Computationally Efficient IFC File Comparison Using RDF Graph Paths

Jyrki Oraskari, jyrki.oraskari@aalto.fi
Department of Computer Science, Aalto University

Abstract

We study computationally efficient methods calculating changes in IFC models that have been converted to RDF graphs. It is shown how to calculate difference sets that are small enough to be descriptive and usable for synchronization while still using limited computing resources. We present an algorithm that utilizes the neighborhood of an anonymous node to find a probable matching node pair in the second version of the model. By limiting the neighborhood only to the most informative paths in the RDF graph, a balance between the certainty of the matching pair and the size of the neighborhood can be found. The algorithm is compared with commonly available RDF diff tools: Jena, and RDFContextTools RDFC14Ner. When compared, our algorithm provided significantly better results.

Keywords: BIM, RDF, IFC, Diff

1 Introduction

It is common practice to use BIM (Building Information Modeling) tools (Eastman, et al., 2011). The tools have a fundamental role in enhancing the productivity and the quality of the design work, but the change management may require some manual work of designers.

All BIM tools have a native internal representation of the model, but they usually support exporting the model into a standard Industry Foundation Classes (IFC) format (ISO, 2013). The format can be converted into a Resource Description Framework (RDF) graph (Lassila & Swick, 1999) which allows us to use the W3C-specified Web of Data technologies, and enable us creating new ways of computing change differences in the models without the need for modifying the existing BIM tools.

The difference between two RDF graphs is typically called a diff that can be presented as the sets of removed and added triples (Berners-Lee & Connolly, 2004).

A diff between IFC model versions is not trivial. Rare objects are identified with Globally Unique Identifiers (GUIDs) (Leach, et al., 2005). Furthermore, a typical IFC export is presented in the IFC-SPF format (IFC STEP Physical File) (Pratt, 2001) where most objects are identified with line numbers that identify them only in the single file. They cannot be used to trace objects between versions since the line number identifiers are not shared between the IFC exports. It makes identifying the changed objects difficult.

When an IFC model is converted to RDF, each attribute reference of an IFC element is mapped into a triple using the IFC ontology. The conversion principles are described in (Pauwels, et al., 2011; Beetz, et al., 2014). Our approach is the same with two exceptions. We use blank nodes to convey the meaning of the document-local identifiers (Beckett & McBride, 2004), and the GUIDs of the graph are converted into Internationalized Resource Identifiers (IRIs) (Dürst & Suignard, 2005) which emphasizes their roles as the identifiers of the elements.

The result of the conversion, the IFC/RDF graph, has the same structure and content as the original IFC model had. They share the difficulty of calculating a diff. The result of the conversion, the IFC/RDF graph, has the same structure and content as the original IFC model had. In this study, our

objective is to find an algorithm that gives as near the optimal difference sets as possible in a practical memory consumption and CPU usage.

In this paper, we present ideas about how the diff algorithm that compares the consecutive IFC/RDF exports of a native BIM model can be designed to be scalable. The CPU and the memory aspects are considered.

The remainder of this paper is organized as follows. Section 2 reviews the related research. Section 3 introduces the Short Paths algorithm. In Section 4, the result of comparing the Short Paths algorithm with Jena, and RDFContextTools RDFC14Ner is shown. Finally, conclusions and ideas for future research are presented.

2 Related research

In (Carroll, 2002) Carroll concentrated on the problem of checking if two RDF graphs are isomorphic. The presented performance improvement for the basic isomorphism test was based on classifying blank nodes according to the number of connected triples where classes could be refined using the adjacent classifications. Furthermore, the same algorithms were extended to classify blank nodes by using the role they appear in the statements. This is employed in Jena (Carroll, et al., 2004).

In (Carroll, 2003) he presented the idea of converting RDF documents into a canonized form so that by using checksums it was possible to check if two RDF graphs represent the same facts. In Carroll's canonization, blank nodes were named depending on the order they are in the lexically ordered and N3 formatted (Berners-Lee & Connolly, 1998) triple set. An individual fill character was used so that the blank node identifiers could be ignored. Furthermore, annotations were used to guarantee that the sorting order of the triples would always be the same.

In 2004 Berners-Lee et. al. introduced the general idea of RDF delta in (Berners-Lee & Connolly, 2004). In that, the difference between two graph versions was defined as triple set subtractions. Insertions were deduced calculating G2-G1 and deletions using G1-G2. Weak and strong diffs were also defined. By a weak diff, they refer to a difference set that can only be applied to the datasets that they were calculated from. A strong diff can be used in a multitasking system and as a patch for an already changed dataset. Blank nodes were handled by ordering triples by the connected subject nodes that had URIs. That format was named pretty printing.

Then, Tummarello et. al. stated in (Tummarello, et al., 2005) that, since blank nodes are not addressable from outside, an RDF graph-subgraph of connected blank nodes should be addressed as a unit. The subgraphs are called Minimum Self-contained Graphs (MSG). An efficient synchronization algorithm that uses Carroll's canonization for signing subgraphs was introduced.

Auer and Herre put the idea further Thet presented an atomic change in (Auer & Herre, 2007). It conveyed the same idea as a changed MSG has, but they could be aggregated hierarchically to express how a series of changes are associated.

Our algorithm is presented in the next chapter. It extends the signature diffing method we presented originally in (Oraskari & Törmä, 2015).

3 Short Paths Algorithm

The basic ideas of the Short Paths algorithm (SP) are in most parts the same as shown in SPCA in (Oraskari & Törmä, 2015). However, the algorithm is redesigned to introduce the ideas of how to keep the calculation efficient and still scalable. It is enhanced to handle the problems raised when diffing large IFC graphs. The ideas could most probably be generalized, but the scope of this article is in the BIM and IFC domain.

SP uses a string-based collection of characteristic information of a blank node neighborhood to differentiate it from other ones. For a blank node and its environment, the information is collected from short triple paths that start from the node to the close neighborhood of the node. The paths can be presented as a group of strings. That characterizing information of the vicinity of the node is called the signature. An RDF graph and the limited set of paths are defined formally:

Definition 1: RDF graph: For mutually disjoint sets of IRIs (I), blank nodes (B) and literal (L). A graph G is set of triples (s p o) $\in I \cup B \times I \times I \cup B \cup L$.

Definition 2: Non-directional RDF Path: Given an RDF graph G, a non-directional RDF path T = (t_1, t_2, t_3, … , t_n) is a sequence of triples, where ∀i ∈ [1,n], t_i = (s_i,p_i,o_i) ∈ G, and ∀i ∈ [1,n -1], s_i=s_i+1 ∨ o_i=o_i+1 ∨ s_i=o_i+1 ∨ o_i=s_i+1.

Definition 3: The Limited Width Path Set LWPS(*x*, *max_path*) for a node x is defined as follows: LSWPS is a collection of non-directional RDF paths {T_1,T_2,T_3, …, T_m}, where ∀j ∈ [1,m],T_j is a non-directional path T = (t_1, t_2, t_3, … , t_n) such that t_1 = (*x*,p_i,o_i) ∈ G, the length of the path |Tj|≤ *max_path*, and ∀y ∈ nodes({T_1,T_2,T_3, …, T_m}), | (y, ?, ?) ∈ G|≤ MAXWIDTH ∧ | (?, ?, y) ∈ G|≤ MAXWIDTH.

Definition 4: Signature Path for a non-directional RDF Path T= (t_1, t_2, t_3, … , t_n) is a string that is composed of ordered set of triples (Y_1, Y_2, Y_3, … , Y_n), where each set Y_i corresponds t_i = (s_i,p_i,o_i) in the path.
Y_i = sort({(s_i rdf:type ?)∈ G}) + sort({(s_i ? l))∈ G ∧ l ∈ L})+ sort({(s_i ? 'B')| (s_i ? b) ∈ G ∧ b ∈ B}) + sort({('B' ? s_i)|(b ? s_i) ∈ G ∧ b ∈ B}).
Likewise, Signature(P) for a non-directional RDF path set P is a sorted collection of the Signature Path strings of the paths.

The positioning structure of an IFC graph describes how an element is located in space in relation to some other components. The chains of relative positioning form a relatively stable skeleton of the model. We use that to give a geometrical structure a generated unique identifier. It decreases the number of blank nodes and makes the corresponding islet of blank nodes smaller. Listing 1 shows the calculation steps. The IfcLocalPlacement nodes are given a checksum based on the data that constitutes the rotation/translation matrix at the placement. If the data is unique, an IRI can be generated (lines 14 -16).

Listing 1 The placement checksum calculation

```
1    a ← http://annotations/
2    Procedure Placement_checksums(Graph G)
3     Ga ← ∅ // Creates an empty annotation graph
4       Function traverse(Node n, current_sum)
5        sum ← sort(coordinates of relativePlacement)
6        sum ← MD5(sum + current_sum)
7        Ga ← Ga + (n a:pc sum)
8        for(x ∈ Πx(x placementRelTo n)do
9          traverse(x, sum)

10    N ←{x|(x type IfcLocalPlacement) ∈ G ∧ |(x placementRelTo ?)∈ G|=0}
11    for n ∈ N do
12      traverse(n, '')
14    temp ← {x, where |(? a:pc Πsum(x a:pc sum))|=1 in Ga }
15      For n ∈ temp do
16        n ← IRI(Πsum(n a:pc sum))
17   Return Ga
```

Since the signature is based on the information that is collected from the close vicinity of the node, we can expect that the existence of the triple pattern in a graph segment has a probability (Bose, 2002). The more signatures we form in the graph segment, the more probable is that there are more than one node neighborhoods that are given the same signature. This has the direct connection to scalability. The more blank nodes we have, the more probable it is that a small neighborhood is not enough to give a unique enough identity for a node. SP handles this by dividing the RDF graph into areas that have a unique identity. Thuss, the signature needs to find a differentiation only locally. When an RDF graph gets larger, the graph segments may still be of a reasonable size.

The idea of dividing the graph into areas is twofold. First, when the graph can be divided into areas that have unique identifiers, the needed information count to differentiate a node in an area is smaller that in the case where the whole graph is used. Second, since an RDF graph is usually

connected, we expect that a new blank node is attached to one or many nodes of an identity that can be referred to, i.e. GUIDs.

Furthermore, if we suppose that, in general, the objects of the graph triples point to elements that are dependent on the subject, we can transitively relay the dependency relation in the graph. That forms a tree structure in the graph. The paths be assumed quite stable, but the IRI at the beginning of the path set can be recognized in a deterministic way even if there are changes in the subgraph. Thus, only the set of the nearest IDs is used as an identifier for the graph area. Listing 2 shows the algorithm.

Listing 2 The division of the graph

```
1    a ← <http://annotations/>
2    Function CheckDependencies(Graph G)
3     Ga ← ∅ // Creates an empty annotation graph
4     N ← iri nodes of the graph G
5     for all x in N do
6        Ga ← Ga +  (x a:dep x) + (x a:len 0)
7     for maximum rounds
8      for all x in N do
9        N ← N − x
10       j ← Πl{(x a:len l)∈ Ga}+1.
11       for all o in (x ? o) ∈ G ∧ o ∈ B
12           i ← if exist (o a:len l)∈ Ga, l, otherwise ∞
13           if j<i do
14               Ga ← Ga − {(o a:len ?) ∈ Ga} +  (o a:len j)
15               Ga ← Ga − {(o a:dep ?) ∈ Ga}
16           if j<=i
17               N ← N + o
18               Ga ← Ga + {(o a:dep w) |(x a:dep w)∈ Ga }
19    Return Ga
```

As in SPCA in (Oraskari & Törmä, 2015), the bottom-up checksums are calculated to recognize the repeating structures of the graph. They are, for example, IfcColor, IfcPoint, and IfcMaterial. The generic function that is presented in Listing 3 works using a simple and repeated rule. First all literals are assumed to have the checksum (lines 4-5). Then a checksum is added to all nodes that are pointing to nodes that have a checksum already (lines 6-11). That is repeated maximum round of times. The checksums are used in the main algorithm (Listing 4, line 24) to give the duplicate structures the same identifier.

Listing 3 A function to find which IRIs a blank node is dependent to

```
1    a ← <http://annotations/>
2    Function Bottom-up_Checksums(Graph G)
3    Ga ← ∅ // Creates an empty annotation graph
4    for n ∈ L and n ∈ nodes of the graph G do
5       Ga ← Ga + (n a:bu n)
6    for maximum rounds
7      for n ∈ { n | (n ? o)∈ G ∧ ∀o ∈ {Π_key(key a:bu ?)})
8         sum ← sort(bu|{(o a:bu bu)∈ Ga ∧ (n ? o)∈ G})
9         sum ← sum + sort({(n ? l))∈ G ∧ l ∈ L})
10        Ga ← Ga + (n a:bu MD5(sum + sort({t|(n rdf:type t)∈ G})))
11    Return Ga
12
13
```

Listings 4 and 5 show the overall description of the Short paths algorithm. First data like the timestamps of the IFC Export are removed since it is presumed that they do not convey information about the actual change (Listing 4 line 1).

SP uses information of the neighboring triples of a blank node to generate a name that has a reasonably good probability matching the node with the corresponding one in the second version of

the graph. The larger the neighborhood is, the more easily the node can be given a unique name, but a large collection of triples used for naming makes the name prone to changes. To achieve the balance, only the close neighborhood of the node is used.

We use dynamic path lengths. It means that for those nodes that have not given a unique signature in the last round, they are given a new one using a step longer paths. The idea behind this is that, in contrast to SPCA, SP finds the path length for a node adaptively. So, the same value can be used for a wide range of graphs. Furthermore, since duplicate entries are more probably made unique nondeterministically so that they do not match a pair in the other graph, a new round with longer paths likely increases the chance for the node to get a match. For a larger graph, the paths are automatically longer when needed. However, the maximum values set a limit for computation that is not exceeded for a node. The upper limit forces the computation time to follow a linear path.

On the other hand, the algorithm keeps the triple neighborhood of only the active node and just the minimum amount of data of the whole graphs in the memory. For this reason, when comparing huge models, the triple count and the minimum quantity of annotations dominate the memory consumption. For that reason, the memory consumption is linear for the large models.

The unique checksums are made into IRIs in line 18 and removed from the node-set in line 17. Lines 6-20 are repeated until a maximum path length is achieved.

To make the triple subtraction possible, the nodes that had given a name that shared with other nodes in a graph, is given a unique naming by adding an index number (lines 22-28).

Listing 4 Short Paths blank node naming algorithm

```
1    G ← G − {(x ? ?) ∪ (? ? x)| {x|(x rdf:type IfcOwnerHistory)}}
2    Ga ← Placement_checksums(G)
3    Ga ← Ga + CheckDependencies(G)
4    Ga ← Ga + Bottom-up_Checksums(G)
5    N ← blank nodes of G
6    For plen=1 to MAXPATH do
7       For all nodes n ∈ N do
8          P ← LWPS(n,plen)
9          P ← filter out paths of P that contain a cycle
10         csum ← MD5(Signature(P))
11         tmp ← empty list
12         For all points where paths in P cross do
13             tmp ← tmp + (position in the paths + path)
14         csum ← MD5(csum+sort(tmp))
15         csum ← MD5(csum+sort(Π_dep(n a:dep dep))) // the area id
16         csum ← csum + Π_pc(n a:pc pc) // the position information
17         Ga ← Ga + (n a:csum csum)
18      temp ← {x, where |(? a:csum Π_csum(x a:csum csum))|=1 in G }
19      For n ∈ temp do
20        n ← IRI(Π_csum(n a:csum csum))
21      N ← N − temp
22      if N is empty stop the loop
23   For all nodes n ∈ N do
24         for M← { m | (m a:csum Π_csum(n a:csum csum)) in G}
25            if all m in M have the same Π_bu(m a:bu bu)
26               m ← IRI(Π_csum(m a:csum csum))
27            else
28               m ← IRI(Π_csum(m a:csum csum)+ index)
29            N ← N - M
```

The actual SP algorithms is:

Listing 5 Short Paths algorithm

```
1    G1 ← Short Paths blank node naming algorithm for graph G1
2    G2 ← Short Paths blank node naming algorithm for graph G2
3    diff ← { G1 - G2, G2 - G1}
```

3.1 The memory usage

The memory consumption of the algorithm is an important aspect when the scalability is considered. A checksum can be used as a compact digital fingerprint that effectively has the same capability to differentiate the blank node from other nodes. For a checksum, SP uses the MD5 message-digest algorithm that can be used to generate the 128-bit hash value of the signature, i.e. the sorted set of path strings. Although theoretically a checksum algorithm loses information, when MD5 is used, the uniqueness of the checksums is almost the same as the original data has.

When the memory consumption of the algorithm was tested using Open BIMServer datasets[1], the usage of the MD5 checksumming saved notable amount of Java heap memory. The results can be seen in Table 1. The memory usage was measured generating an extra full garbage collection in Java and writing down the usage after the clean-up.

Table 1 The memory consumption

Datasets	SP with MD5	SP without MD5
FJK Project Final	9,105K	15,370K
PART02 Wilfer 200302 20070209	95,848K	2,027,109K
Export datasets 1 and 3	39,611K	927,741K

4 Results

The software that was needed for the experiment was written in Java (Gosling, et al., 2005). The IFC- to RDF conversion follows the principles described in (Pauwels, et al., 2011). However, IRIs are used for those entities of IFC-SPF that have a GUID, and blank node are used to represent other entities. The ontology of the test data reflects the IFC specification of the exported dataset. It is based the ISO/PAS 16739 standard and can be expected to be stable. Therefore, in this study, only the RDF data is diffed.The implementation details, the source code of the experiments, and the datasets are available in our GitHub repository[2].

In the experiment, we used a previously crafted series of changes of a small model to test the algorithms. For these, Tekla Structures (Tekla Corporation) and Graphisoft ArchiCAD were used. The test sets were enriched with Solibri Model Checker (SMC)[3] models and the *export* datasets from Open BIMServer (Beetz, et al., 2010). They were used since they are publicly available and consist of a series that can be diffed. SMC models are useful so that it describes the structure of a reasonably large building, and it is one of the rare publicly available IFC models that are given versions. So, all the models and changes are created using a BIM tool.

The dataset statistics are shown in Table 2. In the data, the ArchiCAD and Tekla Structures datasets are the smallest ones. They also have a relatively high large number of IRIs. The data were created to serve as the primary benchmark of an IFC diff. The original IFC data and converted RDF/N3 datasets are available in the code repository. Solibri Model Checker (SMC) example datasets are larger and more complex. They represent a whole office building and contain a realistic change. Thus, they are suitable for testing the scalability of the algorithms. Furthermore, the Open BIMServer *export* datasets represent a two-storey detached house. The sets are relatively large and contain very few GUIDs.

Table 2 Statistic of the data used in the experiments

DataSet Name	Short name	Triples	RDF Nodes	IRIs	Blank Nodes	Literals

[1] https://github.com/opensourceBIM/BIMserver/tree/master/TestData/data
[2] https://github.com/jyrkio/CIB_IFCDiff.git
[3] http://www.solibri.com/products/solibri-model-checker/

ArchiCAD 1	A1	706	470	13%	43%	44%
ArchiCAD 2	A2	1362	822	11%	47%	42%
ArchiCAD 3	A3	1372	849	10%	47%	43%
ArchiCAD 4	A4	1954	1212	8%	50%	42%
SMC 1	S1	1137184	615088	1.2%	69.5%	29,2%
SMC 2	S2	1135693	614328	1.2%	69.5%	29.3%
BIMserver Export 1	E1	41344	22865	0.7%	68.4%	30.9%
BIMserver Export 3	E3	41604	23017	0.7%	68.3%	31.0%
Tekla Structures 1	T1	6839	3340	5.9%	46.2%	47.9%
Tekla Structures 2	T2	7439	3648	5.8%	46.6%	47.6%

The comparison was made so that SP and RDFC14Ner provided RDF string sets that were subtracted. topodiff provided the same functionality. Jena created a new model as a result of a diff. So, the statement count of the difference set models were reported. The Jena Model difference method was used in the tests.

topodiff (Ragozin, 2015) was not usable without modification since it worked only if all of the blank nodes were attached to a unique literal. This was true when the original line numbers of the IFC entity were added as an object literal to the blank nodes that represented the entity in the RDF world. When the unique literals were added, the code was able to identify the blank nodes, and the sorting could take place. The results of the sorting were quite good. So, there was no reason to reject the algorithm from the experiment.

RDFC14Ner (Panaioli, 2005) is Fabio Panaoili's implementation of Carroll's Canonization algorithm. Its problem is that it adds triples to give a blank node the same identity each time for the same dataset. It is useful when one would like to prove that a dataset is not tampered. In this case, the IFC/RDF dataset versions are independent. There is no way to save the insertions to the native data and thus, they only increase the dataset size. That explains why a removal set can be larger than the whole original dataset.

The removal and addition difference set sizes are shown in Table 3 and Table 4. SP provides the smallest sets of removed and added triples in all but the case where the sets were isomorphic. topodiff gave the best results in the test where the same dataset was compared with itself. The result set sizes were clearly the second smallest. RDFC14Ner values were large because of the added annotations. Finally, Jena had consistent values, but the variation of the result set sizes is small. SP was the only algorithm that had small enough result set sizes so that it is practical in a synchronization or finding real changes in the datasets.

Table 3 Removed triples set sizes generated by the algorithms

Datasets	SP	topodiff	RDFC14Ner	Jena
A4-A4	1%	0%	25%	76%
A1-A2	11%	17%	86%	77%
A2-A3	12%	43%	80%	77%
A3-A4	3%	14%	86%	74%
S1-S2	16%	64%	100%	83%
E1-E3	25%	77%	107%	85%
T1-T2	50%	87%	89%	80%

Table 4 Added triples set sizes generated by the algorithms

Datasets	SP	topodiff	RDFC14Ner	Jena
A4-A4	1%	0%	25%	76%
A1-A2	21%	57%	96%	81%
A2-A3	19%	43%	80%	77%
A3-A4	22%	40%	94%	77%
S1-S2	16%	64%	100%	83%
E1-E3	26%	78%	107%	85%
T1-T2	51%	88%	90%	80%

The algorithms were tested using a PC that has Intel i7-3820QM 2.79Ghz CPU and 16 GB RAM. The used operating system is 64-bit Microsoft Windows 7 Enterprise. The CPU usage was tested using timestamps that were created using Java System.nanoTime(). The runtime was measured from the period where the actual comparison was made.

As Table 5 shows, in this experiment, SP and topodiff had similar computation times. RDFC14Ner was clearly faster in large datasets. Jena used the least amount of the CPU time. However, it does not compensate the poor performance when considering the accuracy of the results. The difference sets were inadequately large.

Table 5 Calculation time in seconds

Datasets	SP	topodiff	RDFC14Ner	Jena
A4-A4	0.27	0.29	0.27	0.05
A1-A2	0.10	0.17	0.14	0.05
A2-A3	0.17	0.21	0.20	0.04
A3-A4	0.21	0.24	0.23	0.03
S1-S2	208.19	214.40	60.92	8.33
E1-E3	11.43	2.88	2.61	0.35
T1-T2	0.51	0.60	0.64	0.07

The memory consumption was estimated taking a snapshot of the Java heap memory allocation at the time when the change detection was ready, and the data structures of the diff algorithm were still referenceable. The statistics are presented in Table 6. In the experiment, the memory consumption was in line with topodiff, and RDFC14Ner. Jena used the least amount of memory, which is understandable since SP and RDFC14Ner use Jena as a tool reading the triples into memory. All algorithms had all the graph data in memory all the time. topodiff handles the N3 presentation of the triples directly. For that reason, we had expected it to use less memory in general. However, topodiff uses more memory when the datasets get larger.

Table 6 Java heap memory consumption in MB

Datasets	SP	topodiff	RDFC14Ner	Jena
A4-A4	9.4	5.1	8.5	7.6
A1-A2	8.4	2.9	7.6	7.1
A2-A3	8.8	3.7	7.9	7.3
A3-A4	9.1	4.4	8.2	7.5
S1-S2	1,096.0	2,624.4	1,217.0	651.6
E1-E3	43.1	94.3	46.0	29.3
T1-T2	11.9	17.3	13.5	10.1

5 Conclusions

The paper shows the effectiveness of the RDF graph neighborhood-based algorithms to detect the changes in IFC derived RDF graphs. The presented algorithm provides an accurate low-level tool for

diff IFC/RDF graphs, which allows us using it as an enabler in the Linked Building Data toolset in the DRUMBEAT project. It can be used as a core for a version management system, synchronization tools, change analysis and notification streams. However, the results are domain specific. The generalization of the ideas needs more extensive analysis of the existing RDF datasets.

The Short Paths algorithm provides changesets that are notably smaller than commonly available RDF diff tools: topodiff, Jena and RDFContextTools RDFC14Ner offer. As the difference set size is a crucial aspect in notification systems, in this aspect, SP excels in the comparison.

Since all the systems keep the RDF graphs in the memory during the change detection, the memory consumption is the major limiting factor when considering diffing larger models. The tools did not differ much from each other's, but the memory consumption of the Short Paths algorithm was better than the average when diffing the largest models.

The time usage of all the algorithms was usable for practical considerations. Less than 4 minutes time is quite reasonable when diffing a model of the size of an office building. Nevertheless, as only 12% of the CPU resources were used in the experiments, the considerate amount of optimizing the CPU usage is possible.

These results are the first steps of exploiting the benefits of change detections in parallel design in the Web of Building Data. There are several lines of future work to continue this research. The Efforts of using random sampling from a collection of IFC/RDF models have been made with promising initial results. Optimizing the calculation provides us with much potential to study more. Concurrent processing and partial diffing are interesting research topics. However, our primary focus will be on integrating the diff results into the Linked Building Data toolset. Thus, how the changes can be meditated to linked building information models is one of the crucial questions. Visualizations and smart notifications may reduce the manual work of designers.

Acknowledgements

This research has been carried out at Aalto University in two research projects — DRUM "Distributed Transactional Building Information Management" (RYM/PRE 2010-2014) and DRUMBEAT "Web-Enabled Construction Lifecycle" (2014-2017) — both funded by Tekes, Aalto University, and the participating companies.

References

Auer, S. & Herre, H., 2007. A versioning and evolution framework for RDF knowledge bases. *Perspectives of Systems Informatics,* pp. 55-69.

Beckett, D. & McBride, B., 2004. RDF/XML syntax specification (revised). *W3C recommendation,* Volume 10.

Beetz, J., van Berlo, L., de Laat, R. & van den Helm, P., 2010. *BIMserver. org--An open source IFC model server.* s.l., s.n.

Beetz, J. et al., 2014. Interoperable data models for infrastructural artefacts--a novel IFC extension method using RDF vocabularies exemplified with quay wall structures for harbors. *eWork and eBusiness in Architecture, Engineering and Construction: ECPPM 2014,* p. 135.

Berners-Lee, T. & Connolly, D., 1998. Notation3 (N3): A readable RDF syntax. *W3C Team Submission (January 2008) http://www. w3. org/TeamSubmission,(3).*

Berners-Lee, T. & Connolly, D., 2004. *Delta: an ontology for the distribution of differences between RDF graphs.* s.l.:s.n.

Bose, R., 2002. *Information theory, coding and cryptography.* s.l.:Tata McGraw-Hill Education.

Carroll, J., 2003. Signing RDF graphs. *The Semantic Web-ISWC 2003,* pp. 369-384.

Carroll, J. J., 2002. Matching rdf graphs. In: *The Semantic Web-ISWC 2002.* s.l.:Springer, pp. 5-15.

Carroll, J. J. et al., 2004. *Jena: implementing the semantic web recommendations.* s.l., s.n., pp. 74-83.

Dürst, M. & Suignard, M., 2005. *Internationalized resource identifiers (IRIs),* s.l.: s.n.

Eastman, C., Teicholz, P., Sacks, R. & Liston, K., 2011. *{BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors}.* s.l.:John Wiley \& Sons.

Gosling, J., Joy, B., Steele, G. & Bracha, G., 2005. *Java(TM) Language Specification, The (3rd Edition) (Java (Addison-Wesley)).* s.l.:Addison-Wesley Professional.

ISO, 2013. *Industry Foundation Classes (IFC) for data sharing in the construction and facility management industries,* Geneva, Switzerland: s.n.
Lassila, O. & Swick, R. R., 1999. Resource description framework (RDF) model and syntax specification.
Leach, P. J., Mealling, M. & Salz, R., 2005. A universally unique identifier (uuid) urn namespace.
Oraskari, J. & Törmä, S., 2015. RDF-based signature algorithms for computing differences of IFC models, In Press. *Automation in Construction,* June.
Panaioli, F., 2005. *RDFC14Ner.* s.l.:s.n.
Pauwels, P. et al., 2011. A semantic rule checking environment for building performance checking. *Automation in Construction,* 20(5), pp. 506-518.
Pratt, M. J., 2001. Introduction to ISO 10303 - the STEP standard for product data exchange. *Journal of Computing and Information Science in Engineering,* 1(1), pp. 102-103.
Ragozin, A., 2015. *topodiff - RDF model oriented diff algorithm - Google Project Hosting.* s.l.:s.n.
Tekla Corporation, n.d. *Tekla Structures BIM software | Tekla.* s.l.:s.n.
Tummarello, G., Morbidoni, C., Puliti, P. & Piazza, F., 2005. *Signing individual fragments of an RDF graph.* s.l., s.n., pp. 1020-1021.