherwise explicitly stated on the manuscript, Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Please check the manuscript for details of any other licences that may have been applied. You may not engage in further distribution of the material for any profitmaking activities or any commercial gain. You may freely distribute both the url (http://strathprints.strath.ac.uk/) and the content of this paper for research or private study, educational, or not-for-profit purposes without prior permission or charge.

Any correspondence concerning this service should be sent to Strathprints administrator: strathprints@strath.ac.uk

Consistent RDF Updates with Correct Dense *Deltas*

Sana Al Azwari and John N. Wilson

Department of Computer & Information Sciences, University of Strathclyde, Glasgow, UK,
{sana.al-azwari,john.n.wilson}@strath.ac.uk

Abstract. RDF is widely used in the Semantic Web for representing ontology data. Many real world RDF collections are large and contain complex graph relationships that represent knowledge in a particular domain. Such large RDF collections evolve in consequence of their representation of the changing world. Although this data may be distributed over the Internet, it needs to be managed and updated in the face of such evolutionary changes. In view of the size of typical collections, it is important to derive efficient ways of propagating updates to distributed data stores. The contribution of this paper is a detailed analysis of the performance of RDF change detection techniques. In addition the work describes a new approach to maintaining the consistency of RDF by using knowledge embedded in the structure to generate efficient update transactions. The evaluation of this approach indicates that it reduces the overall update size at the cost of increasing the processing time needed to generate the transactions. . . .

Keywords: RDF updates·inferencing·pruning

.

1 Introduction

Resource Description Framework (RDF) is an annotation language that provides a graph-based representation of information about Web resources in the Semantic Web. Because RDF content (in triple form) is shared between different agents, a common interpretation of the terms used in annotations is required. This interpretation is typically provided by an ontology expressed as RDF Schema (RDFS) or Web Ontology Language (OWL). Both RDFS and OWL are expressed as RDF triples. The schema provides additional semantics for the basic RDF model. In any particular data collection, changes in the domain that are reflected by evolution of the ontology may require changes in the underlying RDF data. Due to the dynamic and evolving nature of typical Semantic Web structures, RDF data may change on a regular basis, producing successive versions that are available for publication and distribution [4]. In the context of such dynamic RDF data collections, which may be very large structures, it quickly becomes infeasible to store a historic sequence of updates in any accessible form as a consequence of the significant storage space needed. An alternative solution to propagation and storage of successively updated copies of a data collection is to compute the differences between these copies and use these as a means of transforming the base data structure into subsequent versions. These differences (the *delta*) show triple content that has been changed between two RDF models and can be used to transform one RDF model into another. Rather than storing all versions of a data structure, it is only necessary to store one version and retain the capability of restoring any version of interest by executing the consecutive *deltas*.

The work presented in this paper addresses the problem of change detection in RDF knowledge bases. An important requirement of change detection tools is their ability to produce the smallest correct *delta* that will efficiently transform one RDF model to another. This is a particularly important problem when RDF collections are large and dynamic. In this context, propagation between server and client or between nodes in a peer-to-peer system becomes challenging as a consequence of the potentially excessive use of network bandwidth. In a scenario where RDF update is carried out by push-based processes, the update itself needs to be minimised to restrict network bandwidth costs. In addition, in pull-based scenarios, it is important to limit server processing so that updates can be generated with maximum efficiency. The contribution of this work is an approach for using the smallest *deltas* that will maintain the consistency of an RDF knowledge base together with an evaluation of the performance challenges of generating this structure.

2 Related Work

Managing the differences between RDF knowledge bases using *deltas* is an important task in the ontology evolution process. because they allow the synchronization of ontology changes [2], the update of ontologies to newer versions, and the reduction of storage overhead required to hold ontology versions [8]. Changes between ontologies can be detected using change detection tools that report changes in low-level (RDF) or high level (ontology) structures. High-level change detection techniques typically focus on exploiting semantic variation between ontologies. Example of these tools include SemVersion [9] and PromptDiff [6]. High-level changes may involve adding or generalising domains or renaming classes [7]. By contrast, low-level change detection techniques focus on reporting ontology changes in terms of simple change operations (i.e. add/delete triples). These tools differ in the level of semantic complexity represented by the ontology languages. Work in low-level change detection tools focuses on the exploitation of useful properties for producing *deltas* (e.g. the *delta* size and the level of execution semantics) that can be interpreted by both human and machine.

For example, Zeginis et al. [10] proposed three RDF/S differential *delta* functions associated with the inferred knowledge from RDFS knowledge bases: dense (ΔD); dense & closure (ΔDC) and explicit & dense (ΔED). These *deltas* vary in the application of inference to reduce their size and are explained in greater detail in Section 3. Results show that ΔD produced the smallest *delta* but was prone to ambiguity and may potentially produce inconsistently updated RDF knowledge bases. In this paper, we characterise ΔD_c, which is a correction method for ΔD that supports consistency when updating an RDF knowledge base. We demonstrate the correctness of ΔD_c and evaluate ΔD_c, ΔED and ΔE in terms of *delta* size and the processing performance of producing the *deltas* using different sizes of synthetic datasets.

M	M'
(Graduate subClassOf Person),	(Head_Teacher subClassOf Teacher),
(Student subClassOf Person),	(Teacher subClassOf Staff),
(Head_Teacher subClassOf Staff),	(Staff subClassOf Person),
(Teacher subClassOf Staff),	(Graduate subClassOf Student),
(Staff subClassOf Person),	(Student subClassOf Person),
(John type Student).	(Teacher subClassOf Person),
	(Head_Teacher subClassOf Person),
	(John type Person).

Fig. 1. Sample data structure before and after update

$$\begin{aligned} \Delta E = &\ \{\text{Del (Graduate subClassOf Person)},\\ &\ \text{Del (Head\_Teacher subClassOf Staff)},\\ &\ \text{Del (John type Student)}\}\\ \cup &\ \{\text{Ins (Head\_Teacher subClassOf Teacher)},\\ &\ \text{Ins (Graduate subClassOf Student)},\\ &\ \text{Ins (Teacher subClassOf Person)},\\ &\ \text{Ins (Head\_Teacher subClassOf Person)},\\ &\ \text{Ins (John type Person)}\} \end{aligned}$$

Fig. 2. The explicit *delta*

$$\begin{aligned} \Delta ED = &\ \{\text{Del (John type Student)}\}\\ \cup &\ \{\text{Ins (Head\_Teacher subClassOf Teacher)},\\ &\ \text{Ins (Graduate subClassOf Student)},\\ &\ \text{Ins (Teacher subClassOf Person)},\\ &\ \text{Ins (Head\_Teacher subClassOf Person)},\\ &\ \text{Ins (John type Person)}\} \end{aligned}$$

Fig. 3. The explicit dense *delta*

3 RDF Change Detection Techniques

RDF updates allow low-level triple operations for insertion and deletion that were formalised by Zeginis et al [10]. In the context of the two example RDF models M and M' in Figure 1, the naïve way of generating the *delta* involves computing the set-difference between the two versions using the explicit sets of triples forming these versions. The explicit *delta* (ΔE) contains a set of triples to be deleted from and inserted into M in order to transform it into M'.

Definition 1 (Explicit *delta*). *Given two RDF models M and M', let* t *denote a triple in these models, Del denote triple deletion which is calculated by $M - M'$, and Ins denote triple insertion which is calculated by $M' - M$. The explicit* delta *is defined as:*

$$\Delta E = \{Del(t) \mid t \in M - M'\} \cup \{Ins(t) \mid t \in M' - M\}$$

From the example in Figure 1, the *delta* obtained by applying the above change detection function is shown in Figure 2.

Executing these updates against M will correctly transform it to M'. However, this function handles only the syntactic level of RDF and does not exploit its semantics. In the latter context, executing some of the updates in ΔE is not necessary as they can still be inferred from other triples. For instance, we can observe from the example in Figure 1 that deleting *(Graduate subClassOf Person)* from M, in order to transform it into M', is not necessary as this triple can still be inferred from the triples *(Graduate subClassOf Student)* and *(Student subClassOf Person)* in M'. Since this update is not necessary, it is useful to remove it from the *delta*. RDF data is rich in semantic content and exploiting this in the process of updating RDF models can minimize the *delta* size and therefore the storage space and the time to synchronize changes between models.

Unnecessary updates can be avoided by applying a differential function that supports reasoning over the closure of an RDF graph. In RDF inference, the closure can be calculated in order to infer some conclusions from explicit triples. This process is carried out by applying entailment rules against the RDF knowledge base. In this work, we consider the RDFS entailment rules provided by the RDFS semantics specification [3]. This specification contains 13 RDFS entailments rules, however only the rules that have an effect on minimizing the *delta* size are used in the current approach for change detection. These rules are shown in Table 1.

Definition 2 (Closure). *Let t be a triple with subject, predicate, object (SPO). The closure of M is defined as M extended by those triples that can be inferred from the graph M. The closure of an RDF graph M is denoted by:*

$$C(M) = M \cup \{t \in (SPO) \mid M \models t\}$$

Example 1. Let $M = \{a\ \mathit{subClassOf}\ b, b\ \mathit{subClassOf}\ c\}$ then the closure of M will contain these triples and a further triple $\{a\ \mathit{subClassOf}\ c\}$.

The rules in Table 1 can be used in the explicit dense function (ΔED), which combines both explicit and inference approaches for computing the *delta*. The inserted set of triples is computed explicitly as in ΔE, while the delete set is computed based on inference using the rule set.

Definition 3 (Explicit dense *delta*). *Let M, M', Del(t), Ins(t) be as stated in Definition 1. Additionally let $C(M')$ denote the closure of M'. ΔED is defined as:*

$$\Delta ED = \{\mathit{Del}(t) \mid t \in M - C(M')\} \cup \{\mathit{Ins}(t) \mid t \in M' - M\}$$

Applying this function to the example in Figure 1 produces the *delta* shown in Figure 3. The inserts in this *delta* are achieved by explicitly calculating the set difference $M' - M$ to provide the set of triples that should be inserted to M in order to transform it into M'. On the other hand, the set of deleted triples is achieved by calculating the closure of M' using the RDFS entailment rules to infer new triples and add them to M'. From the example, the inferred triples in M' are:
(Teacher subClassOf Person)
(Head_Teacher subClassOf Person)
(Head_Teacher subClassOf Staff)
(Graduate subClassOf Student)
These inferred triples are then added to M' to calculate the set difference $M - C(M')$ which results in only one triple to delete: *(John type Student)*. The number of updates produced by this *delta* is smaller than the one produced by the ΔE as a result of the inference process.

The effect of the inference process in minimising ΔED was limited to applying the inference rules when computing the deleted set of triples only. Applying inference

	If KB contains	**Then add to KB**
rdfs1	s rdf:type x *and* x rdfs:subClassOf y	s rdf:type y
rdfs2	x rdfs:subClassOf y *and* y rdfs:subClassOf z	x rdfs:subClassOf z
rdfs3	p rdfs:subPropertyOf q *and* q rdfs:subPropertyOf r	p rdfs:subPropertyOf r

Table 1. Relevant rules

Algorithm 1: Generation of the corrected dense *delta* ΔD_c

```
   Data: $M$,$M'$
   Result: $\Delta D_c$
1  Del = $M - M'$;
2  Ins = $M' - M$ ;
3  for $a \in Del$ do
4      if inferable(a, $M'$) then
5          remove a from Del;
6  for $b \in Ins$ do
7      if (inferable(b, $M$)) and (all antecedents of b $\notin$ Del) then
8          remove b from Ins;
9  $\Delta D_c$ = Del $\cup$ Ins;
```

rules for computing the inserted triples may further reduce the number of updates. For example, inserting the three triples *(Teacher subClassOf Person)*, *(Head_Teacher subClassOf Person)* and *(John type Person)* into M may not be necessary because these triples implicitly exist in M and can be inferred in M using the RDFS entailment rules. In this example, applying *rdfs1* to M would infer *(John Type Person)* while the other two triples could be inferred using *rdfs2*. The application of inference over both the insert and delete sets produces the *dense delta* (ΔD).

Definition 4 (Dense *delta*). *Let M, M', Del(t), Ins(t) be as stated in Definition 1. The dense* delta *is defined as:*

$$\Delta D = \{Del(t) \mid t \in M - C(M')\} \cup \{Ins(t) \mid t \in M' - C(M)\}$$

Figures 4(a) and 4(b) illustrate the distinction between ΔED and ΔD. In the former only the deletes that are not in $C(M')$ need to be carried out. In this case, $C(M)$ is not checked to see whether all of the planned inserts need to be applied. In the case of ΔD, deletes are handled in the same way as in ΔED however inserts are only applied if they are not in $C(M)$. This results in minimising both delete and insert operations.

From the example in Figure 1, the updates generated by applying (ΔD) are shown in Figure 5. ΔD is smaller than either ΔE or ΔED with only three updates to transform M to M'. However, in contrast to ΔE and ΔED, ΔD does not always provide the correct *delta* to carry out the transformation. In this case, applying ΔD to transform M into M' will transform M as shown in Figure 7. This *delta* function does not correctly update M to M' because when applying the updates, *(John type Person)* is not inserted into M and cannot be inferred in M after the triple *(John type Student)* has been deleted.

4 Checking the Dense *Delta*

The contribution of this work is a solution to the correctness of ΔD.

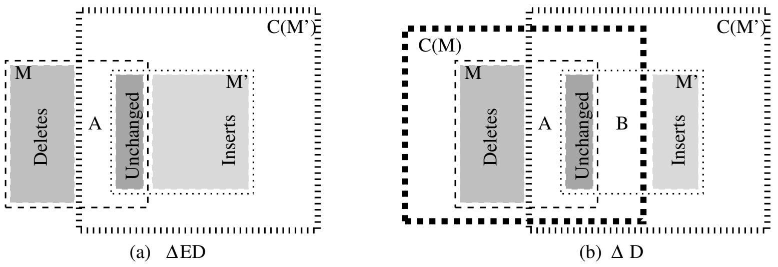
C(M')
M
Deletes
A
Unchanged
M'
Inserts
(a) ΔED
C(M)
C(M')
M
Deletes
A
Unchanged
B
M'
Inserts
(b) Δ D
A – deletes that are still in C(M') once M' has been generated
B – inserts that are already in C(M) before it is updated

Fig. 4. The distinction between ΔED and ΔD.

ΔD = {Del (John type Student)}
∪ {Ins (Head_Teacher subClassOf Teacher),
Ins (Graduate subClassOf Student) }

Fig. 5. The dense *delta* (ΔD)

ΔD_c = { Del (John type Student)}
∪ {Ins (Head_Teacher subClassOf Teacher),
Ins (Graduate subClassOf Student),
Ins (John type Person)}

Fig. 6. The corrected dense *delta* ΔD_c

Definition 5 (Corrected dense *delta*). *Let ΔE, $C(M)$ and $C(M')$ be as defined previously and additionally let $s \rightarrow t$ indicate that s is an antecedent of t. The corrected dense delta ΔD_c is defined as*

$$\Delta D_c = \Delta E - (\{Del(t) \mid t \in C(M')\} \cup \{Ins(t) \mid t \in C(M) \wedge \{s \rightarrow t \mid s \notin Del(t)\}\})$$

Under the semantics of the subset of RDFS rules in Table 1 all *deltas* are unique with respect to the difference between $C(M)$ and $C(M')$. ΔD_c does not require M or M' to be closed and consequently it is not unique.

The corrected dense *delta* is produced by checking triples in both the insert and delete sets of ΔE. Firstly, the delete set should be calculated before the insert set. Secondly, all antecedents for each inferred triple must be checked to see whether they exist in the delete set. If one or both antecedents exist in the delete set then this triple cannot be inferred. To calculate the closure for M in order to compute the insert set, if two triples in M point to a conclusion based on the rules, then these triples are checked against the deleted set. The conclusion cannot be true if at least one of the two triples exists in the delete set, otherwise, the conclusion is true and the triple can be inferred in M. This process (Algorithm 1) produces the corrected dense *delta* ΔD_c.

Because the delete set is calculated first, the triple *(John Type Person)* will not be inferred from *(John Type Student)* and *(Student SubclassOf Person)* given that the former is included in the delete set. The *delta* will result in the updates shown in Figure 6. Applying these updates to M will result in the model in Figure 8. This model is identical to M', indicating the correctness of ΔD_c. The number of updates after fixing the incorrectness problem is increased but it produces a correct *delta*. However, this number is smaller than the number of updates produced by ΔED or equal to it in the worst case. In such a worst case, none of the inserted triples in ΔD_c can be inferred in M because either there are no triples that can be inferred or at least one of the antecedents of every inferable triple is included in the delete set.

	M
Original triples	(Graduate subClassOf Person), (Student subClassOf Person), (Head_Teacher subClassOf Staff), (Teacher subClassOf Staff), (Staff subClassOf Person), ~~(John Type Student).~~
Inserted triples	(Head_Teacher subClassOf Teacher), (Graduate subClassOf Person).

Fig. 7. Incorrect updates

	M
Original triples	(Graduate subClassOf Person), (Student subClassOf Person), (Head_Teacher subClassOf Staff), (Teacher subClassOf Staff), (Staff subClassOf Person), ~~(John Type Student).~~
Inserted triples	(Head_Teacher subClassOf Teacher), (Graduate subClassOf Person). (John Type Person)

Fig. 8. Correct updates

Both ΔED and ΔD_c functions discussed above apply *inference-then-difference* strategy. This implies that the full closure of the RDF models should be calculated and all the possible conclusions under the RDFS entailment rules are stored in these models. By contrast, a backward inference approach uses the *difference-then-inference* strategy. That is, instead of computing the entire closure of M', in the case of ΔED, this method calculates first the set-differences $M - M'$ and $M' - M$, and then checks every triple in $M - M'$ and removes it if it can be inferred in M'. The operation becomes:

$$\text{Remove } t \text{ from } (M - M') \text{ if } t \in C(M')$$

Instead of pre-computing the full closure in advance, this method infers only triples related to the result of $M - M'$. This would be expected to improve the time and space required in change detection by comparison with the forward inference approach.

In the example dataset shown in Figure 1, to calculate ΔED using the backward inference strategy, the sets of inserted and deleted triples are calculated using set-difference operation in the same way as when calculating ΔE. After calculating the changes at the syntactic level, each triple in the delete set is checked to see if it can be inferred in M' using the RDFS entailment rules. For example, the triple *(Graduate subClassOf Person)* in $M - M'$ is checked to see if it can be derived in M'. Using the RDFS entailment rules this triple can be derived from the two triples *(Graduate subClassOf Student)* and *(Student subClassOf Person)*, therefore, this triple is removed from $M - M'$. Rather than checking all the triples in M', only the three triples in $M - M'$ are checked.

For applying the backward inference in ΔD_c, first the set of deleted triples in $M - M'$ is inferred as explained above, then the set of inserted triples in $M' - M$ is also checked to see if it can be derived in M. However, to guarantee the correctness of the *delta*, before removing the inferable triples from the *delta*, antecedents of each inferable triple in $M' - M$ are checked to see if at least one of them exists in $M - M'$. If this is the case, this triple cannot be removed from the *delta*. Algorithm 1 describes the generation of ΔD_c by backward inference.

Both forward inference and backward inference produce the same *delta*, but the latter applies the inference rules on only the necessary triples. However, although the backward inference method is applied to infer only relevant triples, applying the inference on some of these triples might be unnecessary allowing pruning to be applied before backward inference [4]. The general rule for pruning is that if the subject or object of a triple in $M - M'$ or $M' - M$ does not exist in M' or M, respectively, then

						Reduction strength[1]		
Versions	**M**	ΔE	ΔED	ΔD_c	ΔD	ΔED	ΔD_c	ΔD
$(M - M1')$	121374	48136	47270	44270	44212	1.8%	8.0%	8.2%
$(M - M2')$	127374	126710	125228	119228	119098	1.2%	5.9%	6.0%
$(M - M3')$	139374	230372	227334	215334	214926	1.3%	6.5%	6.7%
$(M - M4')$	157374	343594	338663	317662	317109	1.4%	7.5%	7.7%
$(M - M5')$	169374	412233	406129	379129	378482	1.5%	8.0%	8.2%

Table 2. Triple counts used in evaluation.

Abbr.	***delta***
E	explicit
EDFI	explicit dense, forward inference
EDBI	explicit dense, backward inference
EDPBI	explicit dense, pruned,backward inf.
D_cFI	corrected dense,forward inference
D_cBI	corrected dense, backward inference
D_cPBI	corrected dense, pruned, backward inf.

Table 3. Change detection techniques.

this triple cannot be inferred, consequently the triple can be pruned before the inference process begins. Although pruning may reduce the workload for inferencing, it carries a potential performance penalty [1].

5 Results and Discussion

To evaluate the correction method described above in the context of ΔE and ΔED, the correctness, processing time and *delta* size of updates to enhanced RDF KBs of different sizes are assessed. The objective of this evaluation is to compare the different *delta* computation methods (i.e. ΔE, ΔED, ΔD_c) and approaches (i.e. forward inference (FI), backward inference (BI) and pruned backward inference (PBI)) by measuring and comparing their *delta* computation times over synthetic datasets and by validating their effect on the integrity of the resulting RDFS KBs.

The dataset contains both the Gene Ontology (GO) vocabulary and associations between GO terms and gene products including the Uniprot Taxonomy. This data set was chosen because it is frequently updated, with a new version being released every month. The dataset includes five versions selected to show a range of values over the period 2005 and 2014. Using this dataset, the oldest version (i.e. the 2005 version) was transformed to five versions released between 2006 and 2014. This gradually increases the *delta* size with a consequent effect on the performance of the different change detection methods. The real-world data was enhanced by synthetic data prepared by incorporating 20% additional triples representing *subClass*, *subProperty* and *type* properties. Synthetic data was added to ensure that subProperty rule was exercised and to arrange for the model to contain redundant triples (i.e. explicit data that can also be inferred from antecedents). The level of enhancement was chosen to secure a measurable effect without obscuring the structure of the original data.

Using the enhanced datasets, change detection techniques shown in Table 3 were implemented. A triple store was constructed in MySQL to handle the RDF collections and the *deltas*. Indexing was excluded to preserve the validity of the use-case. The Jena framework was used to read the RDF dataset into the triple store and to validate change detection techniques by comparing the updated RDF dataset with the target RDF dataset. All experiment were performed on Intel Xeon CPU X3470 @ 2.93GHz - 1 cpu with 4 cores and hyperthreading, Ubuntu 12.04 LTS operating system and 16GB memory. Garbage collection and JIT compilation were controlled.

[1] Reduction strength is the percentage reduction achieved by inference i.e. $\frac{|\Delta E|-|\Delta ED|}{|\Delta E|}$ or $\frac{|\Delta E|-|\Delta D_c|}{|\Delta E|}$

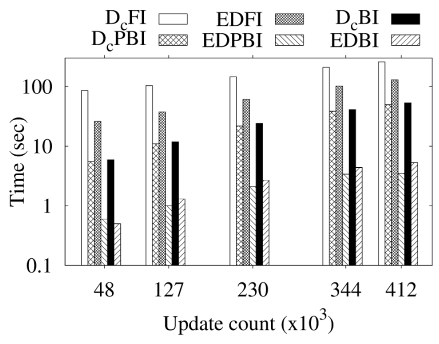

Fig. 9. Inference time

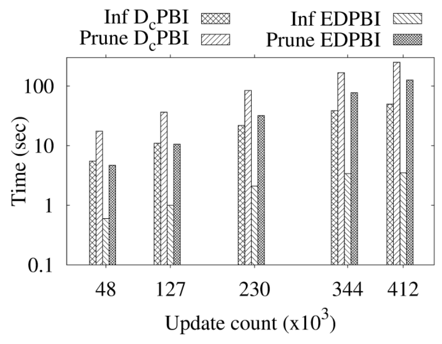

Fig. 10. Reasoning times

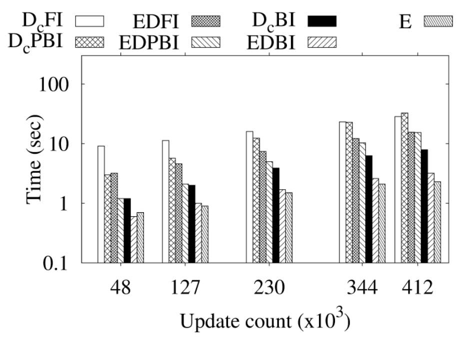

Fig. 11. *Delta* time

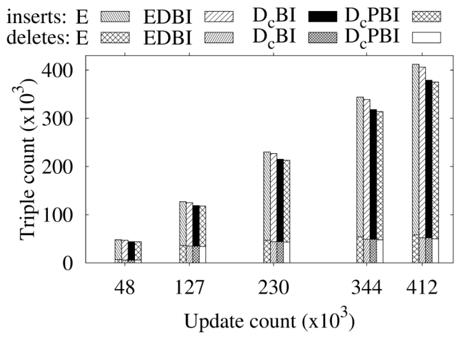

Fig. 12. *Delta* size

Table 2 and Figure (9-12) report the *delta* sizes and the *delta* computation times, respectively. From Table 1, the *deltas* produced by ΔE exceed those of ΔED and ΔD. These *deltas* are smaller than those produced by ΔE as a consequence of applying inference on the delete set of triples (ΔED) and ΔD_c further reduces the *deltas* as a result of inferring both the delete and insert set of triples when calculating the *deltas*. ΔD in turn may be smaller than ΔD_c but its application in the update process may lead to an inconsistent result as noted in Figure 7.

In Figure 9 it can be seen that of the *deltas* evaluated in these experiments, EDBI and the pruned version of the same approach can be generated with the lowest inference time. This is a consequence of both the efficiency of backward inference and the application of inference only to the delete set. At the other end of the spectrum, forward inference methods are slower, as a consequence of the time needed to produce the closure for both models. Forward inference is expensive but becomes useful where models are being queried. However since the focus of this work is updating models, backward inference is a more appropriate approach.

Pruning generally helps to further reduce the inference time however the process adds further expense. Figure 10 shows the reasoning time (i.e the time taken up by both inferencing and pruning). This indicates that for the data structure used, the time

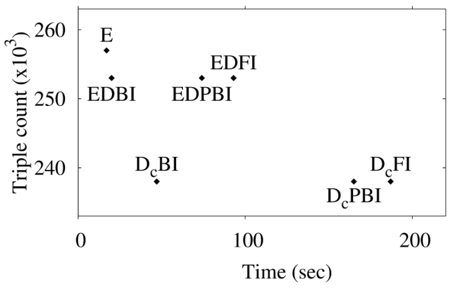

Fig. 13. Comparison of *delta* approaches.

required to carry out pruning exceeds the inference time both for ΔD_c and ΔED. This is consistent with previous findings [1]. The overall *delta* time shown in Figure 11 indicates that taking account of set difference operations, inferencing and pruning, approaches that prune the *delta* set tend to require significantly more processing power than non-pruning approaches. Overall, the ΔE is the fastest process since no pruning or inferencing is carried out. The *delta* sizes shown in Figure 12 indicate that applying inference on this data set reduces the updates that need to be executed, particularly when it is applied to both the insert and delete sets.

The relationship between Figures 11 and 12 is summarised in Figure 13, which is based on the average *delta* size and average generation time for all the data models. Figure 13 shows the interaction between the degree of inference (i.e. the delete set and/or the insert set or no inference at all) and the approach to inferencing (i.e. inferring all triples or only necessary triples) and their impact on the delta size and the delta computation time. It can be seen that ΔD_c has the smallest delta size compared to ΔED and ΔE. It can also be seen that the approach to inferencing affects the delta computation time. Figure 13 indicates that ΔB_c is more efficient (i.e smaller *delta* size and faster generation) than the other methods tested. Overall, Figure 12 shows that the computation time increases in the sequence of explicit, backward inference, pruned backward inference, forward inference whereas the *delta* size increases in the sequence ΔD_c, ΔED, ΔE.

The consistency of M' after *delta* application was evaluated by comparing the in-memory M' produced by applying the *delta* to M in the database with the original in-memory M' using the Jena isIsomorphic method. Applying ΔD_c using the approach described above was found to result in the same M' as that used to generate the *delta*. By contrast, tests carried out to assess the consistency of applying the uncorrected ΔD indicate that in all the models tested, this approach always failed to produce consistent updates.

The overall effect of these results is to indicate that ΔD_c provides a viable route to minimising the data that would need to be transferred from a server to a client in order to update copies of an RDF data store. Pruning may assist this process but comes at a cost of additional processing time, which may be unacceptable in a peer-to-peer context or where updates need to be generated on demand.

By contrast with inference strength[2] [10, p 14:20], reduction strength shown in Table 2 indicates when the size of ΔE, ΔED and ΔD_c are different i.e. when inference is capable of making a difference to the size of the *delta*. When the inference strength is zero, there are no inferences to be made and the model is closed. Under these circumstances, $|\Delta E| = |\Delta D_c|$. However, $|\Delta E|$ may still be equal to $|\Delta D_c|$ when the inference strength is greater than zero. This occurs when, for example, none of the triples in the *delta* are inferable in M.

Example 2. Let $M = \{w\ subClassOf\ x, x\ subClassOf\ y, y\ subClassOf\ z\}$ and $M' = \{w\ subClassOf\ x, x\ subClassOf\ y, y\ subClassOf\ z, n\ subClassOf\ r\}$. Under these circumstances, $\Delta E = \{ins\{n\ subClassOf\ r\}\}$ and since this triple can not be inferred in M, $\Delta D_c = \{ins\{n\ subClassOf\ r\}\}$. Using the expression in footnote 2, the inference strength has a value of 1 but $|\Delta E| = |\Delta D_c|$ i.e. the inference strength is significantly different from zero but there are no inferred triples. This contrasts with the definition provided by [10, p 14:20], which states that inference strength is proportional to the count of inferable triples. Alternatively, the reduction strength in this example is zero, thereby providing an effective guide to indicate when $|\Delta E| = |\Delta D_c|$, which is not clearly shown by the inference strength.

Both inference strength and reduction strength also give an indication of the work load of pruning. High values for these parameters indicate that a large number of triples can be inferred. However, adding such inferable triples provides a large collection of data that needs to be checked for possible pruning before inference can take place.

Example 3. Let $M = \{w\ subClassOf\ x, x\ subClassOf\ y, y\ subClassOf\ z\}$ and $M' = \{w\ subClassOf\ x, x\ subClassOf\ y, y\ subClassOf\ z, n\ subClassOf\ r, w\ subClassOf\ z, w\ subClassOf\ y, x\ subClassOf\ z\}$. Here, $\Delta E = \{ins\{n\ subClassOf\ r\}, ins\{w\ subClassOf\ z\}, ins\{w\ subClassOf\ y\}, ins\{x\ subClassOf\ z\}\}$. Pruning this list will involve checking every entry to ensure that the subject or object does not occur in M in order to prune that triple from the list to be entered into the inference process. Of the four triples added in this example, all must be checked for pruning but only one triple $(ins\{n\ subClassOf\ r\})$ will be removed before the remaining three triples will enter the inference process.

In general terms, reduction strength appears to be a better indication of the differences between ΔE and ΔD_c than inference strength. Similar arguments apply to establishing the difference between ΔE and ΔED

6 Conclusion and Future Work

This paper describes a correction method for dense *deltas* that results in consistent update of RDF datasets. We have eliminated the need for conditions on the dataset

[2] $inference\ strength = \frac{|C(M)| - |M|}{|M|}$

by checking the antecedents of inferable triples in the insert set. If at least one such antecedent is found in the delete set then the inferable triple in the insert set cannot be removed from the *delta*. Otherwise, this triple can be safely removed from the *delta* to minimize its size.

A summary of our results is shown in Figure 13, which characterises the interaction between the degree of inference (i.e. the delete set and/or the insert set or no inference at all) and the approach to inferencing (i.e. inferring all triples or only necessary triples) and their combined impact on the *delta* size and computation time. It can be seen that ΔD_c has the smallest delta size compared to ΔED and ΔE. It can also be seen that the approach to inferencing affects the delta computation time. Figure 13 indicates that backward inference is more efficient (i.e smaller delta size and faster generation) than the other methods tested.

In this work we have investigated the effect of inference degree and inference approach on both the delta computation time and storage space over RDF datasets. Similar methods can be applied to ontologies that are represented in OWL 2. Here the RL rule set [5] is much richer than the rule set for RDFS with consequent potential for benefits to *delta* generation performance and size. Also, it is worth exploring different inference strengths to further evaluate the *delta* sizes and performance of the different approaches to producing these *deltas*. In particular while backward inference may be efficient, combining it with pruning may be expensive in terms of computation time where data is characterised by large inference strengths. Exploiting the inferred triples to infer new information may provide further improvements in update performance.

References

1. Al Azwari, S., Wilson, J.N.: The cost of reasoning with rdf updates. In: ICSC 2015. pp. 328–331. IEEE (2015)
2. Cloran, R., Irwin, B.: XML digital signature and RDF. Information Society South Africa (ISSA 2005), July (2005)
3. Hayes, P., McBride, B.: RDF semantics. W3C recommendation. World Wide Web Consortium (2004)
4. Im, D.H., Lee, S.W., Kim, H.J.: Backward inference and pruning for RDF change detection using RDBMS. J. Info. Science 39(2), 238–255 (2013)
5. Motik, B., Grau, B.C., Horrocks, I., Wu, Z., Fokoue, A., Lutz, C.: OWL 2 Web ontology language profiles, W3C Recommendation 11 December 2012 (2013)
6. Noy, N., Musen, M.: Promptdiff: A fixed-point algorithm for comparing ontology versions. AAAI/IAAI 2002, 744–750 (2002)
7. Papavasileiou, V., Flouris, G., Fundulaki, I., Kotzinos, D., Christophides, V.: High-level change detection in RDF(S) KBs. ACM Trans. Database Syst. 38, 1:1–1:42 (2013)
8. PaPavaSSiliou, S., PaPagianni, C., DiStefano, S.: M2M interactions paradigm via volunteer computing and mobile crowdsensing. In: Misic, V., Misic, J. (eds.) Machine-to-Machine Communications: Architectures, Technology, Standards, and Applications. pp. 295–309. CRC Press (2014)
9. Völkel, M., Groza, T.: SemVersion: An RDF-based ontology versioning system. In: Nunes, M., Isaas, P., Martnez, I. (eds.) Proc. IADIS Int. Conf. WWW/Internet. p. 44. IADIS (2006)
10. Zeginis, D., Tzitzikas, Y., Christophides, V.: On computing deltas of RDF/S knowledge bases. ACM Trans on the Web (TWEB) 5(3), 14 (2011)