IFCdiff: A content-based automatic comparison approach for IFC files

Xin Shi[a], Yu-Shen Liu[a,b,c,*], Ge Gao[a], Ming Gu[a], Haijiang Li[d]

[a] *BIM Research Group, School of Software, Tsinghua University, Beijing, China*
[b] *Key Laboratory for Information System Security, Ministry of Education of China*
[c] *Tsinghua National Laboratory for Information Science and Technology*
[d] *BRE Institute of Sustainable Engineering, Engineering School, Cardiff University, UK*

Abstract

As the usage of IFC (Industry Foundation Classes) files in construction industry is on the dramatic increase, it often requires effective IFC comparison methods to keep track of important changes occurring during the lifecycle of construction projects. However, most IFC comparisons are based on a visual inspection, a manual count and a check of selective attributes. Although a few techniques about automatic IFC comparisons have been developed recently, they are usually very time-consuming, and are sensitive to the GUID change or redundant instances in IFC files. To address these issues, this paper presents a content-based automatic comparison approach, named *IFCdiff*, for detecting differences between two IFC files. This approach starts with a comprehensive analysis of the structure and content of each IFC file, and then constructs its hierarchical structure along with eliminating redundant instances. Next, the two hierarchical structures are compared with an iterative bottom-up procedure instead of the original files. The presented approach fully takes into account the content of IFC files fully without the need of flattening instances in IFC files. In contrast with previous methods, our approach can greatly reduce the computational time and space, and the comparison result is not sensitive to re-

*Corresponding author at: School of Software, Tsinghua University, Beijing 100084, China. Tel.: +86 10 6279 5455; Mobile: +86 159 1083 1178. URL: `http://cgcad.thss.tsinghua.edu.cn/liuyushen/`

Email addresses: `coolstone712@126.com` (Xin Shi), `liuyushen@tsinghua.edu.cn` (Yu-Shen Liu), `gg07@mails.tsinghua.edu.cn` (Ge Gao), `guming@tsinghua.edu.cn` (Ming Gu), `lih@Cardiff.ac.uk` (Haijiang Li)

dundant instances in IFC files. Finally, we demonstrate a potential application to incremental backup of IFC files. The software can be found at: http://cgcad.thss.tsinghua.edu.cn/liuyushen/ifcdiff/.

Keywords: Building Information Modeling (BIM), Industry Foundation Classes (IFC), IFC comparison, Change detection, Similarity and difference

1. Introduction

During the last decade, Building Information Modeling (BIM) has received a considerable amount of attention in the domain of Architecture, Engineering and Construction (AEC) to support lifecycle data sharing [1]. As an open and neutral data format specification for BIM, Industry Foundation Classes (IFC) [2] plays a crucial role to facilitate interoperability between various software platforms. The IFC data format has been widely supported by the market-leading BIM software vendors. Many recent studies also demonstrate the IFC viability in various applications, such as evaluation of design solutions [3], virtual construction [4], construction management [5], model checking [6, 7], path planning [8], file optimization [9], semantic annotation [10] and information retrieval [11].

As the usage of IFC files in construction industry is on the dramatic increase, it often requires an effective IFC comparison method to keep track of important changes occurring during the lifecycle of construction projects. The IFC comparison aims to analyze and identify the differences and similarities between two IFC files. It is a fundamental problem which may arise in many BIM-based applications, such as collaborative building design [12], incremental backup of files, construction project management [5], product data exchange [13, 14, 15], conformance checking [14], handover for operation and maintenance [14]. Previous IFC comparisons are usually based on a visual inspection, a manual count and a check of selective attributes [14, 16, 17, 18]. However, due to the large file sizes and the complex inheritance and referencing relationships of IFC files, such a way of manual inspection is often time-consuming and error-prone; furthermore, it can only report a partial and illustrative view of the compared files [13]. Although a few recent studies have been developed for automatic IFC comparison [13, 17, 19], their methods are usually very time-consuming, and are sensitive to the globally unique identifiers (GUID) change [17, 19] or redundant instances [13] within IFC files.

To address these issues, this paper proposes a content-based automatic IFC comparison approach, named *IFCdiff*, for tracking differences or detecting changes between two IFC files. Our approach starts with a comprehensive analysis of structure and content of each IFC file, and then constructs its hierarchical structure along with eliminating redundant instances at each level. Next the two hierarchical structures are compared with each other for detecting changes in an iterative bottom-up procedure. Our approach fully takes into account the content of IFC files and makes good use of the hierarchical structure of IFC files. Thus, our approach can greatly reduce the computational time and space, and the comparison result is not sensitive to redundant instances within IFC files. In addition, we also demonstrate a potential application of our approach to incremental backup of IFC files.

The paper is organized as follows. Section 2 reviews the related work and summarizes the existing problems. Section 3 introduces some basic concepts and terms of IFC files. Section 4 gives a detailed description of our approach. Section 5 demonstrates the experimental results and a potential application to incremental backup of IFC files. Finally, Section 6 concludes this paper, summarizes our contributions and discusses some future work.

2. Related work

Early studies of IFC comparison mainly conducted a visual inspection of models and a check of selective attributes in the original and exchange models [14, 16, 17, 18]. The visual inspection can be done with various IFC viewers that are available, while the attribute analysis is usually a manual check for building elements. However, only using a visual and manual way for comparing IFC files is inaccurate and incomplete due to the complex referencing and inheritance structure of IFC files [13]. The manual way is useful for only small and simple IFC models, whereas it is not practical for large and complex models in the actual construction projects. Consequently, there is an urgent need for developing automatic IFC comparison tools in the scenario of IFC-based data management.

2.1. Plain text comparison

There are various approaches in use for performing automatic comparison of one IFC file to another. An IFC file is a plain text (ASCII) format with the extension "*.ifc", which is specified by IFC and ISO 10303-21 [20] (also known as "STEP physical file"). Therefore, a direct approach is to use plain text comparison tools for directly comparing two IFC files, regardless of information content of models. Some widely used text comparison tools [21] such as *diff*, *DiffMerge*, *cmp*, *FileMerge*, *SVN*, *CVS* and *BCompare*, can be conducted for this purpose. These tools usually compute the longest common subsequence and highlight the differences between files. However, pure text comparison does not consider specific data organization and representation of an IFC file which includes a complex referencing and inheritance structure. Therefore, the traditional text comparison tools are not suitable for IFC file comparison.

2.2. GUID-based IFC comparison

Another class of approaches is based on the globally unique identifier (GUID) [17, 19] which is an unique identifier for object instances across applications and systems. The GUID-based comparison criteria is as follows. If there is an instance in one IFC file which has the same GUID as an instance in another IFC file, they can be considered as the same instance; otherwise, they are considered as different even with the same attributes of the entity or of its reference entities. The GUID-based comparison is widely adopted by many commercial BIM softwares such as *Autodesk Revit*, *Navisworks* and *Graphisoft ArchiCAD*. Some research articles [19, 13] also discussed how to use the GUIDs for measuring the differences between IFC files. More recently, Oraskari et al. [22] presented RDF-based signature algorithms for computing differences of IFC models, but their algorithms is still closely related to the usage of GUIDs.

However, in the IFC specification, only the entities inherited from *IfcRoot* has a GUID as one of its attributes, while many other entities (e.g. *IfcPropertySingleValue* which are *IfcPropertySet*) not inherited from *IfcRoot* have no GUID [19, 13]. In addition, the GUIDs of instances are often changed during the data exchange between different systems even without any modification to the model itself. Therefore, the GUID-based comparison is not a reliable approach to distinguish two IFC files even if it is quite simple and fast for comparison.

2.3. Graph-based IFC comparison

A third type of approaches was suggested by Arthaud and Lombardo [12] in the co-design scenario, which compares two oriented graphs generated by two IFC files. From this, it is possible to track the differences between two IFC models. However, the matching process of nodes between two oriented graphs still complies with the GUID comparison, where the instances without GUIDs are ignored in the comparison process. Such a graph-based IFC comparison is a non-trivial and time-consuming task for large models. Furthermore, this approach does not handle duplicate data instances in IFC files. In practice, the IFC files generated by various software platforms often include a large number of duplicate data instances [9, 13], which should be processed in the process of IFC comparison. We will discuss this issue in Section 2.5.2 in detail.

2.4. Flattening-based IFC comparison

The fourth type of approaches, presented by Lee et al. [13], utilizes a recursive strategy to flatten the instances in two IFC files, and then compares the flattened data instances instead of the original ones. The "flattening" process is to replace all the reference numbers with their actual values in each IFC file, which makes an IFC file into a structure that does not include any referencing or inheritance structure [13]. This overcomes the difference of reference numbers included in attribute values when comparing pairs of data instances. As a result, IFC comparison is simplified to pure string comparison after flattening.

This flattening-based method firstly reads two IFC files and parses data into instance name, entity name, and attribute values before comparing. In the following example of one data instance, #90 is the instance name, IFCSLAB is the entity name, and the remaining information within parentheses is the attribute values.

#90=IFCSLAB(2VLPPLMIR7fBUKZN0XN2MZ, #13, SLAB_006, , $, #335, #320, $, .FLOOR.)

Since different BIM modeling systems might export IFC files in various ways. As a result, the instance names and reference numbers might be different. To overcome this difference in referencing mechanisms, the files should be "flattened" first, i.e., making files in a structure that does not include any referencing or inheritance structure by replacing the reference identifier numbers with their actual attribute values. The following shows the flattened data instance of #90.

#90=IFCSLAB (2VLPPLMIR7VLPPLMIR7fBUKZN0XN2MZ, $, UNDEFINED, $, $, $, $, $, $, $, ORGANIZATIONNAME, $, $, $, $, GS, GRAPHISOFT, GRAPHISOFT, $, $, 9.0, A-CAD9.0, ARCHICAD, $, .NOCHANGE., $, $, $, 1149148841, S-LAB_006, , $, $, (0.,0.,0.), (0.,0.,1.), (1.,0.,0.), (0.,0.,0.), (0.,0.,1.), (1.,0.,0.), (_43500.,14500.,_200.), (0.,0.,1.), IFCPARAMETERVAL-UE(0.)), ((0.,0.), IFCPARAMETERVALUE(90.)), .T., .CARTE-SIAN.), .F., (0.,0.,0.), (0.,0.,1.), (1.,0.,0.), (0.,0.,1.), 200.)), $, .FLOOR.)

Such a flattening process overcomes the difference of reference numbers included in attribute values when comparing pairs of data instances. As a result, IFC comparison is simplified to pure string comparison after flattening.

The process of file comparison in [13] consists of three main steps: (1) first parsing all data instances and flattening them, and then (2) comparing the flattened instances while ignoring their GUIDs, finally (3) computing the similarity. One main advantage of the flattening-based comparison approach is that it is insensitive to the change of GUIDs of data instances in IFC files. However, the flattening-based file comparison is usually time-consuming for large models, and it is also sensitive to redundant instances appearing in IFC files. In addition, this approach does not deal with the order changes of the properties in property sets in data instances. For example, an data instance *IfcPropertySet* is given below.

#145=IFCPROPERTYSET('3wesF7dHX9B9kkD2hgAhST', #33, 'PSet_Revit', $, (#133, #134, #135, #136, #137, #138));

In the instance **#145**, the last attribute is a collection of attribute instances, i.e. **(#133, #134, #135, #136, #137, #138)**, in which each attribute instance (e.g. **#133**) is an *IfcPropertySingleValue* indicating an attribute value. In this collection, the order of these attribute instances might change during data exchange between different BIM software. Therefore, this requires a special treatment in the file comparison process. However, the flattening-based IFC comparison does not consider this situation, which may result in that the same data instances but with different orders of attributes are considered to be different.

2.5. Summarizing the existing problems

After reviewing the existing approaches [12, 13, 17, 19], we summarize the existing problems as follows.

2.5.1. Sensitivity for GUID changes

Although the GUID-based approach [17, 19] is simple and fast without comparing all attribute values, the GUIDs of data instances are often changed in data exchange from different systems. Therefore, it is not an appropriate way for identifying the differences between IFC files. The graph-based comparison [12] is time-consuming for large models, and it still complies with the GUID comparison during node matching. In addition, this approach is also sensitive to the redundant instances.

In contrast, our method compares the contents and structures of IFC files through an iterative procedure, which does not rely on GUIDs.

2.5.2. Sensitivity for redundant instances

Redundancy in information theory is the number of bits used to transmit a message minus the number of bits of actual information in the message. Informally, it is the amount of wasted "space" used to transmit certain data [9]. Many previous studies (e.g. [9, 13, 16, 19]) have introduced that the exported IFC files in practice often contain a large number of redundant information. Our recent paper [9] also illustrated several possible reasons for an abundance of redundancy in the exported IFC files. For instance, *differences of model mapping mechanism* between various BIM software platforms and the standard IFC data may produce a great deal of redundancy, and *various possibilities offered by the IFC specification* can cause redundancy too [9].

One typical example of redundancy in IFC files is the *identical data instances* [9, 13], which are roughly defined as multiple instances of the same entity with the same entity name and attribute values, but possibly with different instance names. For example, the duplicate instances of the *IfcCartesianPoint* entity with the same value are one common example of redundant information. The identical data instances are the representative of the redundancy that should be dealt with in the process of IFC comparison. Complying with information theory, our approach eliminates the problem of redundancy existing in IFC files before comparing in order to remove the influence on the similarity caused by the redundant instances.

The metric computation based on flattening the instances in [13] is sensitive to redundant instances in IFC files. In Eq. (1), the number of matching instances is highly relevant to the number of redundant instances in the IFC files. Assuming that there are a large number of duplicate instances in File A matched to data in File B, the similarity rate in Eq. (1) will be very high even close to 100%. This is unreasonable, because various IFC files (with or without duplicate instances) of the same building model should describe the same data model. A robust similarity rate computation should be insensitive to the number of redundant instances in IFC files. Although Lee et al. [13] also presented the *matching rate* for indicating how often instances in File A are redundantly produced in File B, it cannot improve the similarity rate computation in essence.

Being different with the flattening-based comparison approach, our approach constructs the hierarchical structures of IFC files along with eliminating redundant instances. Then the two hierarchical structures are compared with an iterative bottom-up procedure instead of the original IFC files. By removing the redundant instances while keeping the complete IFC models, the approach can overcome the influence arising by redundant instances in IFC files. Consequently, our approach can obtain a stable and reliable similarity rate compared with the flattening-based approach [13].

2.5.3. Time-consuming to calculation

The flattening-based approach [13] is also time-consuming for comparison of large IFC files. On the one hand, the comparison between a large number of duplicate instances existing in IFC files will take a lot of time; while it is in fact unnecessary. On the other hand, after all instances in an IFC file are flattened, the generated strings of flattened instances become quite long due to the complex referencing and inheritance structure of IFC. It will cost a lot of time and space to complete the process of instance matching. In general, the flattening process will increase the size of an IFC file several times or even dozens of times. For example, a 10M IFC file in our test cases is increased to 70M after flattening.

Compared with the flattening-based approach in [13], our approach avoids the procedure of flattening instances and is able to gain the similarity rate in a much shorter time. Furthermore, since the redundant instances are removed from the original files when using our approach, the number of data instances to be compared is decreased significantly. This greatly improves the comparison efficiency.

A more formal investigation is given in Appendix A for discussing the complexities of the mentioned algorithms.

2.5.4. Other issues

In the previous work, the order problem of aggregation attributes was not considered in the process of IFC comparison. In IFC, a lot of attributes are in the form of a collection of reference numbers. For example, the relationship object associates one object with several other objects or attributes, and these objects or attributes are recorded as reference numbers in a set. Another example is the property set which includes some reference numbers and each of them stands for one property. Since different systems export data in different ways, the order of aggregation attributes might change during data exchange. However, the previous approaches (also including flattening-based approach [13]) regard this case, i.e. that those instances with the same attribute sets but in different order, as different instances. In contrast, our approach handles the order problem of attributes, which produces a stable similarity rate.

The GUID-based and flattening-based approaches mainly focus on textual comparison between two IFC files. However, since the readability of IFC text file is poor, it is non-trivial for users to find the differences between geometric models only through text comparison. In fact, each IFC file includes geometric information which represents a 3D building model. If the textual comparison results can be associated with the 3D model, it will enable users to intuitively understand the differences and changes between models. This paper develops a prototype *IFCdiff* viewer specifically designed to highlight the different geometric objects between models.

2.6. Tree compression

In computer science, tree compression (or named tree compaction) is a common task and well-studied. Given a tree, the task is to map it as compactly as possible to memory [23], where the range of the mapping depends on specific applications. Many methods such as arithmetic coding and Huffman coding can be used for encoding and decoding of trees on data compression. There have been some typical applications of the tree compression methods such as the compression of pixel trees, syntax compression of program files, and the compression of XML document trees [24].

In this paper, we simplify each IFC file as a tree structure and remove the redundant data instances from this tree, which can be regarded as an application of tree compression to IFC files. Then, the compressed tree structures derived from two IFC files are compared instead of comparing the original IFC files. When the compression of tree structures is considered, two objectives are often involved. The first objective is to reduce the space needed for storing a tree itself, and the second one is to reduce the operation time on the specific application. Our method meets both of the requirements because the space can be saved through removing the redundant nodes while accelerating the functionality of the operations (i.e. IFC comparison).

3. Basic terms and IFC hierarchical structure

This section introduces some basic terms used in this paper and IFC hierarchical structures.

3.1. Basic terms used in the IFC file

As the ISO 16739 standard, IFC defines a conceptual data schema and an exchange file format of building information models. An IFC data file is in an ASCII text format with the extension "*.ifc", which uses the STEP physical file structure according to ISO 10303-21 [20]. The IFC file is composed of a *header section* and a *data section* [25], as shown in Figure 1. The header section describes basic information including the file description, the date and time, the schema version, etc. The data section defines the BIM data including a large number of *entity instances* (or named *data instances*). Each entity instance takes "#" as the beginning of the sentence and has *instance name*, *entity name* and a list of *attribute values*. The instance name (e.g. "#3967") is unique within the scope of an IFC file, which can also be used as a *reference id* cited by other entity instances. An example IFC file is shown in Figure 1, where some basic terms are illustrated.

Note that the instance names in two IFC files are independent of each other, so they cannot be used as a feature to distinguish two data instances. In our approach, the entity name and attribute values are considered for instance comparison.

3.2. Hierarchical structure of the IFC file

IFC divides all entities into *rooted* and *non-rooted* entities. Rooted entities derive from the most abstract class *IfcRoot* and each one has a GUID along with attributes. Non-rooted entities have no GUID, and data instances only exist if referenced from a rooted data instance directly or indirectly.

The IFC data model is essentially constructed in a hierarchical structure, generally with the rooted entity *IfcProject* as the *root node*. This structure is named *IFC hierarchical structure* in this paper. The data instances (e.g.

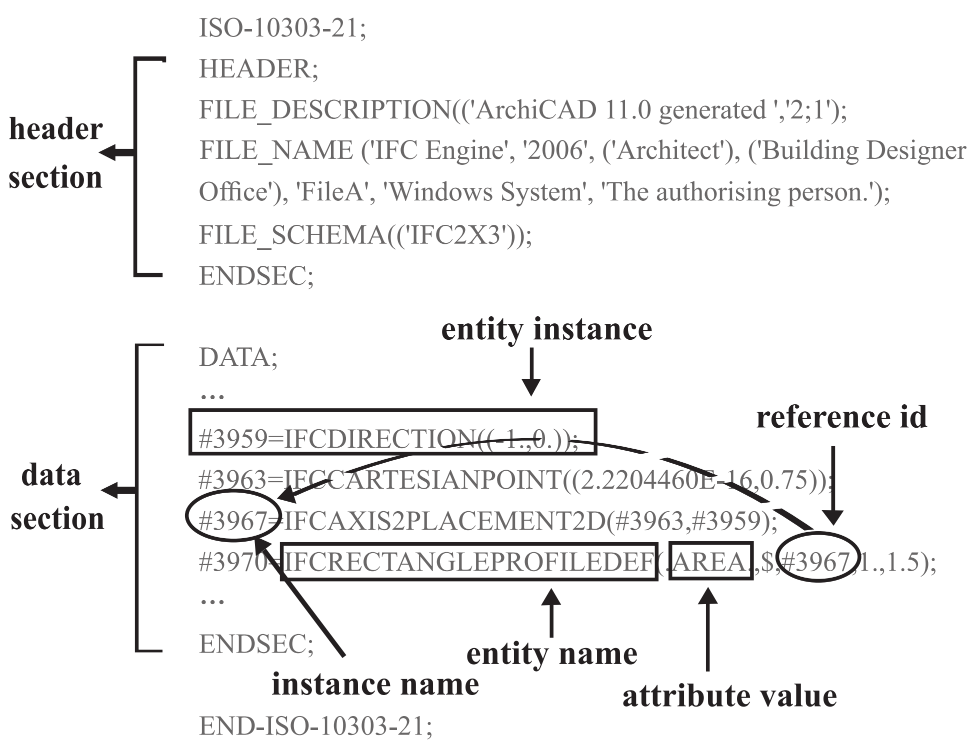


Figure 1: The basic terms used in the IFC file.

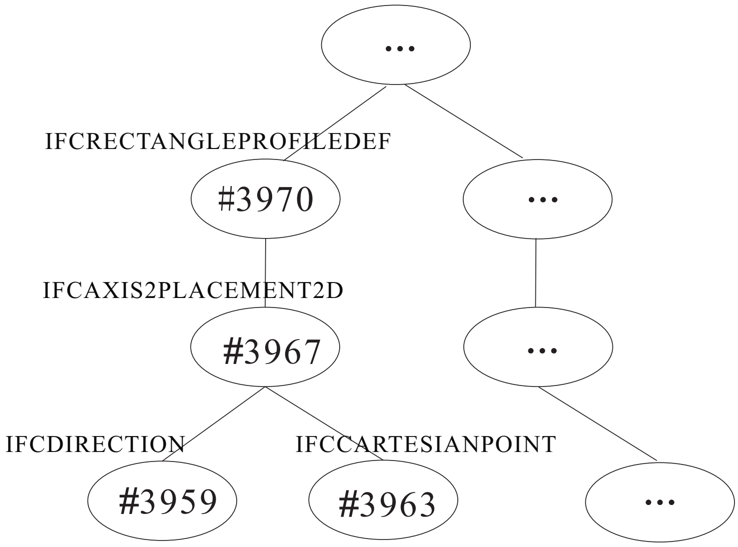


Figure 2: A partial IFC hierarchical structure corresponding to the file fragment in Figure 1, where each node also indicates its corresponding entity name.

non-rooted entities *IfcDirection* and *IfcCartesianPoint*) that do not include any reference id in their attributes are considered as the *terminal nodes*, or level 0. The data instances that directly cite the level 0 nodes are their *parent nodes*, or level 1. Consequently, the data instances are structured as the level n nodes, if they are the parent nodes of level $n-1$. The similar hierarchical representation of IFC file was also used in other IFC-based applications including IFC compression [9] and partial model extraction [26].

Figure 2 shows a partial IFC hierarchical structure corresponding to the file fragment in Figure 1. In Figure 2, the data instances (e.g. "#3959" and "#3963") are recognized as the terminal nodes (i.e. level 0), whose parent node is the data instance "#3967" (i.e. level 1). The data instance "#3970" (i.e. the parent node of "#3967") is recognized as the level 2.

4. The content-based IFC comparison approach

To achieve a fast and redundancy-insensitive IFC comparison, this section introduces a content-based automatic comparison approach for detecting changes between two IFC files. By analyzing the content of each IFC file, the approach first constructs the IFC hierarchical structure along with eliminating duplicate data instances. Then these two hierarchical structures are compared with an iterative bottom-up procedure. The main procedure of our approach is illustrated in Figure 3. Starting with two IFC files as input, our approach contains four steps as follows.

Step 1: Preprocess the data instances and construct the IFC hierarchical structures (see Section 4.1). This step first removes redundant information in each data instance, and then extracts three basic terms(i.e. instance name, entity name and attribute values) from each data instances. Next, based on the extracted terms and their referencing relationships, the hierarchical structures of two IFC files are constructed for further comparison.

Step 2: Compare the terminal nodes between two IFC hierarchical structures along with removing redundant instances (see Section 4.2). This step first identifies and groups identical data instances in the terminal nodes. Then only one data instance of each group is kept, while all other duplicate instances are removed from this group. Next, we compare the updated terminal nodes between two hierarchical structures and find the matching instances between them.

Step 3: Repeat Step 2 for iterative and level-by-level comparison for the remaining data instances between two files (see Section 4.3). Step 3 is a recursive and iterative process terminated until the comparing nodes reach the root node in any one of two files. The matching instances between two files are recorded in a hash table.

Step 4: Finally, compute the similarity rate between two IFC files (see Section 4.4). The similarity rate is defined as the rate of the number of matching instances between two files divided by the total number of instances in the target file. In addition, all matching instances between two files are saved in the hash table, and the differences between files are also recorded for further applications (e.g. incremental backup of IFC files in Section 5.6).

For the reader's convenience, the target file and the source file will be referred to as **File A** and **File B** in this paper, respectively.

4.1. Step 1: Preprocess data instances and construct the IFC hierarchical structures

We preprocess data instances within each input IFC file, and then contruct the IFC hierarchical structures. The preprocessing will remove redundant information (e.g. blank spaces and multi-lines) from each data instance

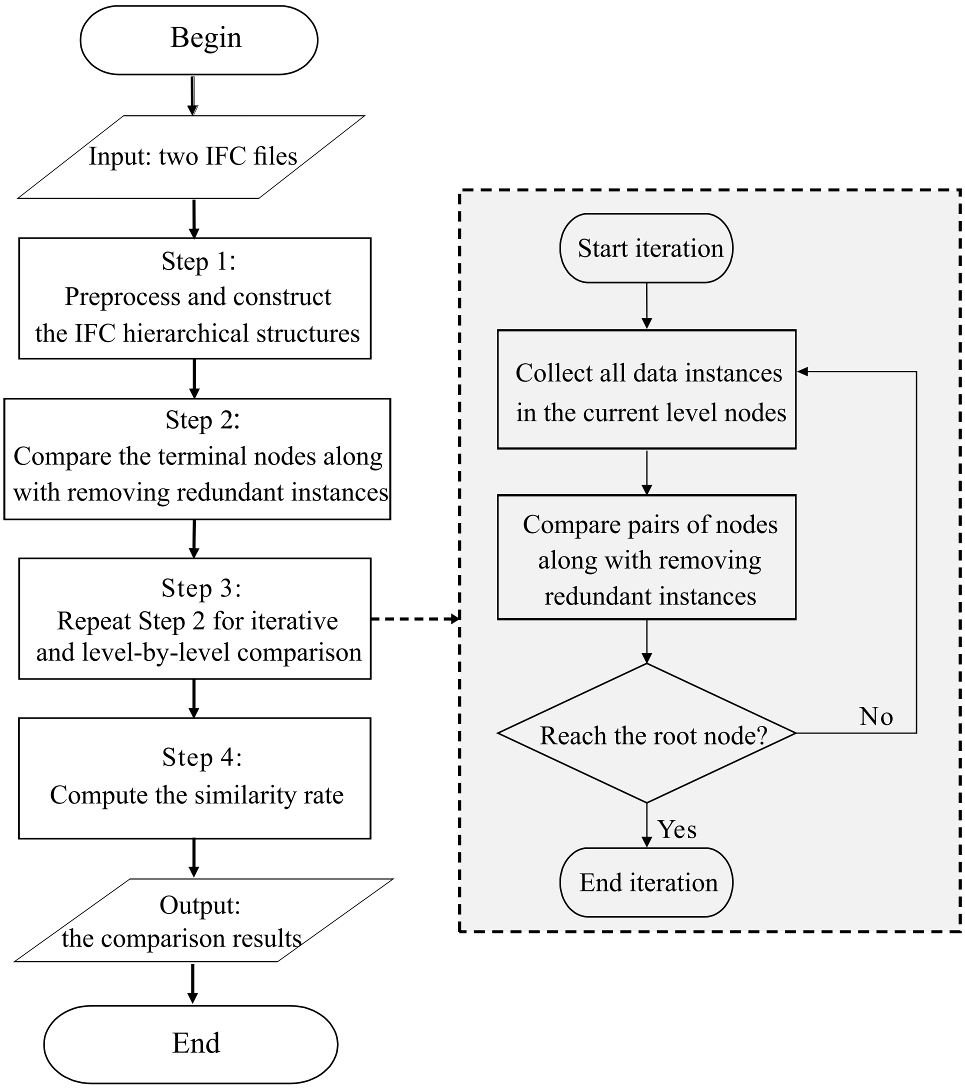


Figure 3: The flow diagram of our content-based IFC comparison approach.

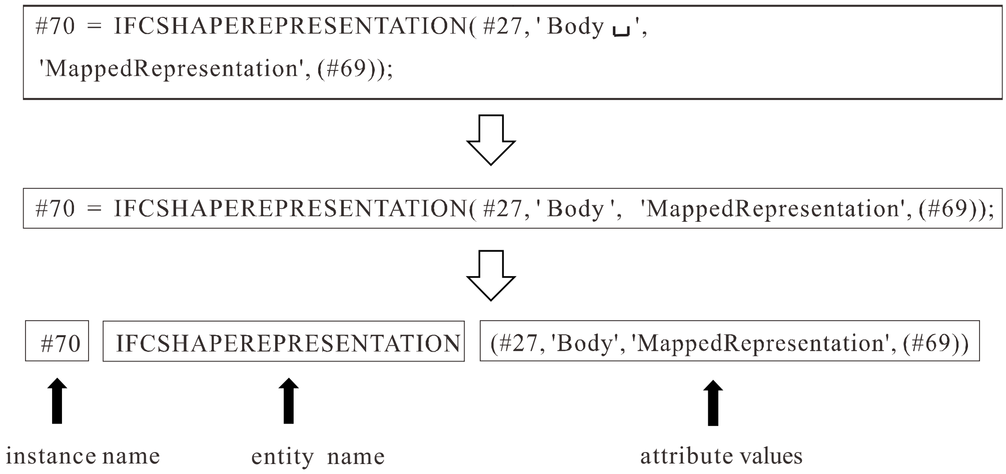


Figure 4: Illustration of preprocessing data instances within each IFC file, where three basic items will be extracted from each data instance.

within each IFC file. Especially, the data instance with multi-lines is converted into a single line. Then, the contents of three basic terms (i.e. instance name, entity name and attribute values) are extracted from each data instance, as shown in Figure 4. The above preprocessing is similar to the strategy in [9]. Finally, based on the extracted terms and their referencing relationships, the hierarchical structure of each IFC file (mentioned in Section 3.2) is constructed for further comparison.

Figure 5 shows an example of two file fragments from the target file (File A) and the source file (File B), respectively. The corresponding IFC hierarchical structures are displayed in Figure 6.

4.2. Step 2: Compare the terminal nodes along with removing redundant instances

The constructed IFC hierarchical structure is a tree-like data structure, so the comparison of two structures can be conducted in a similar way of level-by-level comparison of two trees. To accomplish this goal, we need to traverse two trees simultaneously, where this traversal visits the nodes by levels from bottom to top. The second step of our approach is to compare the terminal nodes between two hierarchical structures along with removing redundant instances.

Firstly, for each file, we collect all data instances on the terminal nodes which do not include any reference id in their attribute values. For example,

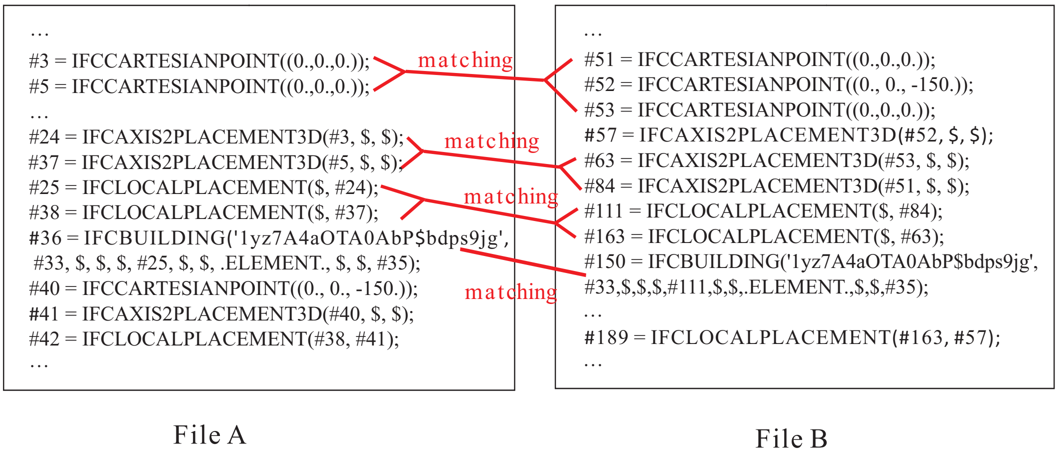
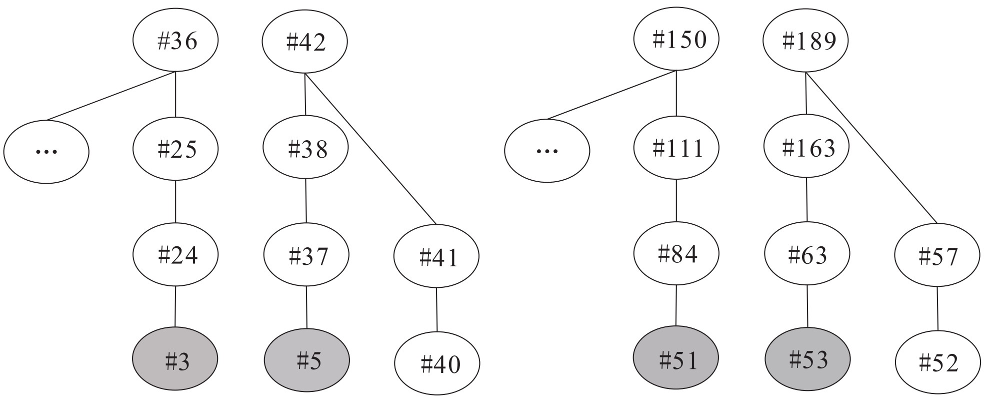


Figure 5: Illustration of two file fragments from the target file (File A) and the source file (File B), respectively. The matching instances between two file fragments are highlighted.



(a) File A (b) File B

Figure 6: The two partial IFC hierarchical structures are displayed, which correspond to File A and File B (in Figure 5), respectively. Here the identical data instances (light gray nodes) on the terminal nodes (i.e. level 0) in each file are identified and grouped.

in File A, the data instances (#3, #5 and #40) are recognized as the terminal nodes in the hierarchical structure (see Figure 6(a)). In File B, the data instances (#51, #53 and #52) are the terminal nodes (see Figure 6(b)).

Secondly, for each file, we identify and group the identical data instances in the terminal nodes by comparing their entity names and attribute values, where the terminal nodes with the same value are clustered into one group. Consequently, we can obtain multiple groups of identical data instances. In Figure 6, the light grey nodes (#3 and #5) denote one group of identical data instances in File A, while #51 and #53 are grouped together in File B.

Thirdly, only one data instance of each group is kept, while all other duplicate instances are removed from this group. Meanwhile, the reference id of attribute values in the remaining data instances in the upper levels will be updated accordingly. As shown in Figure 7, the data instance #3 in the group is kept in File A, while #5 is deleted. Meanwhile, their upper parent nodes (i.e. #24 and #37) are respectively relocated to the data instance #3. File B is processed similarly, where #51 is kept and #53 is removed. After achieving the above process, we can remove all redundant data instances from the terminal nodes both in File A and in File B.

Finally, we compare the new terminal nodes (without duplicate instances) between the two hierarchical structures while ignoring the GUIDs, and find the matching instances between them. Since the data instances in the terminal nodes do not include any reference id, we only compare the values of data instances (i.e. their entity names and attribute values) in terms of string comparison. In Figure 7, as for the terminal nodes, #3 in File A is matched to #51 in File B, where #3 and #51 are the same as "IFCCARTESIANPOINT((0.,0.,0.))" by checking the original file fragments in Figure 5. In addition, we record the pair of matching instances (#3, #51) in a *hash table* (denoted by T) for the upper level comparison.

4.3. Step 3: Repeat the iterative comparison process

In a similar way to Step 2, we need to traverse all nodes of two IFC hierarchical structures by levels from bottom to top. Therefore, we make use of an iterative strategy for comparing the nodes of each level along with removing duplicate instances. The procedure of iterative comparison is described as follows.

Firstly, for each IFC file, these data instances which directly cite the terminal nodes mentioned in Section 4.2 are collected, which will be treated as the new terminal nodes instead of the previous ones. As shown in Figure

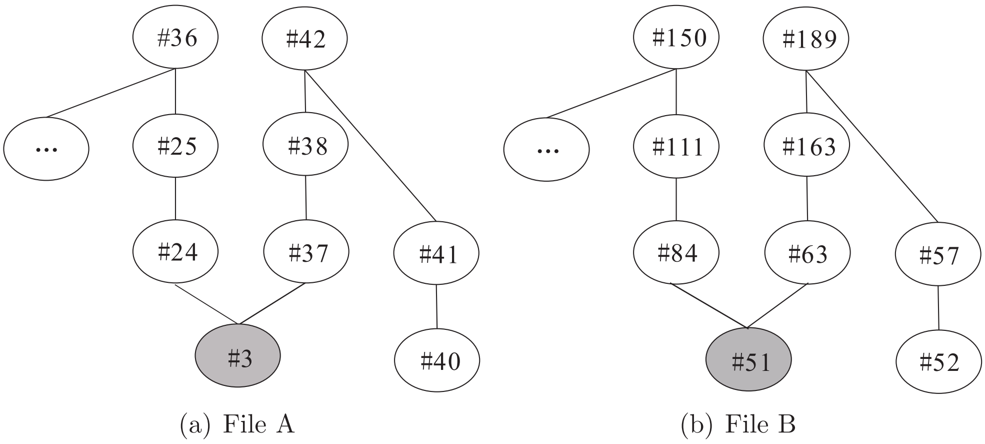
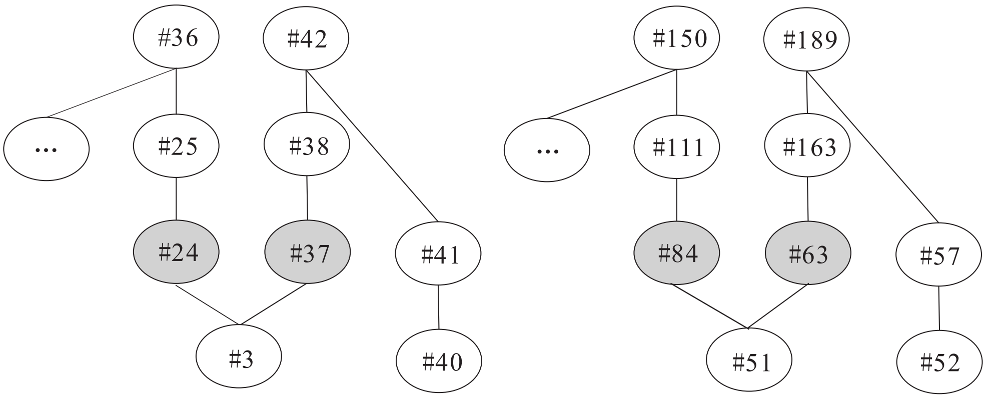


Figure 7: Remove the duplicate instances from the terminal nodes (i.e. level 0) in each file, and update the reference id of their upper parent nodes. (a) #3 is kept in File A, while #5 is removed. (b) #51 is kept in File B, while #53 is deleted.

8, the data instances (#24, #37 and #41) on the level 1 in File A become the new terminal nodes instead of the previous terminal nodes (#3 and #40), while the data instances (#84, #63 and #57) in File B are regarded as the new terminal nodes instead of #51 and #52.

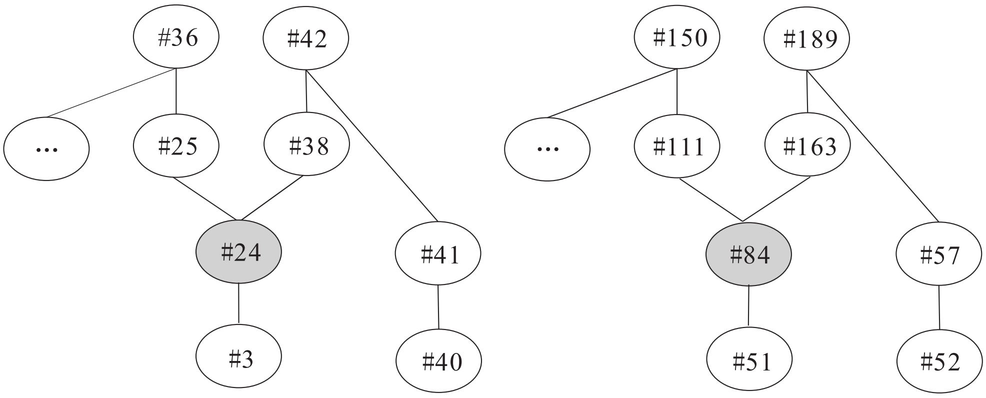
Secondly, in a similar way to Step 2, we group the identical data instances in the new terminal nodes (i.e. level 1), and then the duplicate instances are removed from this level in each IFC file. In Figure 8, the light gray nodes (#24 and #37) denote one group of identical data instances in File A, while the light gray nodes (#84 and #63) are another group in File B. After removing the duplicate instances, the data instance #24 is kept in File A, while #84 is kept in File B, as shown in Figure 9. Meanwhile, the upper parent nodes (i.e. level 2) are updated accordingly (see Figure 9), where the data instances (#25 and #38) are relinked to #24 in File A and the data instances (#111 and #163) are relinked to #84 in File B.

Thirdly, we compare the new terminal nodes (i.e. level 1) between two hierarchical structures while ignoring the GUIDs, and find the matching instances between them. Since the data instances in the level 1 include the reference id in their attribute values, we first compare the reference id and then compare the remaining properties between the two data instances. In Figure 9, the data instance #24 in File A and #84 in File B are accordingly updated as follows.



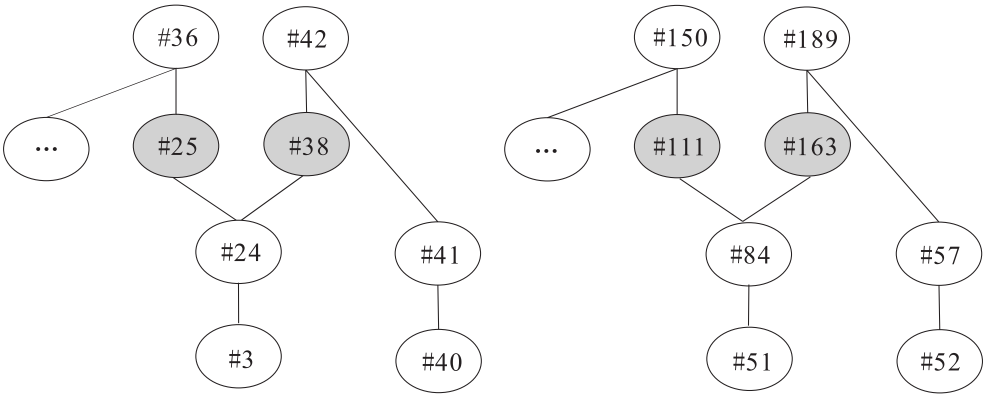
(a) File A (b) File B

Figure 8: Illustration of the iterative comparison process on the level 1. Here we group the identical data instances (light gray nodes) in the level 1 in each file. (a) The nodes (#24 and #37) denote one group in File A. (b) The nodes (#84 and #63) are in one group in File B.



(a) File A (b) File B

Figure 9: Illustration of the iterative comparison process on the level 1. Here we remove the duplicate instances from the nodes of level 1 in each file, and update the reference id of their upper parent nodes. (a) #24 is kept in File A, while #37 is removed. (b) #84 is kept in File B, while #63 is deleted.



(a) File A (b) File B

Figure 10: Illustration of the iterative comparison process on the level 2. Here we group the identical data instances (light gray nodes) in the level 2 in each file. (a) The nodes (#25 and #38) denote one group in File A. (b) The nodes (#111 and #163) denote another group in File B.

```
#24=IFCAXIS2PLACEMENT3D(#3, $, $);
```

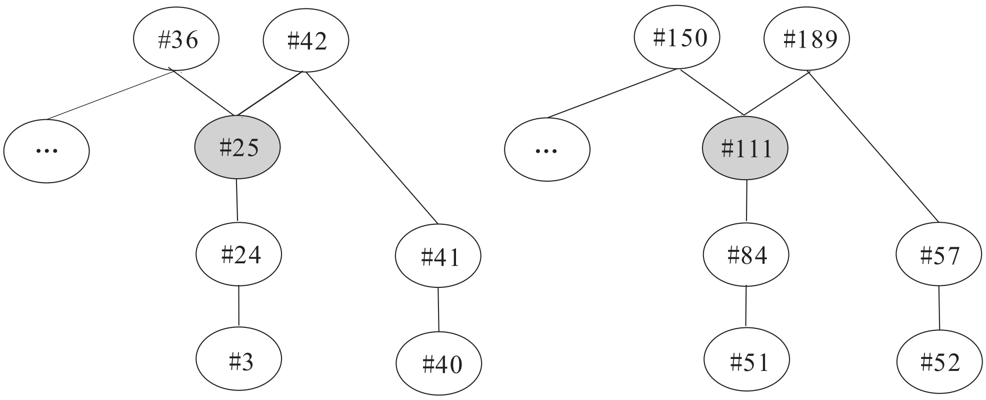
```
#84=IFCAXIS2PLACEMENT3D(#51, $, $);
```

Since we have recorded the matching instances (#3, #51) in the hash table T in the level 0 (see Section 4.2), the reference id of #24 is the same as the one of #84. In addition, the entity names and other properties between #24 and #84 are the same, so they are a pair of matching instances in the level 1. Meanwhile, this pair of matching instances (#24, #84) are continuously added into the hash table T for the upper level comparison.

We repeat the above procedure for further comparing the remaining data instances in two files. If the comparing nodes reach the root node (i.e. *IfcProject*) of File A or File B, the iterative comparison procedure is terminated. Figure 10 and Figure 11 show the above comparison procedure in the level 2. Figure 12 illustrates the pair of matching instances (#36, #150) in the level 3.

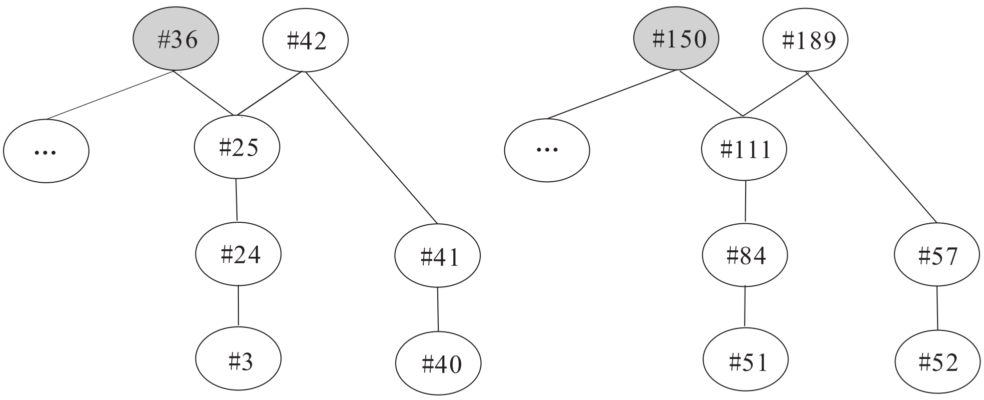
4.4. Step 4: Compute the similarity metric

The last step of our approach is to compute the similarity rate between two IFC files. Being similar to the similarity metric used in [13], we define the



(a) File A (b) File B

Figure 11: Illustration of the iterative comparison process on the level 2. Here we remove the duplicate instances from the nodes of level 2 in each file and update the reference id of their upper parent nodes. (a) #25 is kept in File A, while #38 is removed. (b) #111 is kept in File B, while #163 is deleted.



(a) File A (b) File B

Figure 12: Illustration of the iterative comparison process on the level 3. On the level 3, #36 in File A and #150 in File B are a pair of matching instances.

similarity rate from File A to File B as the rate of the number of matching instances between two files divided by the total number of instances in File A.

$$\text{Similarity}(A, B)(\%) = \frac{|A \cap B|}{|A|}, \tag{1}$$

where $|A|$ is the total number of instances in File A after removing redundant instances, and $|A \cap B|$ is the number of matching instances between File A and File B along with removing redundant instances using our approach. In contrast with the previous flattening-based approach in [13], when using our approach, the number of matching instances in File A compared to File B is the same with those in File B compared to File A, i.e. $|A \cap B| = |B \cap A|$, even if the input files include redundant data instances. Consequently, our approach can obtain a stable and reliable similarity rate.

As for the example of two file fragments in Figure 5 used in this section, the similarity rate from File A to File B [13] is 70.0% (7/10) based on the flattening-based approach, while the similarity rate is 57.1% (4/7) with our approach.

Finally, all matching instances between two files are saved in the hash table T, and the differences between them are also recorded for further applications (e.g. incremental backup of IFC files in Section 5.6).

Computational complexity

Let n and m ($n \geq m$) be the number of data instances in File A and File B, respectively. First, as for IFC hierarchical structures, it takes $O(n)$ (for File A) and $O(m)$ (for File B) to preprocess all the data instances in Step 1. Meanwhile, it takes $O(n \log(n))$ (for File A) and $O(m \log(m))$ (for File B) to remove the redundant data instances and update the reference id [9]. Finally, it takes $O(n \log(m))$ to compare pairs of data instances between two files with an iterative bottom-up procedure. As a result, the total complexity is about $2O(n) + 2O(n \log(n)) + O(n \log(m))$, and therefore an upper bound of running time is $O(n \log(n))$. A more detailed analysis for computational complexity is dependent on the two IFC hierarchical structures, and we leave it to the future work. In our implementation, we use the hash table to save the matching instances to accelerate the node searching and comparison.

4.5. Improvements of approach implementation

In order to address several issues mentioned in Section 2.5, we make some improvements for the presented approach.

(1) Ignoring the GUID change

During data exchange, initial GUIDs of data instances often get lost or changed. Therefore, we compare pairs of data instances while ignoring their GUIDs in Step 2 and Step 3 of our approach. This can overcome the effects of GUID changes during data exchange.

(2) Ignoring the change of owner history information

The owner history information (*IfcOwnerHistory*) contains information about the author, create time, modeling software and so on. This information will be changed whenever an IFC file is imported and exported from a system, even if there is no change in the model itself. Therefore, to identify the actual changes between two models, the owner history information is ignored in the comparison of instance attribute values.

(3) Ignoring the order change of property set

The previous comparison approach does not deal with the problem of the order changes of the properties in property sets. As mentioned above,the attribute of *IfcPropertySet* may be a collection of some attribute instances,which requires special treatment in the file comparison process.When comparing two *IfcPropertySet* instances, we compare all attribute instances in two collections and find the matching data instance with the help of the hash table T.

5. Experimental results and discussions

Our approach has been implemented in a content-based IFC comparison tool, called *IFCdiff*, with Visual C++ under Windows 8. All the experiments were run on an Intel Pentium(R) Dual-Core 3.06GHZ processor with 6 GB memory. Figure 13 shows the screenshot of the *IFCdiff* tool. The user should first open two candidate IFC files (the target and source files), and then click the button "**Compare**" to perform the comparison procedure. The comparison results and the similarity metrics are displayed at the bottom. Alternatively, one can select multiple checkboxes to ignore the GUIDs, owner history information and the order of property set.

In order to visualize the compared models and their differences, we also developed an *IFCdiff* viewer, as shown in Figure 14. In the main interface of the viewer in Figure 14(a), the corresponding differences of two input IFC files are highlighted in the text boxes in the middle, the similarity metrics

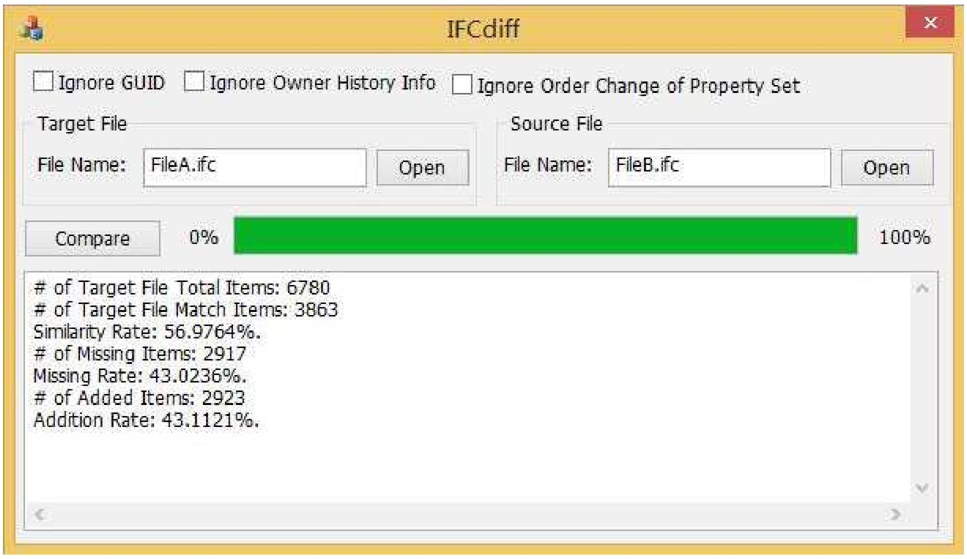


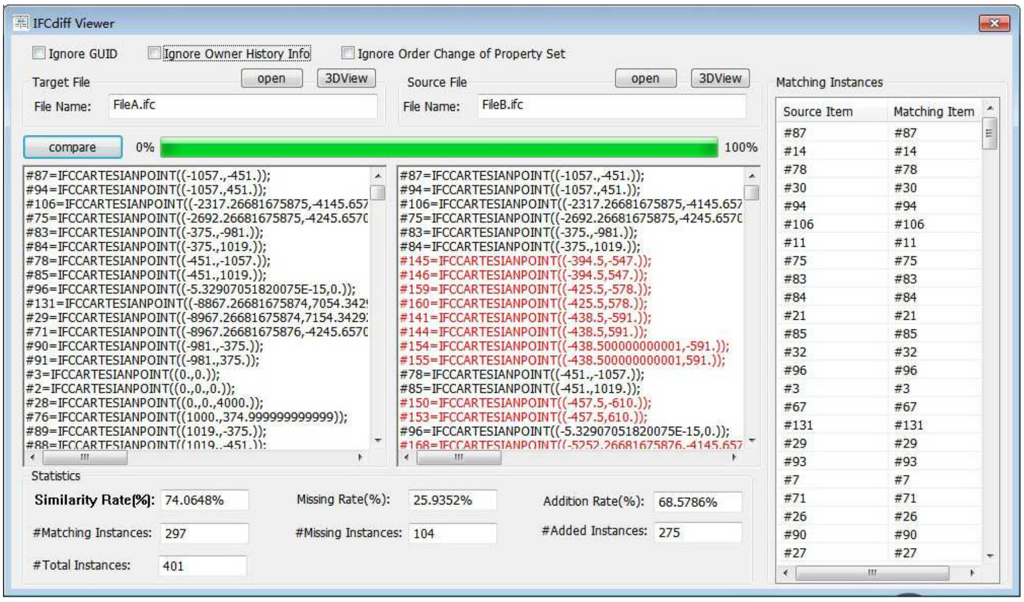
Figure 13: The screenshot of our comparison tool *IFCdiff*.

and a summary of the analysis are given at the bottom, and the matching data entities between two files are listed on the right. By clicking the button "**3DView**" of each file in Figure 14(a), the viewers of 3D models will pop up in Figure 14(b) and Figure 14(c), where the matching building elements are highlighted with the same color. This enables users to check the visual differences and changes between IFC models quickly.

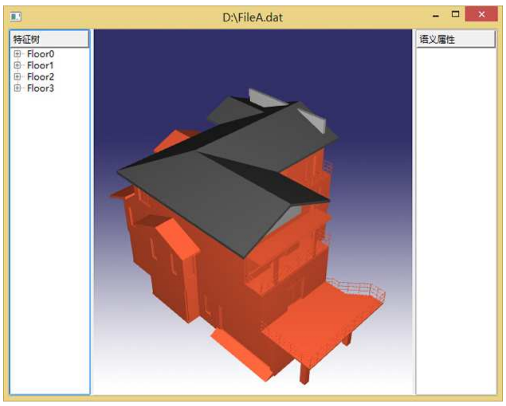
To evaluate the performance of the presented approach, this section tests our approach on some selected IFC files, and the experiments are conducted by comparing with other existing approaches. Finally, we demonstrate a potential application to incremental backup of IFC files.

5.1. Comparison with plain text comparison methods

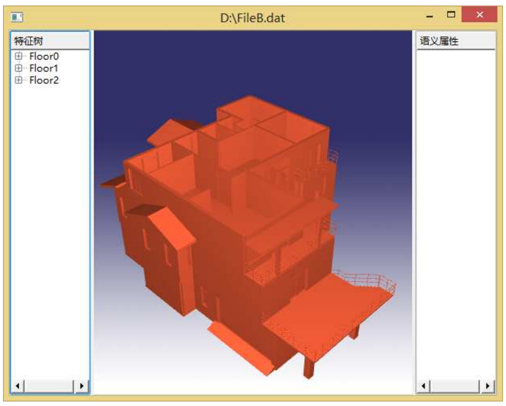
The first experiment compares our method with plain text comparison methods. Many plain text comparison tools [21] are able to achieve file comparison to highlight the differences between files. Such tools generally perform string comparison of string-by-string or line-by-line and highlight the differences and changes between two files. Here we typically choose the tool *DiffMerge* to compare two IFC files and show their differences. In the test case, a building model first was built in *Graphisoft ArchiCAD 16* and was exported as File A (Figure 15(a)). Then the same model was slightly modified



(a) The main interface of the *IFCdiff* viewer



(b) 3D model of File A



(c) 3D model of File B

Figure 14: The screenshot of the *IFCdiff* viewer.

```
#724= IFCPROPERTYSINGLEVALUE('IsExternal',$,IFCBOOLEAN(.T.),$);
#725= IFCPROPERTYSET('3vCh$0ythZzhLClJPLrosJ',#15,'Pset_WallCommon',$,(
#721,#722,#723,#724));
#727= IFCRELDEFINESBYPROPERTIES('260N_OFnjMqInhaDQjXR_Y',#15,$,$,(#509),
#725);
#731= IFCPROPERTYSINGLEVALUE('\S\6\S\(\S\N\S\;\S\O\S\_',$,IFCINTEGER(0),
$);
#732= IFCPROPERTYSINGLEVALUE('\S\5\S\W\S\2\S\?\S\F\S\+\S\R\S\F',$,
IFCLENGTHMEASURE(0.),$);
#733= IFCPROPERTYSINGLEVALUE('\S\R\S\Q\S\8\S\=\S\W\S\E\S\5\S\W\S\2\S\?',
$,IFCBOOLEAN(.F.),$);
```

(a) File A

```
#674= IFCPROPERTYSINGLEVALUE('IsExternal',$,IFCBOOLEAN(.T.),$);
#675= IFCPROPERTYSET('3vCh$0ythZzhLClJPLrosJ',#15,'Pset_WallCommon',$,(
#671,#672,#673,#674));
#677= IFCRELDEFINESBYPROPERTIES('260N_OFnjMqInhaDQjXR_Y',#15,$,$,(#509),
#675);
#681= IFCPROPERTYSINGLEVALUE('\S\6\S\(\S\N\S\;\S\O\S\_',$,IFCINTEGER(0),
$);
#682= IFCPROPERTYSINGLEVALUE('\S\5\S\W\S\2\S\?\S\F\S\+\S\R\S\F',$,
IFCLENGTHMEASURE(0.),$);
#683= IFCPROPERTYSINGLEVALUE('\S\R\S\Q\S\8\S\=\S\W\S\E\S\5\S\W\S\2\S\?',
$,IFCBOOLEAN(.F.),$);
```

(b) File B

Figure 15: The file comparison results using the plain text comparison tool *DiffMerge*.

Table 1: The number of data instances in test cases (M1 – M4).

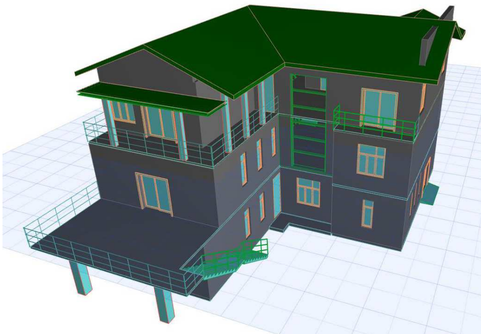
IFC files	#instances
M1 (with redundance)	106,438
M2 (without redundance)	45,461
M3 (with redundance)	103,541
M4 (without redundance)	44,931

through removing a window, and it was re-exported as File B (Figure 15(b)). Figure 15 shows the corresponding parts of file fragments with the same contents but with different instance names. For example, #724 in File A and #674 in File B are the identical data instances but with different instance names. *DiffMerge* recognizes that the two parts are totally different while highlighting their differences. The main reason is that plain text comparison methods cannot deal with specific data organization and representation of IFC files including the complex referencing and inheritance structures. In contrast, our approach recognizes the two parts as the same.

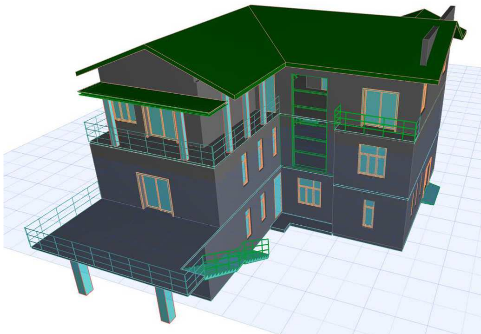
5.2. Comparison with the flattening-based file comparison method

The second experiment compares our method with the flattening-based file comparison method [13]. The used IFC files were exported through *ArchiCAD*, which are referred to as M1 – M4. The corresponding models are visualized in Figure 16. In the four test files, M1 contains a large number of duplicate data instances, while M2 is the non-redundant file obtained by removing the duplicate data instances from M1 (using our *IFCCompressor* tool [9]). M3 is obtained by deleting the roof of M1 in ArchiCAD and exporting the file, while M4 is the non-redundant file obtained by removing the duplicate data instances from M3. Table 1 shows the number of data instances in each IFC file. For instance, the original M1 file contains 106,438 instances, while the non-redundant M2 file just includes 45,461 instances.

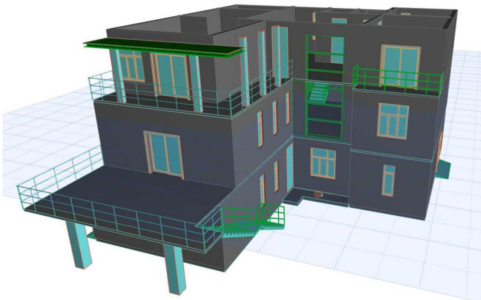
Table 2 shows the similarity rates computed using our method and the flattening-based method [13]. M1 (with redundance) and M2 (without redundance) are the same model but with a different number of data instances; similarly, M3 (with redundance) and M4 (without redundance) are the same model but with a different number of instances. In general, a robust approach of IFC comparison should be capable of obtaining a stable similarity rate between M3 (or M4) and M1 (or M2). The results in Table 2 suggest



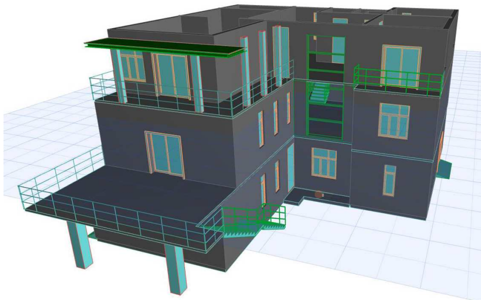
(a) M1 (with redundance)



(b) M2 (without redundance)



(c) M3 (with redundance)



(d) M4 (without redundance)

Figure 16: Visualizing the models of four test IFC files (M1 – M4). M1 contains a large number of duplicate data instances, while M2 is the non-redundant file through removing the duplicate instances from M1. M3 is the re-exported file after deleting the roof of M1 in *ArchiCAD*, while M4 is the non-redundant file through removing the duplicate instances from M3.

Table 2: The similarity rates computed using our method and the flattening-based method [13].

Target	Source	SR using our method[a]	SR using flattening[b]
M3	M1	83.2%	81.6%
M3	M2	83.2%	81.6%
M4	M1	83.2%	83.2%
M4	M2	83.2%	83.2%

[a] "SR using our method" is the similarity rate computed by our methods.

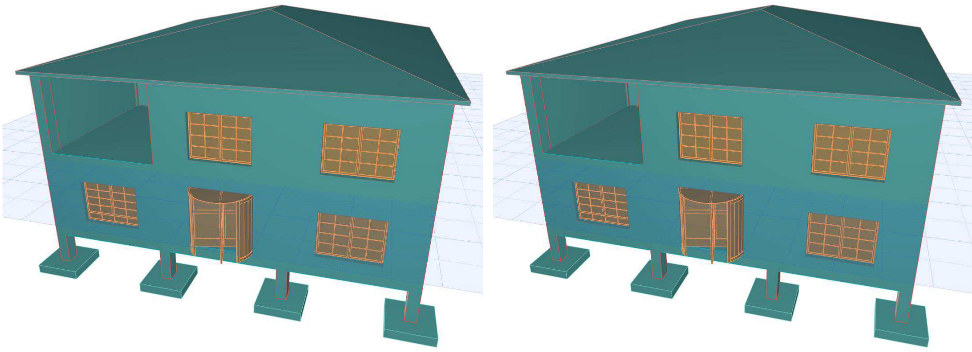
[b] "SR using flattening" is the similarity rate computed by the flattening-based method [13].

that our approach is not sensitive to redundant instances within IFC files, which can obtain the consistent similarity rate (83.2%) between M3 (or M4) and M1 (or M2). In contrast, the flattening-based method obtains two different similarity rates (81.6% and 83.2%), because of the redundant instances within IFC files. As for the flattening-based method, if there are a large number of duplicate instances matched in two files, the similarity rate tends to be high; otherwise, if there are a large number of duplicate instances not matched in two files, the similarity rate tends to be low.

5.3. Experiments under different parameter conditions

The third experiment compares our method under different parameter conditions. In Figure 13 and Figure 14, one can select multiple checkboxes to ignore the GUIDs, owner history information and the order of property set. This can explicitly improve the comparison results. We select two IFC files (referred to as M5 and M6) for this test, as visualized in Figure 17. A building model was first generated in *ArchiCAD* and then exported as M5. Next, M5 was imported into *ArchiCAD* and re-exported as M6 without any modification. Before performing file comparison, the duplicate instances of M5 and M6 have been removed using our *IFCCompressor* tool [9].

Table 3 shows the similarity rates using the *IFCdiff* with different parameters. The similarity rate without any specific parameters is about 85.85%, which is the same as the similarity rate in the flattening-based method [13]. The reason is that the two input files M5 and M6 have no redundant instances; consequently, the flattening-based method [13] can obtain the same result. In Table 3, the similarity rate with ignoring the GUIDs is also 85.85%,



(a) M5 (b) M6

Figure 17: Visualizing the models of M5 and M6. A building model first was generated in *ArchiCAD* and then exported as M5. Next, M5 was imported into *ArchiCAD* and re-exported as M6 without any modification.

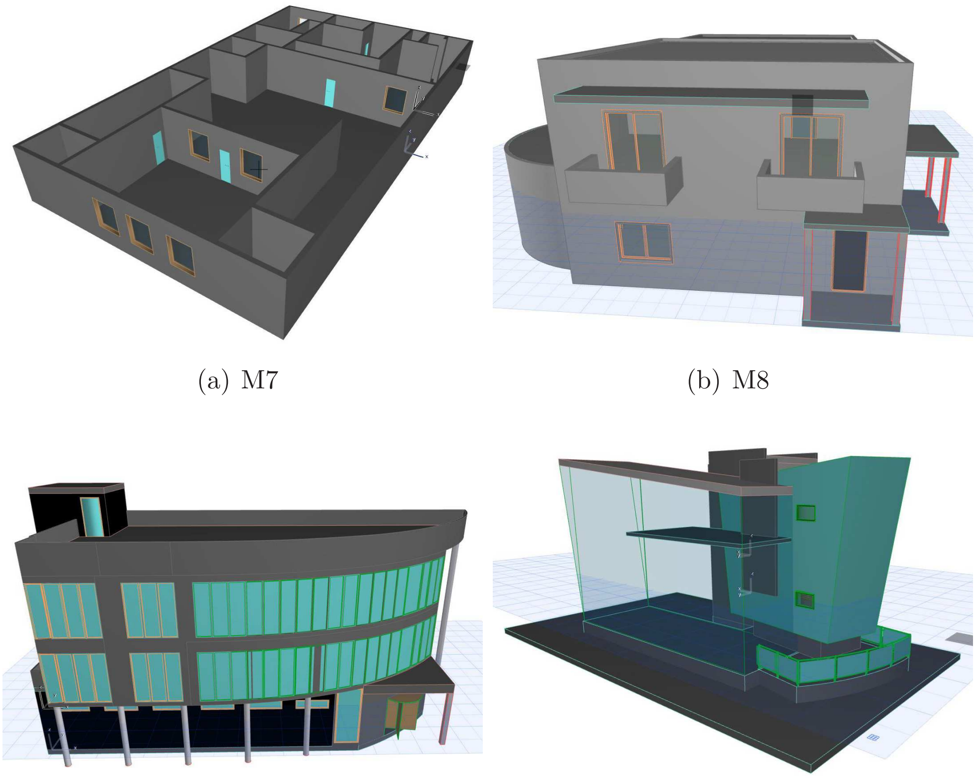
which indicates that *ArchiCAD* preserves the GUIDs well during data exchange. Another reason is that the building model was built on *ArchiCAD* and was exported as M5 and M6 still through *ArchiCAD*. In other words, the GUIDs were generated and maintained by the same system (i.e. *ArchiCAD*) itself. However, the GUID preservation rate often is low when a model is imported and exported in two different systems, as illustrated by Lee et al. [13].

In this table, the similarity rate with ignoring owner history information is about 95.12%, which is highest in this table. The reason is that a large number of data instances cite the entity *IfcOwnerHistory* which holds the modeler and modeling software information. The owner history information changes whenever a file is imported and exported from a system, even if no revisions are made to the model. Therefore, when ignoring the changes of owner history information, the similarity rate can be improved significantly.

Finally, we test the similarity rate while ignoring the order of properties in property sets, which is about 86.26% better than the default (i.e. 85.85%). This suggests that the orders of some attribute instances have been changed during the data exchange process, even if importing and exporting was done in the same system (i.e. *ArchiCAD*). The reason is the difference of model mapping mechanism between the internal models of BIM software platforms and the standard IFC data model, where the properties of some objects held in *ArchiCAD* are in different order to that in the IFC data model.

Table 3: The similarity rates using the *IFCdiff* with different parameter conditions.

Parameters	Similarity rate
Default (without any specific parameters)	85.85%
Ignore the GUIDs	85.85%
Ignore owner history information	95.12%
Ignore the order of property set	86.26%



(a) M7 (b) M8

(c) M9 (d) M10

Figure 18: Visualizing the four models (M7 – M10) used for testing computational time and space.

5.4. Computational time and space

5.4.1. Computational time

The fourth experiment compares computational time and space between our method and other methods [13]. In this test, four building models were developed in *Autodesk Revit 2014* and exported as the initial IFC files (referred to as M7 – M10), as visualized in Figure 18. Then we import the four files into *Revit* and *ArchiCAD*, and export them as new IFC files without making any changes to the models. The new IFC files are renamed M7_R, M8_A, M9_R and M10_A, where "_R" and "_A" denote that the files are exported through *Revit* and *ArchiCAD*, respectively. The new files are used as the target files, while the initial files are used as the source files.

Table 4: The details of paired IFC files used for testing computational time and space.

No.	Target files			Source files		
	Name	Size(MB)	#instances	Name	Size(MB)	#instances
1	M7_R	0.591	11,753	M7	0.425	8,287
2	M8_A	1.040	24,277	M8	1.257	26,234
3	M9_R	2.519	42,884	M9	3.511	68,114
4	M10_A	9.817	215,354	M10	4.364	91,023

Table 4 gives the details of those files to be compared, where "No." is the index of paired files to be compared, "Size(MB)" is the file sizes, and "#instances" is the number of data instances within the files. Note that although there is no modification in the imported and exported models, the file sizes and the number of data instances still suffer some changes. For example, M10 contains 91,023 instances, while there is a great increase of instances in the re-exported M10_A (215,354 instances). The main reason is that different systems map data into the IFC files in different ways.

Next, comparisons of paired files are made using the flattening-based method [13] and our approach, respectively. Table 5 shows the computational time of the two methods. In Ref. [13], the process of file comparison mainly contains three steps: (1) parsing data instances into the memory, (2) flattening all the instances and (3) comparing pairs of instances. In Table 5, we list the computational time of each step of the flattening-based method and the

Table 5: Computational time of two methods for the paired IFC files in Table 4.

No.	Flattening-based method				Ours	RT^a(%)
	Parse(s)	Flatten(s)	Compare(s)	$T_1{}^b$(s)	$T_2{}^c$(s)	
1	0.3276	2.0592	7.7377	10.1245	1.3728	86.44%
2	0.3276	2.0436	43.8987	46.8159	4.8048	89.74%
3	2.7924	4.6956	254.5	261.988	8.5333	96.74%
4	7.1605	135.861	1702.0	1845.02	81.2609	95.60%

[a] "RT" is the percentage of reduced time when using our approach.
[b] "T_1" is the total time of the flattening-based method.
[c] "T_2" is the time of our approach.

total time ("T_1"). In addition, the time of our approach is given by "T_2", and the percentage of reduced time is listed by "RT", where $RT = (T_1 - T_2)/T_1$. The result in Table 5 shows that our approach can significantly reduce the time in the file comparison process. For example, the percentage of reduced time is about 95.60% for the comparison of M10_A and M10. As mentioned in Section 2.5.3, the flattening-based method is often time-consuming for comparison of large IFC files, especially with numerous duplicate instances. It takes a lot of time to perform the two steps of flattening and comparing in [13]. In contrast with the flattening-based method, our approach has an advantage when dealing with large file comparison. In this experiment, for example, the flattening-based method costs 1845.02s for the comparison of M10_A and M10, while our approach just takes 81.2609s to process the comparison of the same files (reducing 95.60%). The result shows that the percentage of reduced time with our algorithm is generally very high (the average is 92.13%) for tested cases.

5.4.2. Computational space

In general, the flattening process in [13] also increases the size of an IFC file several times or even dozens of times. Table 6 shows the size of original target files in Table 4 and the size variation after running the flattening-based method and our approach. As seen in this table, when using the flattening-based approach, the sizes of some files are increased by more than ten times. In contrast, our approach reduces the file to a smaller size, which greatly improves the comparison efficiency.

Table 6: Space requirements of two methods in the file comparison process.

No.	Original[a](MB)	Flattening-based method		Ours	
		Flatten[b](MB)	Increase[c](%)	Process[d](MB)	Reduce[e](%)
1	0.591	33.292	5533.16%	0.466	26.82%
2	1.040	9.407	804.52%	0.722	44.04%
3	2.519	31.959	1168.72%	1.905	32.23%
4	9.817	70.175	614.83%	6.398	53.44%

[a] "Original" is the size of original target files in Table 4.
[b] "Flatten" is the size of space after flattening all instances in the target files.
[c] "Increase" is the percentage of increased space when using the flattening-based method.
[d] "Process" is the size of space after removing redundant instances using our approach.
[e] "Reduce" is the percentage of reduced space when using our approach.

In addition, the flattening-based method needs to use all the instances for comparison. Unlike that, since enormous duplicate instances are removed from the original files based on our approach, the number of actual instances used for comparison is significantly decreased. Table 7 lists the number of instances used for comparison based on the flattening-based method and our approach. The result shows that the average percentage of reduced instances using our approach reaches 25% for the tested cases.

Table 7: Counting the number of instances used for comparison based on the flattening-based method and our approach.

No.	Target files	N_1[a]	N_2[b]	Reduce[c](%)
1	M7_R	11,753	10,005	14.87%
2	M8_A	24,277	16,393	32.48%
3	M9_R	42,884	31,911	25.59%
4	M10_A	215,343	154,617	28.20%

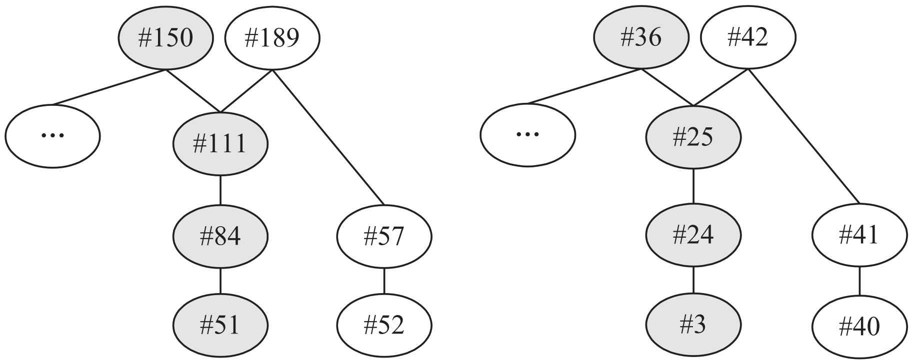
[a] "N_1" is the number of instances used for comparison based on the flattening-based method.
[b] "N_2" is the number of instances used for comparison based on our approach.
[c] "Reduce" is the percentage of reduced instances when using our approach.

Figure 19: Visualizing the IFC model used in a real-life case.

5.5. Preliminary test in a real-life case

In order to test the performance of our approach in a real-life case, an apartment building model in the Yunnan province in China is selected as a preliminary test. The architectural design model was developed in *Revit*, and exported as an IFC file. This selected model has been used for our previous case studies including IFC-based path planning [8] and IFC compression [9]. The original IFC file is about 156.0 MB, which includes more than 2.8 million data instances with numerous duplicate instances. The corresponding model is visualized in Figure 19.

In this case study, we first remove all duplicate instances from the original file using the *IFCCompressor* tool [9]. Then the newly non-redundant file is compared with the original file, which produces the similarity rate of 100%. The result suggests that our approach is not sensitive to redundant instances even in large IFC files. When using our *IFCdiff* tool, the time cost of comparison process is about 371.1s. In contrast, the flattening-based method fails to handle such large IFC files.



(a) File A (b) File B

Figure 20: An example for illustrating the incremental backup content, where File A is the previous version and File B is the current version. The identical data instances are highlighted by the light gray nodes, while the different data instances are the white nodes.

5.6. Application to incremental backup of IFC files

One potential application of our approach is for incremental backup of IFC files. An *incremental backup* is a type of data backup that backs up only the new or changed data since the last incremental backup. The design and management of building models follow an iterative process, which often includes frequent revision and updating on one or more basic models during the lifecycle of a construction project. This requires an effective method for IFC file backup.Time and disk space can be saved by only backing up the changed data.

The traditional *full backup* backs up all data on a disk even if minor changes are made to the files, which is time-consuming and space-intensive for IFC data management. Therefore, incremental backups are often desirable as they consume smaller storage space and are quicker to perform than full backups. Although pure text comparison methods can be directly used for incremental backup of IFC files, they cannot deal with specific data organization and representation of IFC files, as mentioned in Section 5.1. As a result, numerous consistent data instances (with different instance names) are considered to be different, and the size of incremental backup data is often close to the full backup.

The incremental backup mainly consists of identifying and recording the changed data since the last backup. Our comparison approach can be directly

```
…
#40 = IFCCARTESIANPOINT((0., 0., -150.));
…
#41 = IFCAXIS2PLACEMENT3D(#40, $, $);
…
#42 = IFCLOCALPLACEMENT(#38, #41);
…
```

Figure 21: Illustrating the differences between the previous and current versions, which are saved to record the changed data in incremental backup.

Table 8: Comparison of storage space between the full backup and our incremental backup.

Previous	Current	Full backup(MB)	Incremental backup(MB)	Reduce(%)
M2	M4	2.231	0.603	72.97%

applied to identify the differences between two IFC files. Then the differences are saved as the portion that has changed. Figure 20 shows an example for illustrating the incremental backup content, where File A is the previous version and File B is the current version. Here the identical data instances are highlighted by the light gray nodes, while the different data instances are the white nodes. Finally, the differences between two files are saved to record the changed data since the last incremental backup, where we typically save the different data instances and the hash table of matching instances in a specific file form (see Figure 21).

We also test our incremental backup strategy on two actual files (M2 and M4), as shown in Figure 16. Table 8 shows the comparison of storage space between the full backup and our incremental backup, where M2 and M4 are assumed to be the previous and current versions, respectively. The result suggests that our incremental backup saves around 73% space in contrast to the full backup. The incremental backup of IFC files is an attractive research topic, and its full implementation including an efficient recovery process will be left to our future work.

6. Conclusion, contribution and discussion

This paper presents a content-based automatic comparison approach for IFC files, and presents the development of a file comparison tool *IFCdiff*. The novelty is to build the hierarchical structures for comparing IFC files along with eliminating their redundant instances in the comparison process. Here the built hierarchical structures of two IFC files are compared with an iterative bottom-up procedure instead of comparing the original files. Such a process does not need to flatten all the data instances in IFC files. In the comparison process, all matching instances between two files are saved in the hash table, while the differences between them are also recorded for further applications. To evaluate the performance of our approach, the presented approach is tested on some IFC files exported through several commercial BIM software platforms. Finally, we make use of the presented approach to demonstrate a potential application to incremental backup of IFC files. The experimental results show that our approach outperforms the previous methods.

The significant contributions of our work are summarized as follows.

- We build the hierarchical structures for comparing IFC files with an iterative bottom-up procedure. Compared with the previous flattening-based approach, our approach avoids the procedure of flattening instances. As a result, our approach can greatly reduce the computational time and space in the file comparison process. The experimental result shows that the percentage of reduced time with our algorithm is generally very high (the average is 92.13%) for tested cases.
- In the level-by-level comparison of hierarchical structures, we also eliminate redundant instances appearing in two IFC files. This brings two advantages. On the one hand, by removing redundant instances can significantly decrease the number of data instances to be compared, which improves the comparison efficiency. On the other hand, by removing the redundant instances while keeping the complete IFC models, the comparison result using our approach is not sensitive to redundant instances in IFC files, which brings a stable and reliable similarity superior to the previous methods.
- We apply the presented comparison approach to incremental backup of IFC files. Here our approach is used for identifying and recording the changed data between the previous version and the current one.

The result suggests that our incremental backup can greatly save the storage space in contrast to the full backup.

Some previous studies have contributed to the issue of removing redundant instances in a single IFC file, such as Solibri IFC Optimizer [27] and IFCCompressor [9]. In this paper, we follow a similar manner to [9] to remove redundant instances in each level comparison, so the strategy of removing redundant instances used in this paper is not new. In practice, however, the IFC files generated by various software platforms often include a large number of redundant instances [9, 13], so the redundant instances should be considered in the process of IFC comparison. In this paper, we argue that it is meaningful and important to make use of the hierarchical structures for comparing IFC files along with removing redundant instances. This combination of hierarchical comparison and redundance elimination can significantly speed up the file comparison process and obtain a stable similarity. Even if two IFC files without redundancies are compared to each other, our approach can still reduce the computational time and space in contrast with the previous flattening-based approach. The *IFCdiff* presented in this paper can be considered as a complementary tool for the existing IFC tools.

Our comparison method only deals with the *syntax* content of data instances explicitly extracted from the input IFC file itself, but the *semantic* content of data instances implicitly derived from the IFC file is not handled yet. The domain of geometry comparison and objectified relationships belongs to the semantic comparison problem, which is quite complicated and will be our future work. We give an example for illustrating this problem as follows. In our method, one data instance explicitly extracted from the input IFC file itself consists of three terms (i.e. instance name, entity name and attribute values). If the entity name and attribute values between two data instances are consistent, they are considered to be the same. In contrast, if the geometric representation is different between two data instances, they will be potentially considered to be different in our method. This may not be always true. In particular, the IFC schema provides various geometric representations (such as swept, CSG and B-rep) for a solid model, which can be freely chose by an BIM modelling system. This means that the same solid model may have many different geometric representations. To address this issue, a possible way of geometry comparison is to first discretize the solid models into 3D meshes and then use some existing 3D shape comparison methods for the discretized shapes [28, 29, 30, 31, 32, 33].

Supplementary material

The online *IFCdiff* tool and its demonstration can be accessed at: `http://cgcad.thss.tsinghua.edu.cn/liuyushen/ifcdiff/`.

Acknowledgements

The research is supported by the National Natural Science Foundation of China (61472202, 61272229). The third author is supported by the National Key Technologies R&D Program of China (2015BAF23B03).

Appendix A. Comparison of computational complexities

In this section, we mainly compare computational complexities between our approach and the flattening-based algorithm [13]. If only considering the comparison process, both the time complexities of our method and the flattening-based method are $O(n\log(n))$ (see Section 4.4) with the assumption that the length of each data instance is a constant, where n is the average number of data instances between two IFC files to be compared. Our method is able to accomplish the comparison of reference instances by only comparing the instance name with the help of an auxiliary map storing matched instance names. However, the flattening-based method replaces all the reference numbers with their actual values in each IFC file, which makes an IFC file into a structure that does not include any referencing or inheritance structure. This will result in that the length of the flattened data instance increases dramatically (see an example in Section 2.4). Therefore, the length of the data instance cannot be seen as a constant any more. Assuming that the average length of the flattened data instance is L, the time complexity of the flattening-based algorithm is $O(Ln\log(n))$, which of course will take much more time than our method.

In addition, our algorithm has some extra advantages in contrast to the flattening-based method from the perspective of space complexity. In order to facilitate the description and analysis, the comparing file and the compared file can be simplified as a hierarchical structure to establish the reference mechanism based on a tree. Let d be the depth of the tree and L be the average length of each data instance. We also assume that the number of attributes of the instance is K and they are all references. Finally, from the bottom of the tree, each data instance is referenced by others for h times.

In our algorithm, the number of the terminal level is k^{d-1} and the total length is $k^{d-1}L$. Similarly, the second layer from the bottom has k^{d-2} data instances and the total length is $k^{d-2}L$. Therefore, we can deduce that the space complexity of our algorithm is

$$S_1 = k^{d-1}L + k^{d-2}L + ... + L. \tag{A.1}$$

Meanwhile, in the flattening-based method, the number of the terminal level is k^{d-1} and the total length is $k^{d-1}L$. Similarly, the second layer from the bottom has k^{d-2} data instances and the total length is $k^{d-1}Lh+k^{d-2}L$. Thus, we can deduce that the space complexity of the flattening-based algorithm is

$$\begin{aligned} S_2 &= k^{d-1}L + (k^{d-1}Lh + k^{d-2}L) + (k^{d-1}Lh + k^{d-2}L)hL + k^{d-3} + ... + L \\ &= k^{d-1}Lh + (k^{d-1}Lh + k^{d-2}L) + ... + k^{d-1}L + k^{d-2}L + ... + L \\ &= k^{d-1}Lh + (k^{d-1}Lh + k^{d-2}L) + ... + S_1. \end{aligned} \tag{A.2}$$

Apparently, the whole file increases dramatically large after the flattening process, which explains why the flattening-based method takes much longer than our method.

References

[1] C. Eastman, P. Teicholz, R. Sacks, K. Liston, BIM Handbook: A Guide to Building Information Modeling for Owners, Managers, Designers, Engineers and Contractors, 2nd Edition, John Wiley and Sons, NJ, 2011.

[2] BuildingSMART, Industry Foundation Classes (IFC). Available from: `http://www.buildingsmart-tech.org/specifications/ifc-overview/` (2014).

[3] S. Jeong, Y. Ban, Computational algorithms to evaluate design solutions using space syntax, Computer-Aided Design 43 (6) (2011) 664–676.

[4] J. Zhang, F. Yu, D. Li, Development and implementation of an Industry Foundation Classes-based graphic information model for virtual construction, Computer-Aided Civil and Infrastructure Engineering 29 (1) (2014) 60–74.

[5] R. Vanlande, C. Nicolle, C. Cruz, IFC and building lifecycle management, Automation in Construction 18 (1) (2008) 70–78.

[6] C. Eastman, J. Lee, Y. Jeong, J. Lee, Automatic rule-based checking of building designs, Automation in Construction 18 (8) (2009) 1011–1033.

[7] P. Pauwels, D. V. Deursen, R. Verstraeten, J. D. Roo, R. D. Meyer, R. V. de Walle, J. V. Campenhout, A semantic rule checking environment for building performance checking, Automation in Construction 20 (5) (2011) 506–518.

[8] Y.-H. Lin, Y.-S. Liu, G. Gao, X.-G. Han, C.-Y. Lai, M. Gu, The IFC-based path planning for 3D indoor spaces, Advanced Engineering Informatics 27 (2) (2013) 189–205.

[9] J. Sun, Y.-S. Liu, G. Gao, X.-G. Han, IFCCompressor: A content-based compression algorithm for optimizing Industry Foundation Classes files, Automation in Construction 50 (2015) 1–15.

[10] G. Gao, Y.-S. Liu, M. Wang, M. Gu, J.-H. Yong, A query expansion method for retrieving online BIM resources based on Industry Foundation Classes, Automation in Construction 56 (2015) 14–25.

[11] G. Gao, Y.-S. Liu, P. Lin, M. Wang, M. Gu, J.-H. Yong, BIMTag: Concept-based automatic semantic annotation of online BIM product resources, Advanced Engineering Informatics 31 (2017) 48–61.

[12] G. Arthaud, J. Lombardo, Automatic semantic comparison of STEP product models: Application to IFC product models, in: Innovations in Design & Decision Support Systems in Architecture and Urban Planning, Heeze, The Netherlands, 2006, pp. 447–463.

[13] G. Lee, J. Won, S. Ham, Y. Shin, Metrics for quantifying the similarities and differences between IFC files, Journal of Computing in Civil Engineering 25 (2) (2011) 172–181.

[14] R. Lipman, M. Palmer, S. Palacios, Assessment of conformance and interoperability testing methods used for construction industry product models, Automation in Construction 20 (4) (2011) 418–428.

[15] W. Gielingh, An assessment of the current state of product data technologies, Computer-Aided Design 40 (7) (2008) 750–759.

[16] T. Pazlar, Ž. Turk, Evaluation of IFC optimization, in: Proceedings of CIB W78 Conference on Bringing ITC Knowledge to Work, 2007, pp. 61–66.

[17] Y.-S. Jeong, C. Eastman, R. Sacks, I. Kaner, Benchmark tests for BIM data exchanges of precast concrete, Automation in Construction 18 (4) (2009) 469–484.

[18] T. Pazlar, Ž. Turk, Interoperability in practice: Geometric data exchange using the IFC standard, ITcon 13 (2008) 362–380.

[19] H. Ma, K. M. E. Ha, C. K. J. Chung, R. Amor, Testing semantic interoperability, in: Proceedings of Joint International Conference on Computing and Decision Making in Civil and Building Engineering, Montreal, Canada, 2006, pp. 1216–1225.

[20] ISO 10303-21:2002, Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure (2002).

[21] File comparison tools. Available from: `http://en.wikipedia.org/wiki/Comparison_of_file_comparison_tools` (2015).

[22] J. Oraskari, S. Törmä, RDF-based signature algorithms for computing differences of IFC models, Automation in Construction 57 (2015) 213–221.

[23] J. Katajainen, E. Mäkinen, Tree compression and optimization with applications, International Journal of Foundations of Computer Science 1 (4) (1990) 425–447.

[24] G. Busatto, M. Lohrey, S. Maneth, Efficient memory representation of XML document trees, Information Systems 33 (4–5) (2008) 456–474.

[25] T. Liebich, IFC 2x Edition 3 Model Implementation Guide (Version 2.0). (2009).

[26] L. Zhang, R. Issa, Ontology based partial building information model extraction, Journal of Computing in Civil Engineering 27 (6) (2013) 576–584.

[27] Solibri, Solibri IFC Optimizer. Available from: `http://www.solibri.com/solibri-ifc-optimizer.html` (2014).

[28] Y.-S. Liu, K. Ramani, M. Liu, Computing the inner distances of volumetric models for articulated shape description with a visibility graph, IEEE Transactions on Pattern Analysis and Machine Intelligence 23 (12) (2011) 2538–2544.

[29] Y.-S. Liu, Y. Fang, K. Ramani, IDSS: deformation invariant signatures for molecular shape comparison, BMC Bioinformatics 10 (157) (2009) 1–14.

[30] Y. Fang, Y.-S. Liu, K. Ramani, Three dimensional shape comparison of flexible protein using the local-diameter descriptor, BMC Structural Biology 9 (29) (2009) 1–15.

[31] Y.-S. Liu, Q. Li, G.-Q. Zheng, K. Ramani, W. Benjamin, Using diffusion distances for flexible molecular shape comparison, BMC Bioinformatics 11 (480) (2010) 1–15.

[32] J. Feng, Y.-S. Liu, L. Gong, Junction-aware shape descriptor for 3D articulated models using local shape-radius variation, Signal Processing 112 (2015) 4–16.

[33] Y.-S. Liu, H. Deng, M. Liu, L. Gong, VIV: Using visible internal volume to compute junction-aware shape descriptor of 3D articulated models, Neurocomputing 215 (2016) 32–47.