

IFCCompressor: A content-based compression algorithm for optimizing Industry Foundation Classes files



Jing Sun^a, Yu-Shen Liu^{a,b,c,*}, Ge Gao^{a,d}, Xiao-Guang Han^{a,d}

^a School of Software, Tsinghua University, Beijing, China

^b Key Laboratory for Information System Security, Ministry of Education of China, China

^c Tsinghua National Laboratory for Information Science and Technology, China

^d Department of Computer Science and Technology, Tsinghua University, China

ARTICLE INFO

Article history:

Received 4 April 2014

Received in revised form 25 October 2014

Accepted 30 October 2014

Available online xxxx

Keywords:

Building Information Modeling (BIM)

Industry Foundation Classes (IFC)

Content-based compression

IFC optimization

ABSTRACT

As the commonly used open and neutral file format for Building Information Modeling (BIM) data, IFC (Industry Foundation Classes) aims to facilitate interoperability between various software platforms in the AEC industry. However, the IFC files generated from different systems often contain enormous redundant information, which will greatly limit IFC-based data storage and exchange, management, transmission and other applications. To address this issue, this paper presents a content-based compression algorithm, named IFCCompressor, for optimizing IFC data files. Its goal is to make a large IFC file as small as possible by reducing its redundant information. The algorithm is achieved through an iterative compression procedure based on an IFC model's tree structure. The optimization procedure can be lossless or be constrained by an error bound. Compared with pure compression (ZIP) regardless of information content, the presented algorithm starts with a comprehensive analysis of structure and content of IFC file, and then eliminates its redundant information without changing the original file format. Unlike partial model extraction methods, our algorithm results in a complete IFC model but with a more compact IFC physical file. In contrast with some commercial IFC optimization tools such as Solibri IFC Optimizer, the algorithm can make the size of IFC files smaller. The experimental results show that the algorithm is particularly effective for some office/residential building models with a large number of duplicated components. The compression rate with our algorithm is generally very high (the average is 40.32%) for tested cases. The online IFCCompressor tool and its demonstration can be accessed at: <http://cgcad.thss.tsinghua.edu.cn/liuyushen/IFCCompressor/>.

© 2014 Elsevier B.V. All rights reserved.

1. Introduction

BIM (Building Information Modeling) technology has been receiving an increasing attention in the AEC (Architecture, Engineering and Construction) industry [1]. Compared with the traditional CAD technology, BIM is capable of restoring both geometric and rich semantic information of building models, as well as their relationships, to support lifecycle data sharing. As the commonly used open and neutral file format for BIM data, IFC (Industry Foundation Classes) [2] led by the buildingSMART, formerly known as International Alliance for Interoperability (IAI), plays a crucial role to facilitate interoperability between various software platforms in the AEC industry. Today, the IFC data format has been widely supported by the market-leading BIM software vendors. A list of software applications/utilities, providing IFC import and/or export

functionality, is available at the buildingSMART website [3]. A number of recent research articles have concerned extracting and managing IFC information for various applications, such as automatic rule-based checking [4], evaluation of design solutions [5], construction cost estimating [6], construction management [7], partial model extraction [8,9], IFC-based path planning [10] and others.

However, the IFC files generated by various software platforms often contain enormous redundant information. For example, one data instance in an IFC model may be mapped to multiple instances in the exported IFC file, which can result in a hardly manageable IFC file due to its text based record. Some previous studies [8–17] have exposed the large IFC file size as a major burden on IFC-based data storage and exchange, management, transmission and other applications. For instance, VTT research project SPADEX [12,16] listed this issue as one of seven major implementation obstacles in IFC applications.

The limitation of large file sizes can be overcome in several ways. A direct way is to use plain text compression (e.g. ZIP) algorithms or tools for reducing the file size [18], regardless of information content of IFC files. However, the ZIP file format does not change the typical loading (IFC import) time and memory consumption because it has to

* Corresponding author at: School of Software, Tsinghua University, Beijing 100084, China. Tel.: +86 10 6279 5455, +86 159 1083 1178 (mobile); fax: +86 10 6279 5460.

E-mail addresses: sunjing11@mails.tsinghua.edu.cn (J. Sun),

liuyushen@tsinghua.edu.cn (Y.-S. Liu).

URL: <http://cgcad.thss.tsinghua.edu.cn/liuyushen/> (Y.-S. Liu).

be decompressed to the original IFC file in memory first. Another way is to extract a partial model or required information from the original complete IFC model with respect to specific applications [8,9,15]. Nevertheless, partial model extraction is limited by specific situations or purposes, which also requires users to understand the entire complex inheritance and aggregation structures of IFC. Regardless of specific application requirements, the third way is the content-based compression/optimization methods, which focus on analyzing the structure of an IFC file itself and eliminating its redundant information. A well-known representative tool is “Solibri IFC Optimizer” [19]. However, as a commercial software, its algorithm and implementation are not available. Our method falls into the third way.

This paper presents a content-based compression algorithm, named *IFCCompressor*, for optimizing large IFC files. Its goal is to make a large IFC file as small as possible by reducing its redundant information. Unlike partial model extraction, our algorithm results in a complete IFC model but with a more compact IFC physical file. Compared with pure ZIP compression regardless of information content, the presented algorithm reduces the redundant information content of an IFC file itself without changing the original IFC file format. In contrast with some commercial IFC optimization tools such as Solibri IFC Optimizer, our algorithm can make the size of IFC files smaller. Our method is tested on a number of IFC files which are exported through several commercial BIM software platforms, including *Autodesk Revit Architecture* and *Graphisoft ArchiCAD*. The experimental results show that the algorithm is particularly effective for some office/residential building models with a large number of duplicated components.

1.1. Related work

Some recent studies have indicated that the IFC-based data exchange required further improvement in the technology before IFC could be fully adopted in the project data exchange [17,20]. A prominent problem is how to deploy IFC as an exchange format for large projects. Several literature [12,13,16,17,20] indicate specific technical issues in IFC-based data exchange, such as geometric misrepresentation, loss of object information, confusion in interdisciplinary revision, application-specific IFC input/output, and large IFC file sizes. This paper focuses on the last issue, i.e. large IFC file sizes. Large file sizes make it difficult to handle and exchange an IFC model as a file, as reported in Refs. [8–15]. As mentioned above, the limitation of large IFC file sizes can be overcome in three ways: pure ZIP compression, partial model extraction, and content-based compression/optimization.

1.1.1. Pure ZIP compression

The most direct way is to use plain text compression (ZIP) for reducing the size of IFC data files, regardless of information content of IFC models. For instance, IFC-ZIP is a typical ZIP compressed format, which consists of an embedded IFC data file with file extension “.ifcZIP” [18] using the PKzip compression algorithm. The buildingSMART [18] announces that IFC-ZIP files usually compress the IFC data files down by 60–80%. Pazlar and Turk [12] compared the compression rate between IFC-ZIP and other IFC optimization tools [19] in some case studies.

The ZIP compressed format (e.g. IFC-ZIP) is usually effective only for IFC data transmission, such as for sending a large IFC model as e-mail attachment. However, the ZIP compressed format does not change the typical loading (IFC import) time and memory consumption in essence because it has to be decompressed to the original IFC file in memory first. In addition, parsing the IFC-ZIP format is software-dependent, which means that only the IFC-ZIP supported software can parse this kind of file format. This also leads many software vendors to do extra development for importing the IFC-ZIP format. Since the IFC file format has been strictly defined in terms of its schema information, the optimization efforts should focus on the structure and content of the IFC file.

1.1.2. Partial model extraction

The second way is to extract a partial model or required information from the original complete IFC model with respect to specific applications [8,9,15]. As a building information model is transferred between various participants, more and more information will be added to the model by different stakeholders at different stages during the building lifecycle [8,9], which often results in very large IFC files. With the vast volumes of information being stored in BIM models, the problem of “information overload” has been increasingly recognized in the AEC field [8,9,21]. This issue can be solved by extracting an information subset or a partial model from a complete building information model.

Recently, several approaches have been proposed to extract partial IFC models for specific application requirements [8,9,22]. For instance, Beetz et al. [22] developed a tool for converting IFC information into the OWL, which was applied to partial model extraction of IFC model by combing a graph query technique. Zhang and Issa [8] presented an approach to extract a partial IFC model from the complete IFC file, where an ontology-based framework is introduced for querying specific information from the IFC model [23]. More recently, Won et al. [9,15] described a no-schema algorithm for extracting a partial model from an IFC instance model.

In some cases, the above methods could reduce the size and complexity of an IFC model for specific applications. However, partial model extraction is highly dependent on specific application requirements or purposes, which may not be always available. In addition, partial model extraction still remains a research challenge. Defining a reasonable partial model representation often requires users to understand the entire complex inheritance and aggregation structures in IFC specification [9], which is a time-consuming and difficult task for the novice. Another main issue is that the extracted partial IFC model is only an incomplete model, which limits many applications that require access to the complete information stored in the original model during the lifecycle of the project.

1.1.3. Content-based compression/optimization

In order to remove the redundant information in an IFC file while keeping its complete building model, the third way is the content-based compression/optimization methods. Such methods first analyze the structure and content of an individual IFC data file and then eliminate its redundant information, regardless of specific application requirements. It can greatly reduce the file size and processing time required for working with the IFC file. In the presented research work, the term *optimization* or *compression* represents the process of reducing the size of IFC data file without any information loss or with an error bound.

Some possible reasons for the abundance of redundancy in the generated IFC data files are as follows.

- (1) *Differences of model mapping mechanism* between various BIM software platforms and the standard IFC data. It is worth mentioning that each BIM software platform has its own internal representation and structure for a building information model. In order to achieve applications and data exchanges in accordance with the IFC specification, mapping a platform internal schema into the IFC schema has to be fulfilled by the platform itself, i.e. mapping between the internal model and the IFC model for exports and mapping between the IFC model and the internal model for imports [12,17,20,24,25]. Neither mapping is trivial, but perfect structure mapping cannot be expected in practical applications due to the specific internal representation [12]. Ref. [12] has shown some case studies, in which different model mappings and description approaches may greatly increase the size of IFC files.
- (2) *Various possibilities offered by IFC specification*. Some applications describe the same contents, but they can be constructed differently than those based on IFC specification. Therefore, IFC translator implementers can freely choose different approaches that best serve their needs. Lee et al. [20] indicated that such various possibilities

can make two IFC files exported from the same model have large differences in physical file size, number of instances, attribute values and inconsistency in object type.

- (3) If an IFC file is repeatedly imported and exported among different systems, its file size may rapidly increase even if no modification is made to the original model [20].

A few of recent studies attempted to consider the problem of content-based IFC optimization. A representative software tool is the Solibri IFC Optimizer [19]. According to the official announcement on Solibri Inc., the optimizer tool is lossless and optimizes the IFC file into a much smaller size than the original. It mainly removes the redundant entities from the IFC file by updating the references. Pazlar and Turk [12] examined some IFC models for evaluating the effectiveness of several IFC optimization methods including Solibri IFC Optimizer. They [12] confirmed some advantages such as decreasing the IFC file size and improving the loading performance. However, they also found that Solibri IFC Optimizer is not as good as its announcement, and in some cases this software is still doubtful [12]. In addition, as a commercial software, the algorithm and implementation of Solibri IFC Optimizer are unavailable, and users have no way to find out what exactly happened to the optimization procedure.

Large IFC file sizes have been a big issue in practice, where numerous duplicated instances and references in the IFC file are a serious problem [20]. Therefore, an ideal IFC file-size optimizer, which can eliminate redundant instances, should be developed to relieve this redundancy problem [12,19,20]. However, in the long run a redundancy check and compression should be included in a normal IFC development procedure.

1.1.4. XML compression

Another possible way is to convert the original IFC file into the ifcXML format [26,27]. From this many XML (Extensible Markup Language) compression techniques [28] can be used for ifcXML. The XML compression approaches can be roughly divided into two categories [28]. The first category is based on general text compression, which treats XML documents as normal plain text documents and thus applies the traditional text compression techniques (e.g. ZIP). The second category is based on XML-conscious compression, which takes advantage of the XML document structure to achieve better compression rate over general text compression. According to the dependence on the XML schema information, the XML-conscious compression approaches can be further classified: schema-dependent compression and schema-independent compression. Here schema-dependent compression requires the availability of the schema information of the XML documents during their encoding and decoding processes.

The ifcXML format is often suitable for interoperability with XML tools and some applications of exchanging partial building models. However, due to the large sizes of typical building models, this format is less commonly practiced. Our algorithm can be regarded as a schema-dependent compression algorithm (only for IFC files), which specifically utilizes the IFC schema information to go through the compression process. It is of interest to extend our algorithm to ifcXML compression with the help of the IFC schema, and we leave this for future work.

1.2. Contributions

Our main contributions can be summarized as follows.

- A content-based compression algorithm, named IFCCompressor, is presented for optimizing an individual IFC file. In essence, our algorithm is a schema-dependent compression algorithm, which utilizes the IFC schema information to complete the compression process. Our algorithm can decrease the IFC file size considerably and reliably. The compression rate with our algorithm is generally very high (the average is 40.32% for tested cases).
- Compared with pure plain text compression techniques (e.g. ZIP) that are regardless of information content of IFC models, our algorithm

starts with a comprehensive analysis of the structure and content of an IFC data file and then eliminates its redundant information without changing the original IFC file format.

- Unlike partial model extraction methods that produce an incomplete sub-model with respect to specific application requirements, our algorithm results in a complete IFC model but with a more compact IFC physical file.
- In contrast with Solibri IFC Optimizer, our algorithm can make the size of IFC files smaller. Our optimization procedure can be lossless or be constrained by an error bound.

2. Structure of the IFC data file

IFC specifies a conceptual data schema and an exchange file format for BIM data, which is an official International Standard ISO 16739:2013. In this paper, we typically choose the IFC2 × 3 specification that contains 653 entities and over 300 supplementary data types as well as extensible property sets. The most common way of exchanging data using IFC today is in a file format with the extension “*.ifc”, called IFC-SPF. Here SPF stands for STEP Physical File defined by ISO 10303-21 [29] using the clear text encoding of the exchange structure, in which each line typically consists of a single object record. It has the advantage of resulting in compact size yet readable text. For convenience, the IFC-SPF file is simply referred to as *IFC data file* (or *IFC file*) in this paper. Each IFC data file, using the STEP Physical File format (or ISO 10303-21 structure), is split into two sections: *header section* and *data section* [30], following the initial keyword *ISO-10303-21*.

2.1. Header section

The header section has information on the file description, the date and time when the file was done, the name of company and authorizing person of the file, the IFC version used, etc. (see Fig. 1). The head section chooses the keyword “HEADER” as the beginning and “ENDSEC” as the end. The header section has a fixed structure consisting of 3 to 6 groups in the given order, which is very short relative to the data section, so the header section would not be compressed in our work.

2.2. Data section

The data section contains the main data of building information model described in the IFC data file, which includes the geometric and semantic information of building objects as well as their relationships. The data section chooses the keyword “DATA” as the beginning and “ENDSEC” as the end (see Fig. 1). The data section is formed by numerous *entity instances* (or named *data instances*), where each entity instance takes “#” as the beginning of the sentence, followed by three parts: *instance name*, *entity name* and a list of *attribute values*. Many entity instances of an IFC entity with the same entity name can exist in one model file. An instance name (e.g. “#80693”) shall be encoded as a number sign, “#”, followed by a sequence of DIGIT, LOWER and UPPER characters. The instance name can also be used as a *reference id* cited by other entity instances. In the IFC file, the instance name is generally encoded by digits. Note that even without this assumption our algorithm is still working due to the uniqueness of the instance name. Several remarks for instance name should be mentioned below [30].

- The instance name is unique within the scope of an IFC file.
- The instance name is only valid for a single exchange. If the same user's project is exported for the second time, the instance name may be inconsistent.
- The instance name's order within the data section is not important.

Fig. 1 gives an example for the structure of IFC data file and some basic terms used throughout this paper. We refer the readers to the comprehensive introduction of IFC data file structure from Ref. [30].

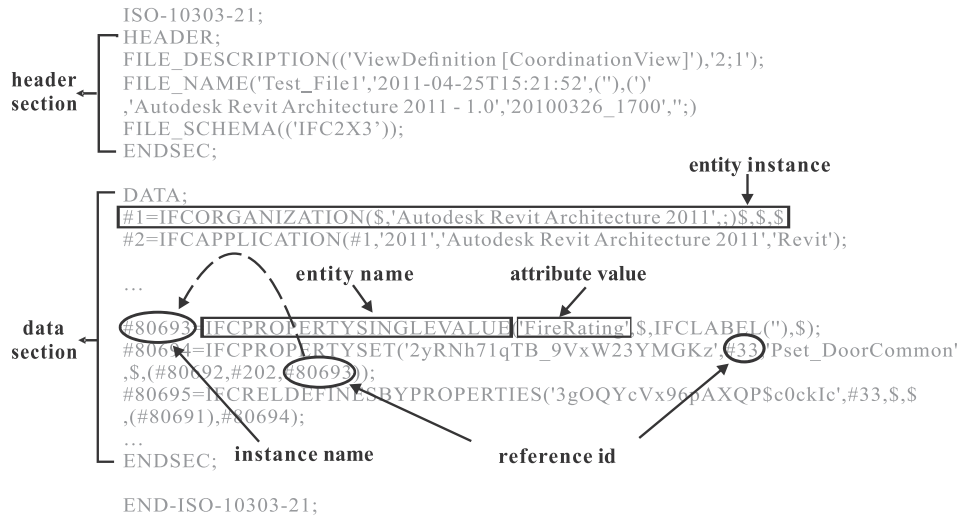


Fig. 1. An example for the structure of IFC data file and some basic terms used throughout this paper. The IFC file is split into two sections: header section and data section, and the basic terms are highlighted including instance name, entity name, attribute value and reference id.

3. The content-based IFC compression algorithm

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. Data compression is a way to reduce or eliminate redundancy. Complying with information theory, our method solves the problem of redundancy existing in IFC files in AEC field.

An IFC file exported through different IFC-supported software often includes a number of *identical data instances* [20] (i.e. multiple instances of the same entity with the same entity name and attribute values, but with different instance names). The identical data instances are the

typical redundancy that should be removed from the IFC file in an ideal situation [12,20]. For example, duplicated instances of the *IfcDirection* entity with the same value are one common example of such cases. An ideal IFC file-size compressor/optimizer should be able to efficiently identify those duplicated instances and remove them from the IFC file.

To achieve the content-based IFC compression algorithm, the main problem lies on how to identify all identical data instances and remove the duplicated ones. However, it is difficult to check all redundant instances for a large IFC file in one step, since there are often tens or even hundreds of thousands of data instances with the complex referencing and inheritance structure within the large IFC file. This

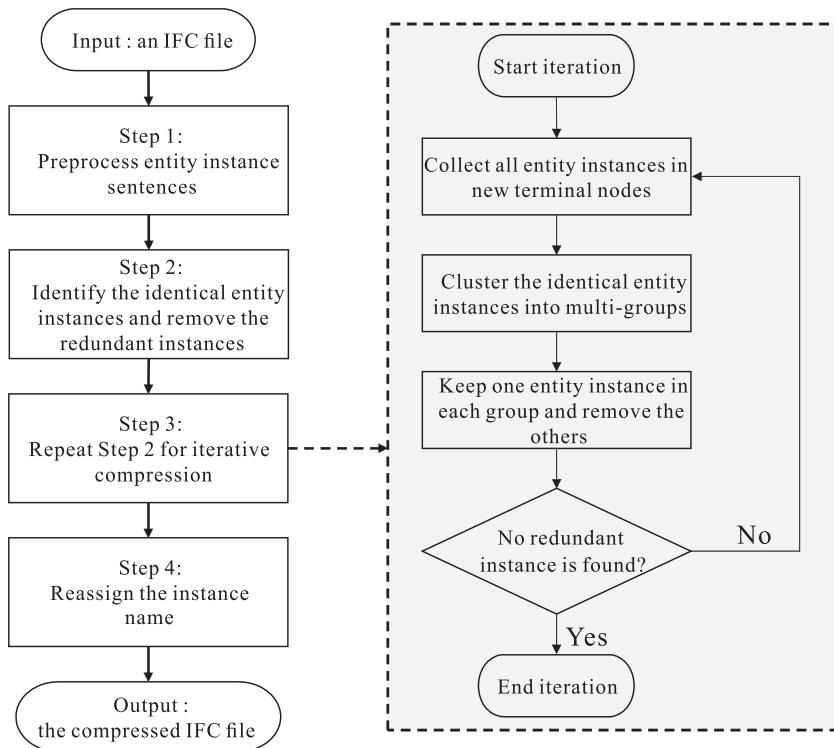


Fig. 2. The flow diagram of the content-based IFC compression algorithm.

paper presents an iterative compression algorithm for locating all identical data instances and removing the duplications. This is achieved through an iterative procedure with the aid of IFC model's tree structure. Fig. 2 shows the main procedure of our algorithm. Starting with an IFC data file as input, our algorithm mainly consists of the following four steps.

- Step 1 Preprocess the data instance sentences within the input IFC file. This step removes redundant information (e.g. blank spaces and multi-lines) in the sentence of each data instance and extracts the basic terms for further utilization. It aims to obtain a compact expression for each data instance to be saved in memory (see Section 3.1).
- Step 2 Identify the identical data instances and remove the redundant instances. This step first collects all the data instances on the terminal nodes of IFC model's tree (see Section 3.2 for details). Among all the terminal nodes, the identical data instances are identified and clustered into multiple groups. Only one data instance for each group is kept, while the others in the group are regarded as redundancy and removed.
- Step 3 Repeat Step 2 for further compressing the remaining data instances in the IFC file. Step 3 is a recursive and iterative process and it will be terminated until all the data instances are completely processed or all redundancies removed (see Section 3.3).
- Step 4 Finally, reassign the instance name (the number) for each data instance in the IFC file. This step first ranks all data instances based on their cited times, and then assigns a small instance name to the data instances with large cited times (see Section 3.4).

3.1. Step 1: preprocess data instance sentences within the input IFC file

The first step of our algorithm is to preprocess data instances in the input IFC file. Its goal is to remove redundant information in each data instance sentence and extract the basic terms for further utilization. The main procedure is as follows.

- (1) First, the sentence of data instance with multi-lines is converted into a single line.
- (2) Then, the content of three basic terms (i.e. instance name, entity name and attribute values) of each data instance sentence are extracted.

- (3) Next, redundant information, such as blank spaces, within the extracted attribute values is removed.

After the above preprocessing, we can obtain a compact expression for each data instance sentence to be saved in memory. Fig. 3 depicts the above procedure with an example.

3.2. Step 2: identify identical data instances and remove redundant instances

The second step of our algorithm is to identify the identical data instances and remove the duplications. As mentioned above, the identical data instances are multiple instances of the same entity with the same entity name and attribute values but with different instance names [20]. The main procedure is given below.

Firstly, we collect all data instances on the terminal nodes of IFC model's tree. An IFC model's tree [8] is a data structure, with the *IfcProject* entity as the root node, or level 1. All the entities with a containment relationship specified in the IFC schema with level *n* entities are structured as level *n + 1* nodes, or the children of level *n* nodes in the tree. The basic IFC entities, which cannot hold or contain other entities, are always placed as the leaves of the tree, called the *terminal nodes* in this paper. A terminal node (e.g. the *IfcCartesianPoint* entity) can be simply located as the data instance that does not include any reference id in its attribute values. Fig. 4 shows a sample of IFC file fragment, in which the data instances (e.g. "#3", "#4", "#5" and "#6") are recognized as the terminal nodes. The partial tree structure corresponding to the file fragment is shown in Fig. 5(a).

Secondly, we compare the collected terminal nodes and locate the identical data instances to be clustered into multiple groups. After preprocessing data instance sentences in Step 1, we can simply compare the values of two data instances (i.e. their entity names and the attribute values) in terms of string comparison. Here the terminal nodes with the same value are clustered into one group. Consequently, we can obtain multiple groups. In Fig. 5(a), the light gray nodes (i.e. data instances #3, #4, #5 and #6) denote one group of identical data instances.

Thirdly, only one data instance of each group is kept with all other redundant instances removed. Meanwhile, the reference id of attribute values in the remaining data instances is updated accordingly. As is shown in Fig. 5(b), the data instance #3 in the group is kept, while the other data instances (#4, #5 and #6) are removed. In addition, the

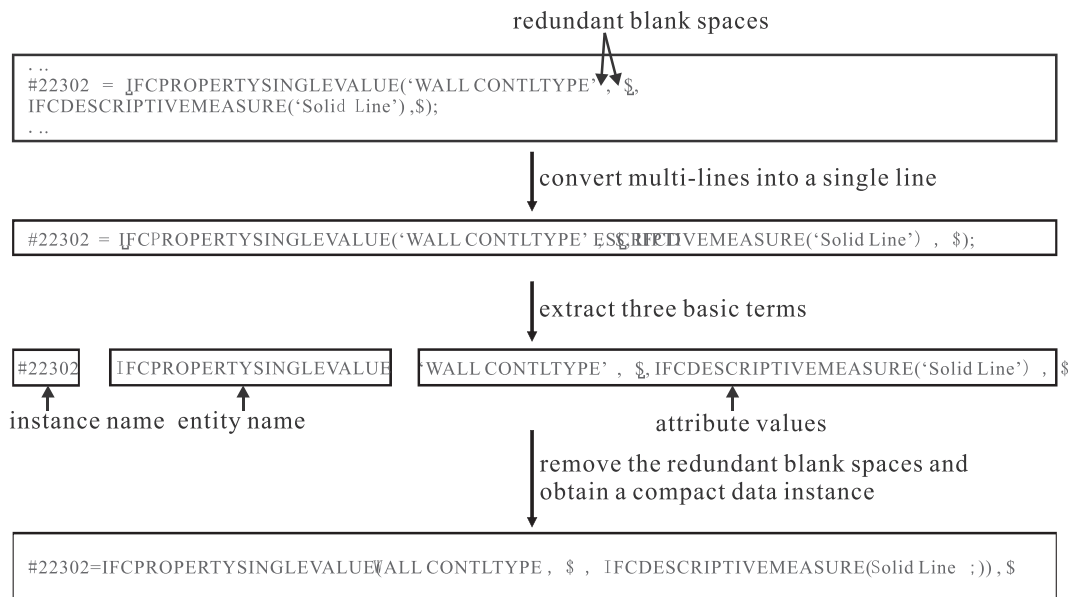


Fig. 3. Illustration of preprocessing data instance sentences within the IFC file.

```

...
#3=IFCCARTESIANPOINT((0.,0.,0.));
#4=IFCCARTESIANPOINT((0.,0.,0.));
#5=IFCCARTESIANPOINT((0.,0.,0.));
#6=IFCCARTESIANPOINT((0.,0.,0.));
...
#1422715=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1422716=IFCMAPPEDITITEM(#290462,#1422715);
#1422717=IFCSHAPEREPRESENTATION(#27,'Body','MappedRepresentation',(#1422716));
#1422718=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1422719=IFCMAPPEDITITEM(#290463,#1422718);
#1422720=IFCSHAPEREPRESENTATION(#29,'Plan','MappedRepresentation',(#1422719));
#1422721=IFCPRODUCTDEFINITIONSHAPE($,$,(#1422717,#1422720));
...
#1422727=IFCDOOR('1$OQR48_T1GgA0Zocauppv',#33,'door2',$,'TLM3023',#1422726,#1422721
,'1240474',2299.99999999998,3000.000000000001);
...
#1772907=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1772908=IFCMAPPEDITITEM(#290462,#1772907);
#1772909=IFCSHAPEREPRESENTATION(#27,'Body','MappedRepresentation',(#1772908));
#1772910=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1772911=IFCMAPPEDITITEM(#290463,#1772910);
#1772912=IFCSHAPEREPRESENTATION(#29,'Plan','MappedRepresentation',(#1772911));
#1772913=IFCPRODUCTDEFINITIONSHAPE($,$,(#1772909,#1772912));
...
#1772919=IFCDOOR('1$OQR48_T1GgA0ZocaujLj',#33,'door1',$,'TLM3023',#1772918,#1772913
,'1246222',2299.99999999998,3000.000000000001);
...

```

Fig. 4. A sample of IFC file fragment for depicting the iterative compression algorithm.

upper parent nodes (i.e. data instances #1422718, #1772910 and #1772907) are respectively relocated to data instance #3. After the above process, we achieve an initial compression for these data instances on the terminal nodes in the IFC model's tree.

3.2.1. Handle float-point rounding (FPR) error

The problem of float-point rounding (FPR) error might be caused during IFC data storage and exchange between various software platforms. For instance, one of the attribute values in the entity *IfcCartesianPoint* is the float-point number, but it might go wrong (e.g. from 0.560000 to 0.559999) after file exchange between two different platforms. To solve this problem, our algorithm also offers an error bound for identifying identical data instances. See Section 4.3 for the detailed introduction and experimental results with FPR error bound.

3.3. Step 3: repeat the iterative compression process

Due to the complex inheritance and referencing relationships in large IFC files, it is not enough to obtain a compact IFC file only by removing the redundant instances on the terminal nodes in the IFC model's tree. Therefore, we make use of an iterative strategy for marking out all identical data instances hidden in the tree and removing the duplications. The main procedure is given below.

We first collect all the data instances which cite the terminal nodes mentioned in Section 3.2, while the collected data instances will be treated as the new terminal nodes instead of the previous ones. As is shown in Fig. 5(b), the data instances (i.e. #1422715, #1422718, #1772910 and #1772907) are becoming the new terminal nodes instead of the previous terminal node #3.

Similar to Step 2, we locate the identical data instances on the new terminal nodes, cluster them into multiple groups, and remove the redundant data instances in each group. As is shown in Fig. 5(b), the light gray nodes (i.e. data instances #1422715, #1422718, #1772910 and #1772907) denote one group of identical data instances. In Fig. 5(c), data instance #1422715 in the group is kept, while the others in the group are removed. Meanwhile, the upper parent nodes (i.e. #1422719 #1772911 #1772908) are respectively relinked to the same data instance #1422715.

We repeat the above procedure for further compressing the remaining data instances. If all the terminal nodes reach the tree's root node (e.g. *IfcProject*) or no more identical data instance is found, the procedure is terminated. Fig. 5(c), (d) and (e) shows the above compression procedure in several iterations. The final tree is given in Fig. 5(f), where no more identical data instance remains.

Fig. 6 shows the compressed IFC file fragment with our algorithm. For the specific IFC file fragment, 11 redundant data instances are identified and removed among the original 20 ones, resulting in about 50% compression rate.

3.4. Step 4: reassign the instance name

The last step of our algorithm is to reassign the instance name for obtaining a smaller IFC file. Every data instance in the IFC file is given a unique instance name in the form, e.g. "#1422719". The instance name must consist of a positive integer that is typically less than 2^{63} [29]. However, unnecessary large integers assigned to instance name will also increase the IFC file size due to numerous references to these large integers. This case may happen to many IFC files when exchanged between different commercial BIM platforms several times. As is mentioned before, the instance name is only meaningful to the IFC file and its specific order does not matter [30], so we can reorder the instance name and reassign a small unique integer to it. This can be done in the process as follows.

- (1) Count the *cited times* of each data instance within the IFC file. For example, the cited times of data instance #3 in Fig. 6 is the number of all other data instances with reference to #3.
- (2) Rank all data instances based on their cited times.
- (3) The instance name with the large cited times will be reassigned a small unique integer.

We apply the above process to reassign the instance name within the complete IFC file whose sample file fragment has been shown in Fig. 6. Before reassigning the instance name, the complete IFC file size is about 81.2 MB. In contrast, after reassigning all instance name, the file size is reduced to 78.5 MB. The corresponding file fragment after updating is shown in Fig. 7. Here the large instance name (e.g. #1772919 in Fig. 6) is

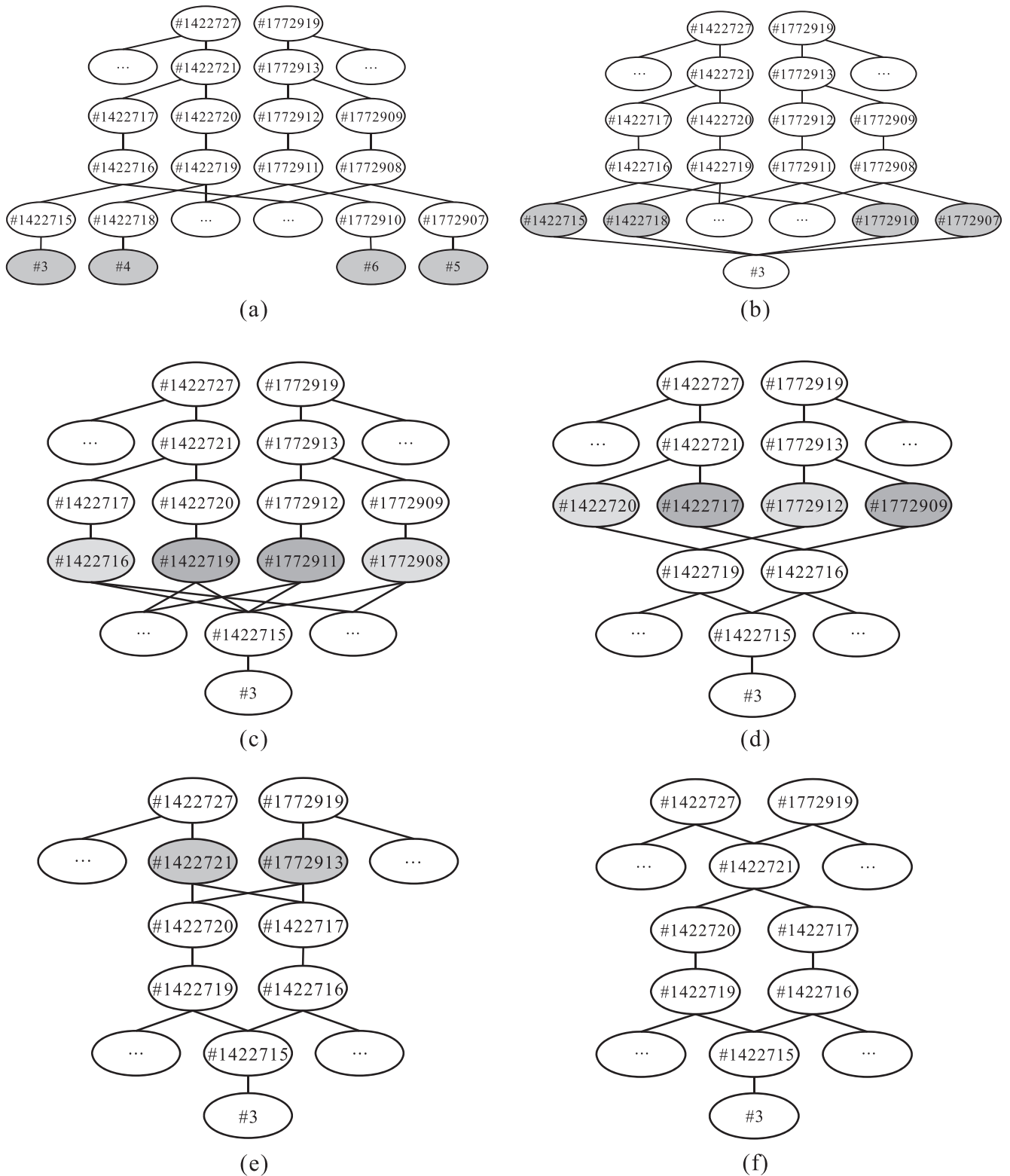


Fig. 5. The illustration of the algorithm procedure for compressing the sample of IFC file fragment in Fig. 4, where the partial tree corresponds to the IFC file fragment. (a) Grouping the identical data instances (light gray nodes) on the terminal nodes of the tree. (b) Removing the redundant instances and updating the corresponding reference id, and grouping the upper parent nodes of the tree again. (c), (d) and (e) Repeating the above compression procedure in several iterations. (f) The final result.

reassigned to the small one (e.g. #8596 in Fig. 7) and the corresponding reference id in attribute values is also updated. Note that the last two data instances #998625 and #998626, being cited just once, are assigned to a relatively large number (see Fig. 7).

3.4.1. Computational complexity

Let n be the number of all data instances in the original IFC file. First, it takes $O(n)$ to preprocess all data instances in Step 1. Then, it mainly includes node sorting $O(n \log(n))$ and node searching $O(\log(n))$ in the

```

...
#3=IFCCARTESIANPOINT((0.,0.,0.));
#4=IFCCARTESIANPOINT((0.,0.,0.));
#5=IFCCARTESIANPOINT((0.,0.,0.));
#6=IFCCARTESIANPOINT((0.,0.,0.));
...
#1422715=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1422716=IFCMAPPEDITEM(#290462,#1422715);
#1422717=IFCSHAPEREPRESENTATION(#27,'Body','MappedRepresentation',(#1422716));
#1422718=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1422719=IFCMAPPEDITEM(#290463,#1422715);
#1422720=IFCSHAPEREPRESENTATION(#29,'Plan','MappedRepresentation',(#1422719));
#1422721=IFCPRODUCTDEFINITIONSHAPE($,$,(#1422717,#1422720));
...
#1422727=IFCDOOR('1$OQR48_T1GgA0Zocaupp',#33,'door2',$,'TLM3023',#1422726,#1422721
,'1240474',2299.9999999998,3000.00000000001);
...
#1772907=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1772908=IFCMAPPEDITEM(#290462,#1772907);
#1772909=IFCSHAPEREPRESENTATION(#27,'Body','MappedRepresentation',(#1772908));
#1772910=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1772911=IFCMAPPEDITEM(#290463,#1772910);
#1772912=IFCSHAPEREPRESENTATION(#29,'Plan','MappedRepresentation',(#1772911));
#1772913=IFCPRODUCTDEFINITIONSHAPE($,$,(#1772909,#1772912));
...
#1772919=IFCDOOR('1$OQR48_T1GgA0ZocaujLj',#33,'door1',$,'TLM3023',#1772918,#1422721
,'1246222',2299.9999999998,3000.00000000001);
...

```

(a)

```

...
#3=IFCCARTESIANPOINT((0.,0.,0.));
...
#1422715=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#3,1.,$);
#1422716=IFCMAPPEDITEM(#290462,#1422715);
#1422717=IFCSHAPEREPRESENTATION(#27,'Body','MappedRepresentation',(#1422716));
#1422719=IFCMAPPEDITEM(#290463,#1422715);
#1422720=IFCSHAPEREPRESENTATION(#29,'Plan','MappedRepresentation',(#1422719));
#1422721=IFCPRODUCTDEFINITIONSHAPE($,$,(#1422717,#1422720));
...
#1422727=IFCDOOR('1$OQR48_T1GgA0Zocaupp',#33,'door2',$,'TLM3023',#1422726,#1422721
,'1240474',2299.9999999998,3000.00000000001);
...
#1772919=IFCDOOR('1$OQR48_T1GgA0ZocaujLj',#33,'door1',$,'TLM3023',#1772918,#1422721
,'1246222',2299.9999999998,3000.00000000001);
...

```

(b)

Fig. 6. Illustrating the compressed IFC file fragment corresponding to the one in Fig. 4 using our algorithm. (a) The redundant data instances in Fig. 4 are highlighted by strikethrough. (b) The correspondingly compressed IFC file is displayed after our iterative compression process.

IFC model's tree structure from Step 2 to Step 4. If using *hash map* data structure to save the data instances, the data instances can be directly accessed through their instance names without node searching. As a

result, an upper bound of running time, processing the whole iterative compression process, is $O(n^2 \log(n))$ (normal sort and search) or $O(n^2)$ (using hash map) for n nodes. A more detailed analysis for computational

```

...
#29=IFCCARTESIANPOINT((0.,0.,0.));
...
#44=IFCCARTESIANTRANSFORMATIONOPERATOR3D($,$,#29,1.,$);
...
#390=IFCSHAPEREPRESENTATION(#5,'Body','MappedRepresentation',(#998625));
#391=IFCSHAPEREPRESENTATION(#14,'Plan','MappedRepresentation',(#998626));
...
#402=IFCPRODUCTDEFINITIONSHAPE($,$,(#390,#391));
...
#8595=IFCDOOR('1$OQR48_T1GgA0Zocaupp',#33,'door2',$,'TLM3023',#997391
,#402,'1240474',2299.9999999998,3000.00000000001);
#8596=IFCDOOR('1$OQR48_T1GgA0ZocaujLj',#33,'door1',$,'TLM3023',#997883
,#402,'1246222',2299.9999999998,3000.00000000001);
...
#998625=IFCMAPPEDITEM(#348056,#44);
#998626=IFCMAPPEDITEM(#348057,#44);
...

```

Fig. 7. Illustration for reassigning the instance name of the IFC file, whose sample file fragment has been shown in Fig. 6.

complexity depends on the tree structure, and we leave it for future work. In our implementation, we typically use hash map to save the data instances.

4. Experimental results and discussions

The above-mentioned algorithm has been implemented in a content-based IFC compression tool, called *IFCCompressor*, in Visual C++ 2012 under Windows 7. All the experiments are run on an Intel i5-3210M processor with 8 GB memory. Fig. 8 shows a screen shot of the *IFCCompressor* tool. On the left, the user should first select an IFC file and then click the button “Start” to perform the content-based compression algorithm. After the compression, the compressed IFC file is automatically saved in the same directory. Alternatively, one can choose the option “Float-Point Rounding (FPR)” to obtain a compression constrained by an error bound, where the final compression result is displayed in the “Compression Status” area. On the right, the comparison of file sizes between the original file and the compressed one is displayed.

4.1. Metrics for quantifying compression and similarity

We use two metrics to quantify the experimental results between the original file and its compressed file, which are the compression rate and the similarity rate.

4.1.1. Compression rate (CR)

The compression rate [12] is used to quantify the reduction in data instances relative to the original data instances. It is defined as:

$$\text{Compression rate (\%)} = \frac{n-m}{n}, \quad (1)$$

where n is the total number of data instances in the original IFC file and m is the number of data instances in the compressed IFC file. The compression rate is often notated as a percentage. The more redundant instances an IFC file has, the higher compression rate our algorithm has.

4.1.2. Similarity rate (SR)

The similarity rate presented by Lee et al. [20] is used to quantify the similarity between the original IFC file and its compressed file. It is defined as the rate of the number of matching instances in File A to

the instances in File B divided by the total number of instances in File A, i.e.

$$\text{Similarity rate (\%)} = \frac{\text{Number of matching instances in File A to File B}}{\text{Total number of instances in File A}}, \quad (2)$$

where “Number of matching instances” denotes the number of instances in File A that are identical to the instances in File B while ignoring the instance name. This similarity rate has been developed to show how much information in File A is preserved in File B during the data exchange. It defines the percentage of information in the comparing file that is the same as the information in the compared file. The similarity rate is less than or equal to 100%. As adopting the similarity rate to our evaluation, the original and compressed IFC files should be “flattened” first, i.e., making each IFC file in a structure that does not include any referencing or inheritance structure by replacing the reference id with its actual property values [20]. This overcomes the difference of reference id included in attribute values when comparing pairs of data instances.

4.2. Test cases

To evaluate the performance of the content-based IFC compression algorithm, we test the presented algorithm on six different IFC files. The IFC files were exported through several commercial BIM modeling software platforms (including *Autodesk Revit Architecture* and *Graphisoft ArchiCAD*) without any manual modification to the files. The corresponding IFC models are visualized in Fig. 9, which are referred to as M1–M6. Table 1 gives the details of test case files used in the section, where “Size(MB)” is the size of the original IFC file, “#instances” is the number of data instances within the original IFC file, and “Exporter” indicates the specific software and its system version which exports the corresponding file.

Our algorithm is tested on the six IFC files in Table 1 using the two metrics in Eqs. (1) and (2). Table 2 shows the compressed IFC file size, compression rate and similarity rate. In this table, all the compressed files are acquired by our algorithm without FPR error bound, so the similarity rate of each file is 100%. The compression rate is generally very high (the average is 40.32%), with a maximum of 61.96% in the M6 file and a minimum of 14.17% in the M2 file.

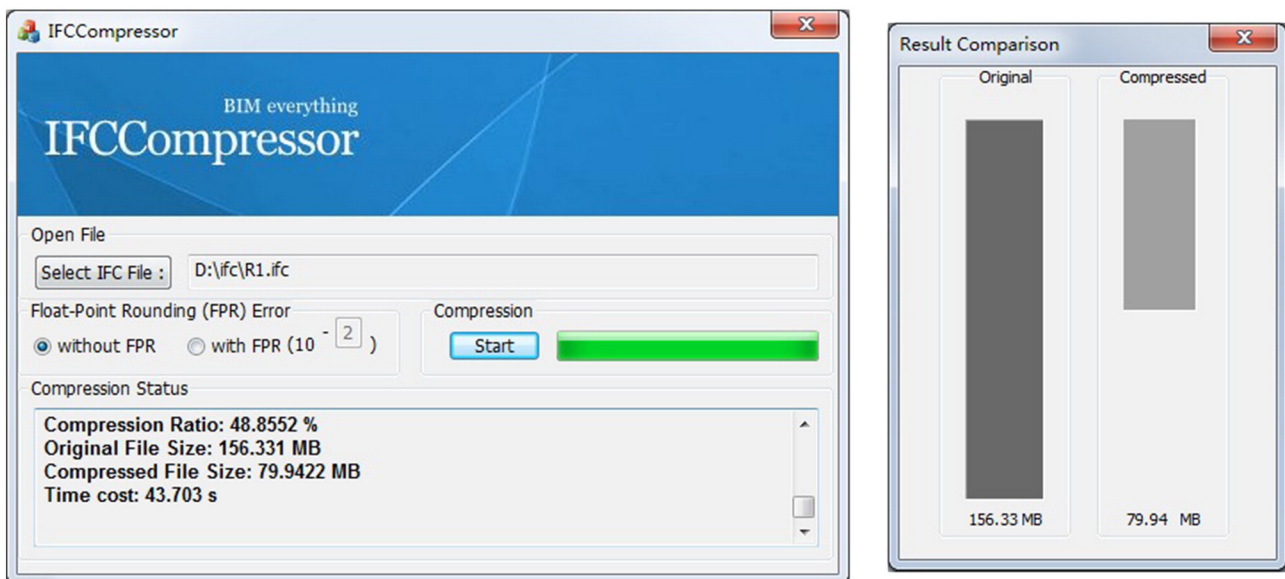


Fig. 8. A screen shot of the *IFCCompressor* tool.

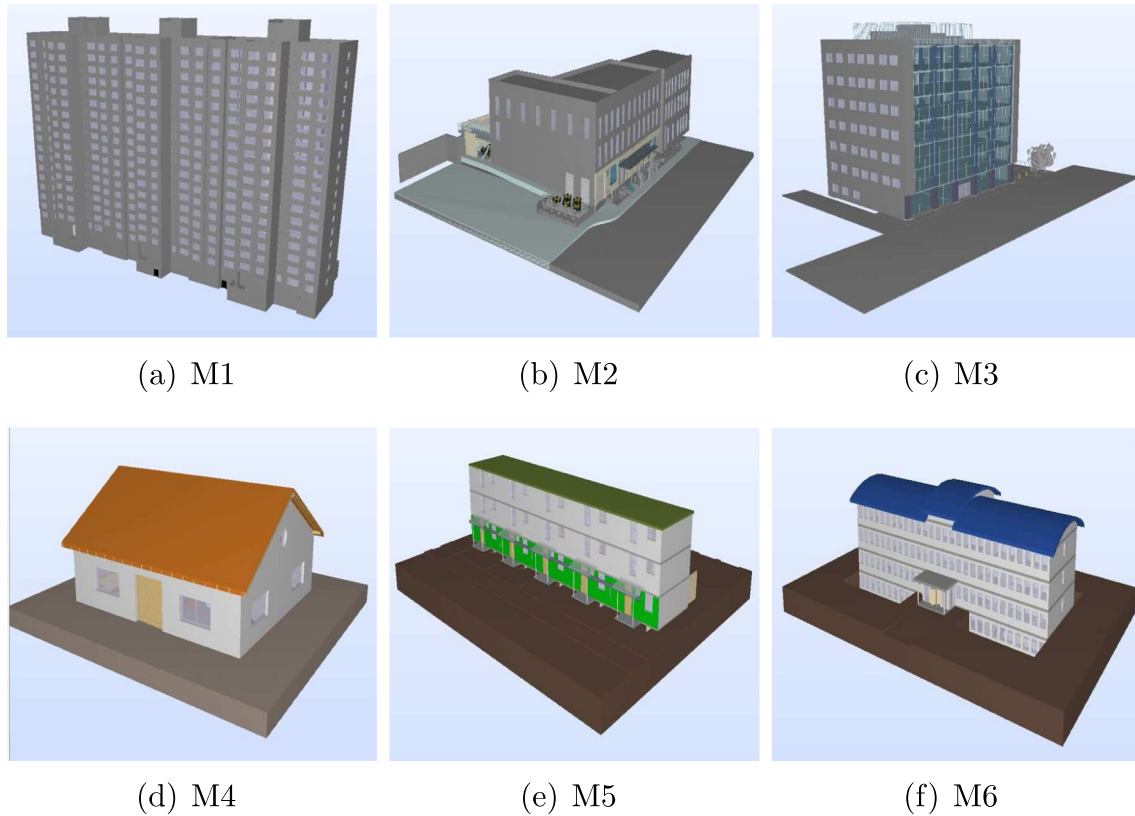


Fig. 9. Visualizing the models of six IFC files tested in the experiments.

The reason for such a big difference in compression rate mainly lies in two factors. One factor relates to the shape/structure of the original IFC model. If the model itself contains a large number of duplicated components, there may be many redundant data instances in the original file. For instance, the case with the second high compression rate (55.19%) in the M1 file is an apartment building, which includes many repeated data instances such as many staircases with regular shapes.

The other factor is the exporting software. The software and its system version which produced the original IFC file mainly determine the quantity of the redundancy. If an exporting software has been developed for solving this redundancy problem, there will be less redundancy information in the exported IFC file, which will lead to a low compression rate using our algorithm. Unfortunately, it seems that the older versions *Revit 2009* and *ArchiCAD 11* used in our experiments rarely take into account the IFC exporting optimization, while the new versions may have made some optimizations and can export a more compact IFC file. For instance, the two cases with low compression rates are the M2 (14.17%) and M4 (24.56%) files, which are exported through the relatively new versions *Revit 2012* and *ArchiCAD 14*, respectively. The *IFCCompressor* presented in this paper can be considered as a complementary tool for the existing IFC exporters.

Table 1
The details of six IFC files tested in the experiments.

Models	Figure	Size(MB)	#instances	Exporter
M1	9(a)	55.8	853,186	<i>Revit 2011</i>
M2	9(b)	25.7	453,586	<i>Revit 2012</i>
M3	9(c)	7.1	117,553	<i>Revit 2009</i>
M4	9(d)	4.0	82,226	<i>ArchiCAD 14</i>
M5	9(e)	3.2	62,667	<i>ArchiCAD 11</i>
M6	9(f)	2.7	49,664	<i>ArchiCAD 11</i>

In addition to the compression rate in Table 2, we also analyzed the compressed contents of the files. Table 3 lists the top six IFC entities included in each test model with the largest number of redundant instances. In this table, we found that the IFC entities named *IfcCartesianPoint*, *IfcAxis2Placement3D* and *IfcPropertySingleValue* are the highest redundant entities in almost every model. In the IFC specification, the entity *IfcCartesianPoint* defines the coordinates of a point in 2D/3D space, *IfcAxis2Placement3D* denotes the location and orientation in 3D space, and *IfcPropertySingleValue* defines a property object which has a numeric or descriptive value assigned. The results in Table 3 suggest that the geometry coordinates/location/orientation and properties of models often carry the largest number of redundant instances. Furthermore, the geometry shapes of complex models may also affect the compression rate of redundant entities. For example, the entity *IfcPolyLoop* (bounding loop of planar region in space) is ranked as a high redundant entity in some models (e.g. M1, M2, M4 and M5), while it is ranked as relatively low in others (e.g. M3 and M6).

Table 2
The experimental results using our algorithm.

Models	Size1 ^a (MB)	Size2 ^b (MB)	CR ^c	SR ^d	Time(s)
M1	55.8	28.2	55.19%	100%	17.7
M2	25.7	21.7	14.17%	100%	6.6
M3	7.1	4.8	30.93%	100%	2.2
M4	4.0	2.7	24.56%	100%	1.1
M5	3.2	1.4	55.13%	100%	1.0
M6	2.7	1.2	61.96%	100%	1.1

^a "Size1" is the original IFC file size.

^b "Size2" is the compressed IFC file size using our algorithm without FPR.

^c "CR" is the compression rate computed by Eq. (1).

^d "SR" is the similarity rate computed by Eq. (2).

Table 3

The top six IFC entities included in each test model with the largest number of redundant instances.

IFC entities	M1	M2	M3	M4	M5	M6
IfcFace	✓	✓		✓	✓	
IfcPolyLoop	✓	✓		✓	✓	
IfcPolyLine						✓
IfcDirection						✓
IfcSurfaceStyle				✓		
IfcCartesianPoint	✓	✓	✓	✓	✓	✓
IfcFaceOuterBound	✓	✓		✓	✓	
IfcAxis2Placement2D			✓			
IfcAxis2Placement3D	✓	✓	✓		✓	✓
IfcPropertySingleValue	✓	✓	✓		✓	✓
IfcShapeRepresentation				✓		

4.3. Result comparison with FPR error bound

The problem of float-point rounding (FPR) error might be caused during IFC data storage and exchange between various software platforms. A typical example is that the coordinates of many *IfcCartesianPoint* data instances may be exported as the results that appear to be incorrect by very small amounts [12]. For example, the geometry coordinates of a 3D point (0.0, - 0.560000, 0.0) may be exported as (0.0, - 0.559999, 0.0) instead of - 0.560000. This is not a problem or a limitation of the existing IFC exporters, and it occurs because the Institute of Electrical and Electronics Engineers (IEEE) 754 floating-point standard requires that decimal numbers should be stored in binary format. To minimize the effects of floating point arithmetic storage inaccuracy, our algorithm also offers FPR error bound for identifying identical data instances, which can achieve a higher compression rate than the previous lossless compression algorithm.

Our IFCCompressor tool provides the “Float-Point Rounding (FPR)” option, which processes the FPR errors of decimal numerical values of attribute values and in this way contributes to the IFC optimization. We deal with the FPR problem mainly for the geometry coordinates of 2D or 3D points represented in the entity *IfcCartesianPoint*, and in this way correct the rounding errors with the predefined precision. In practice, the users can set an error bound on the FPR option to round the values of geometry coordinates.

Fig. 10 compares the file sizes for six test models compressed using our algorithm (with or without FPR). Here we typically set the FPR error bound as 10^{-2} . As a result, we found that the contribution to compression rate using FPR is very limited when compressing

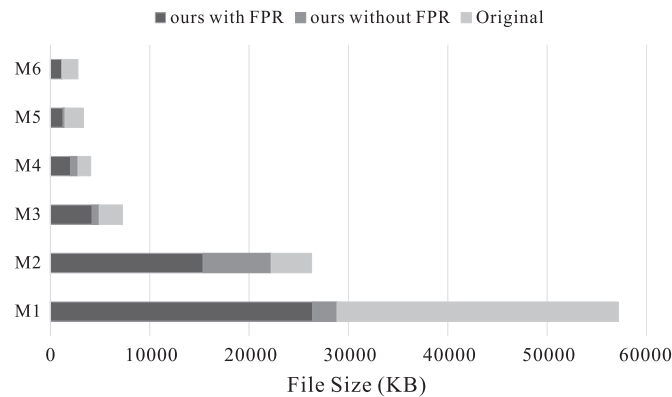


Fig. 10. Comparing the file sizes for six test models compressed using our algorithm (with or without FPR), where the FPR error bound is typically set as 10^{-2} .

Table 4

Comparing the compression rate for six test models compressed using our algorithm (with or without FPR) and Solibri IFC Optimizer, respectively.

Model's name	CR without FPR	CR with FPR (10 ⁻²)	CR using Solibri
M1	55.19%	60.07%	50.12%
M2	14.17%	27.79%	12.86%
M3	30.93%	37.29%	26.50%
M4	24.56%	40.48%	23.30%
M5	55.13%	59.87%	52.71%
M6	61.96%	62.81%	55.88%

small models, but usually increases with large and complex models. Table 4 shows the corresponding compression rates for test cases using our algorithm with or without FPR, where the file sizes of small models (e.g. M6) are rarely changed during compressing with FPR, in contrast to large models (e.g. M2). Furthermore, unlike the results of compression by the Solibri IFC Optimizer (see the right column in Table 4), our algorithm (with or without FPR) can make the file sizes smaller, leading to higher compression rate for all models.

Fig. 11 shows the relative compression rate for six test models in different FPR precisions. The relative compression rate is the absolute value of difference between the current compression rate and the minimal compression rate (i.e. FPR precision is set as 10^{-6}). The result in Fig. 11 shows that the compression rate of all models increases as the FPR precision decreases (from 10^{-6} to 10^0).

4.4. Comparison with Solibri IFC Optimizer

Solibri IFC Optimizer (or named Solibri for short) [19] is a well-known content-based IFC compression tool. Based on this commercial tool, Pazlar and Turk [12] have examined and discussed the compression results of some simple models. This subsection will compare our work with Solibri for the six test models. Table 4 shows the comparison result of compression rate between our algorithm and Solibri. The result suggests that our algorithm, even without FPR error bound, still achieves higher compression rate than Solibri for all models.

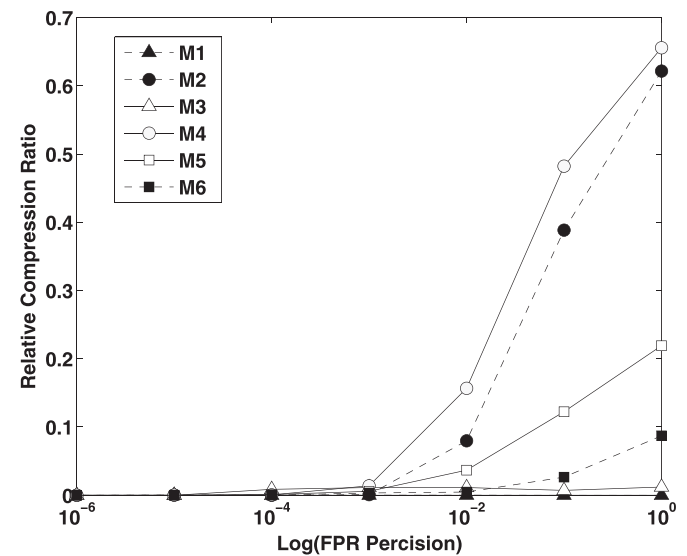


Fig. 11. Illustrating the relative compression rate for six test models at different FPR precisions. As the FPR precision decreases (from 10^{-6} to 10^0), the compression rate of all models increases.

Table 5

Illustrating the number of some redundant instances that are not eliminated by Solibri, but are re-compressed by our algorithm (without FPR), where “#Mi” is the number of redundant instances re-compressed by our algorithm for the model Mi.

IFC entities	#M1	#M2	#M3	#M4	#M5	#M6
<i>IfcLocalPlacement</i>	1988	2020	219	290	0	109
<i>IfcMappedItem</i>	2005	1170	1426	51	183	535
<i>IfcProductDefinitionShape</i>	11,983	1041	1487	67	487	1040
<i>IfcShapeRepresentation</i>	20,288	1437	2003	75	612	1150
<i>IfcStyledItem</i>	7031	277	77	551	231	184

However, since Solibri does not provide its compression process in detail, we cannot find out how and how many redundant data instances are eliminated during the process. To analyze the difference between our algorithm and Solibri, we first compress each of the six models (from M1 to M6) using Solibri and export the compressed IFC file. Then the compressed file is re-compressed using our algorithm. The process can check whether the file compressed by Solibri can be further optimized by our algorithm. Based on the above process, Table 5 shows some redundant instances that are not eliminated by Solibri, but are re-compressed by our algorithm (without FPR). In another way, we compress each original model first with our algorithm, and then export the compressed IFC file into Solibri for re-compression. The result is that no redundancy is found with Solibri, which shows that our algorithm obtains a more compact IFC file than Solibri.

4.5. Case study

In order to test the performance of our algorithm in large projects, the BIM model of an apartment building in Yunnan province in China is selected as a case study. The architectural design model was generated in

Revit Architecture 2011, and brought into our tool by the IFC format. The IFC model was also used for our previous studies on IFC-based path planning for 3D indoor spaces [10]. The original IFC file includes more than 2.8 million data instances and its file size is about 156.0 MB. The corresponding model is visualized in Fig. 12.

In the case study, we first apply our algorithm to the original IFC file through the iterative compression procedure, which eliminates the redundant data instances. The compression rate without FPR is 48.86%, and that with FPR (precision is set as 10^{-2}) is 72.13%. The time cost of compression process is 40.2 s, which is almost the same as that by Solibri (about 40s). In contrast to the compression rate (46.89%) using Solibri for optimizing the same file, our algorithm (with FPR or without FPR) can make the file size smaller, leading to a higher compression rate for the same model.

To analyze the compressed content of the model in the case study, we did two experiments, as shown in Figs. 13 and 14. In the first experiment, we first count the number of removed data instances for each IFC entity, and then compute the corresponding proportion of all removed data instances. For example, the total number of removed data instances for all entities is about 1352 K, of which 161 K instances belongs to *IfcFace*, so the proportion of removed *IfcFace*'s instances to the total removed instances is 11.9%. Fig. 13 shows these IFC entities which have the high redundant instances included in the original model. The result suggests that the geometry related entities (95.2%) have a higher proportion to removed instances than non-geometry related entities (4.8%) in the case study. For the geometry related entities, the four entities (i.e. *IfcCartesianPoint*, *IfcPolyLoop*, *IfcFaceOuterBound* and *IfcFace*) have a higher proportion than others. For the non-geometry entities, the proportion to removed instances for *IfcPropertySingleValue* is 2.4% (i.e. half of 4.8%), but far less than the geometry related entities.

In the second experiment, we count the number of removed data instances only for building element entities used in the model, and



Fig. 12. Visualizing the IFC model used in the case study.

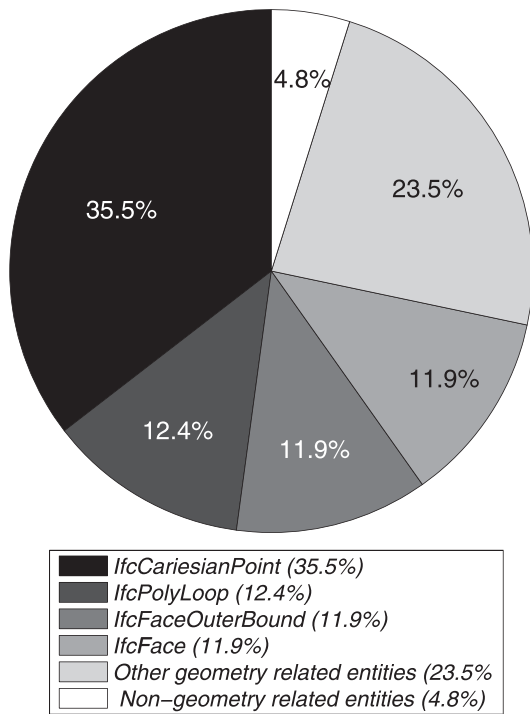


Fig. 13. Analyzing the compressed content of the model in case study when using our algorithm.

analyze their compression rate. The building element, named *IfcBuildingElement* in IFC specification, comprises all elements that are primarily part of the construction of a building, i.e. its structural and space separating system. The building elements such as walls, beams or doors, are all physically existent and tangible objects. As the main part of IFC entities, any *IfcBuildingElement* (e.g. its subtype *IfcDoor*) can be represented by one or several geometric representations, where the data instances of *IfcBuildingElement* will indirectly cite the geometry related entities (e.g. *IfcCartesianPoint*).

In order to analyze how much redundant information has been removed for each of the building elements (e.g. *IfcWall*, *IfcWindow* and *IfcDoor*), we define a new metric for quantifying the compression rate of the individual entity, especially for building element's entity. Given an IFC entity as *X*, the *compression rate of entity X* is defined as the rate

of the number of data instances to entity *X* in the compressed file divided by the number of data instances to entity *X* in the original file, i.e.

$$\text{Compression rate of entity } X (\%) = \frac{\text{Number of data instances to entity } X \text{ in compressed file}}{\text{Number of data instances to entity } X \text{ in original file}} \quad (3)$$

Fig. 14 shows the compression rate of entity in building elements used in the model. From this figure, we can see that the compression rate of many entities (e.g. *IfcSlab*, *IfcDoor*, *IfcCurtainWall*, *IfcStair*, *IfcWindow*, *IfcRailing* and *IfcPlate*) is very high and even more than 40%, while the compression rate of several other entities (e.g. *IfcRamp*, *IfcRoof* and *IfcWall*) is very low – close to zero. The average rate reaches 32.9%. The result of analysis indicates that the compression rate of different entities highly depends on the shape representation/complexity of the model, which may have great differences even within one model file. Note that the model tested in the case study is a multi-storeyed residential building, which has a large number of duplicated building components. For example, many doors and windows in every storey have the consistent geometrical representation, but with different local placements, where these consistent geometrical representations can be compressed. Thus, the result indicates that our algorithm is very effective for this kind of office/residential building models whose IFC files may have large numbers of redundancies.

4.6. IFC importing performance comparison

The original IFC file and its compressed file in Section 4.5 are reused for checking the performance of IFC importing, which aims to show whether our compression algorithm can offer faster loading for IFC applications. For convenience, the original IFC file in the previous case study is named R1.

We check the loading performance improvement by importing the original IFC file (R1 file) and its compressed one (using our algorithm without FPR) to the three most widely used IFC-supported software, i.e. *Autodesk Revit 2014 (Revit)*, *ArchiCAD 17 (ArchiCAD)* and *Solibri Model Viewer V9 (Solibri)*. After loading the R1 file and its compressed file into each of these software, we find that there is no distinct difference in terms of memory consumption.

In Table 6, the result shows that the compressed R1 file indicates shorter loading time compared with the original R1 file. In *Revit 2014*, it takes about 32 min to load the original R1 file and 30 min to load the compressed file. In *ArchiCAD 17*, the compressed R1 file is loaded in 2.19 min, while the original R1 file is loaded in 3.28 min. In *Solibri Model Viewer*, it takes only 0.52 min to load the compressed IFC file, compared against 1.04 min to load the original IFC file. The result shows that our

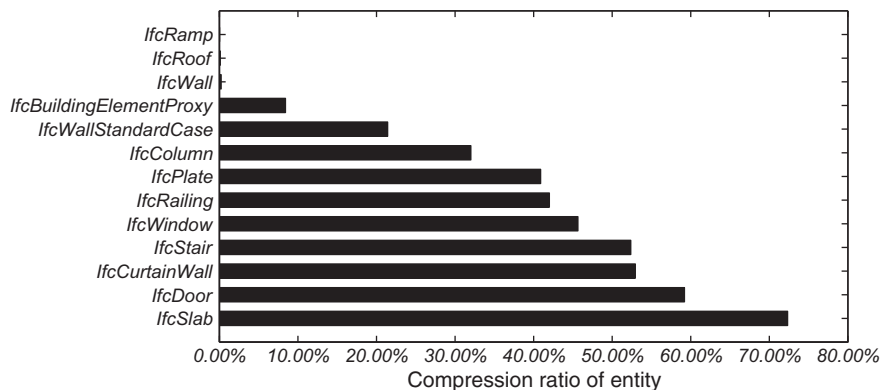


Fig. 14. Comparing the compression rate of entities in *IfcBuildingElement*.

compression algorithm can significantly reduce the loading time in both *ArchiCAD* and *Solibri*, and slightly reduce the loading time in *Revit*. Here, the memory consumption of the original R1 file and that of the compressed file are almost same. The reason for this result is that no matter what IFC toolkit or software is used, there are at least three stages to help reducing processing time for the compressed IFC file.

- Stage 1: Reading the IFC file from local disk/web server. During this stage, our algorithm can definitely decrease reading time, since a smaller number of data instances are read after redundant data instances are removed.
- Stage 2: Parsing the IFC file in memory. Our algorithm can also reduce the parsing time for IFC file due to the smaller search space for the data instances' attribute values containing reference id.
- Stage 3: Mapping the IFC file to the internal data structure of the BIM system and visualizing the IFC model. This stage always occupies the most loading time when importing an IFC file, and it will affect the improvement of loading time of compressed IFC file.

By removing redundant data instances in the original IFC file, our algorithm provides a possibility for those IFC toolkits to identify the instances whose attribute values share the same content, such as the two *IfcDoor* instances that share the same geometrical shape and attribute values. If the IFC toolkit can utilize this feature and avoid handling the same content repeatedly, it will save a lot of time. *Solibri* and *ArchiCAD* are probably such software, which take less time loading the compressed IFC file than the original. However, compared with *Solibri* and *ArchiCAD*, we have found that the IFC importer in *Revit* is not as optimized but takes more time to load the compressed IFC file and the original in our test.

The IFC file compressed by our algorithm can reduce the time cost when a software tries to read the IFC file from local disk or web server. Our algorithm also provides the possibility that can help IFC importer identify the instances having the same content, which avoids handling for many times. However, the practical effect highly depends on the utilized IFC toolkit itself.

4.7. Comparison with ZIP compression

Our algorithm does not compete with ZIP or other ZIP-related algorithms. Instead of focusing on how to encode the original IFC file or generating a new archive file format, we mainly focus on how to reduce redundant data instances by analyzing the content within the original IFC file. After the original IFC file is optimized by our algorithm, the file still remains an IFC file format (not ZIP or IFC-ZIP). Therefore, the ZIP or other ZIP-related algorithms are still applicable to the compressed IFC file. By combining ZIP and our algorithm, it is expected to achieve higher compression rate than ZIP alone.

Table 7 shows the compression rates of ZIP alone and of combination of ZIP and our algorithm, where "ZIP" denotes the compression rate using ZIP alone to the original IFC files, and "ZIP + Ours" denotes the compression rate using ZIP to the optimized IFC files processed after our algorithm. Here the compression rate is typically computed as the rate of reduction in file size relative to the uncompressed size, i.e.

$$\text{Compression rate (\%)} = 1 - \frac{\text{Size of compressed file using ZIP}}{\text{Size of uncompressed file}}$$

Table 6
IFC importing performance comparison.

Files	Size	<i>Revit 2014</i>	<i>ArchiCAD 17</i>	<i>Soliri</i>
R1	156.0 MB	32 min	3.28 min	1.04 min
Compressed R1	82.6 MB	30 min	2.19 min	0.52 min

Table 7

The comparison of compression rate between ZIP alone and combination of ZIP and our algorithm.

Models	ZIP ^a	ZIP + Ours ^b
M1	79.86%	88.71%
M2	80.32%	84.62%
M3	81.46%	87.41%
M4	81.41%	87.19%
M5	79.65%	90.25%
M6	78.48%	88.80%

^a The compression rate using ZIP alone to the original IFC files.

^b The compression rate using ZIP to the optimized IFC files processed after our algorithm (i.e. ZIP + our algorithm).

In the test, we select the six models from Table 1. The result in Table 7 shows that the combination of ZIP and our algorithm can achieve higher compression rate than ZIP alone. In addition, we also use *gzip* to conduct the compression experiment, and the result tends to be similar to that of the ZIP.

5. Conclusion and future work

Since large IFC file size has been an issue in the data exchange process, this paper introduces a content-based compression algorithm and develops a software tool named *IFCCompressor* for optimizing IFC files. The algorithm is achieved through an iterative compression procedure based on an IFC model's tree structure. The optimization procedure can be lossless or be constrained by float-point rounding (FPR) error bound. Compared with pure ZIP compression methods, our algorithm is based on comprehensive analysis of the structure and content of IFC data file without changing the file format. Unlike partial model extraction methods, our algorithm results in a complete IFC model but with a more compact IFC physical file. In contrast with *Solibri IFC Optimizer*, our algorithm can make the size of IFC files smaller. To evaluate the performance of our algorithm, the presented algorithm is tested on some large IFC files exported through several commercial BIM software platforms. The experimental results demonstrate that the presented algorithm is able to obtain higher compression rates than *Solibri*. In addition, an apartment BIM model in Yunnan province, China, is selected as a case study, which indicates that our algorithm is very effective for this kind of office/residential building models with a large number of duplicated components.

Our compression tool offers several potential benefits including suitable model size for data transmission, faster loading for IFC applications, smaller memory occupation and lower data storage costs for data management. The presented *IFCCompressor* tool is developed to relieve the redundancy problem. However, in the long run a redundancy check should be included in a normal IFC development procedure. Therefore, the *IFCCompressor* can be considered as an alternative and complementary tool for the existing IFC exporters. In addition, although the proposed algorithm was tested on IFC Version 2 × 3 case files, it could be applied to higher version (e.g. IFC4). Some other future work is addressed below.

- The *IFCCompressor* tool has been tested by the authors in many cases (and also tested by the anonymous reviewers), and it will be available to the designers around the world using BIM in actual/complex projects. As potential end users or beneficiaries, we hope that the designers take interest in testing the tool and give valuable further feedback, validation and confidence.
- In addition, although we have explained that our algorithm without FPR will not change the IFC model itself, we will do more tests to check if the use of this tool will have unintended consequences (i.e. data loss) during the generation of COBie data (extracted from IFCs) in the future.
- To make our algorithm more useful, we also plan to design some plugins embedded into some widely utilized software such as *Revit*.

The sources of this redundancy might be many, such as differences of model mapping mechanism, various possibilities offered by IFC specification, and even modeling methods in BIM software. Since the IFC files are mainly exported through different IFC toolkits or BIM software, the redundancy heavily depends on IFC translator development in these software. A possibly better way to tackle redundancy in the data representation is to provide IFC translator developers with very specific development guidelines. The proposed compression algorithm and the developed tool can also be used to analyze redundancy in IFC files. We expect that the proposed compression algorithm will open up a perspective for developing a formal and standard approach to removing redundancies.

Supplementary material

The online *IFCCompressor* tool and its demonstration can be accessed at: <http://cgcad.thss.tsinghua.edu.cn/liuyushen/IFCCompressor/> <http://dx.doi.org/10.1016/j.autcon.2014.10.015>.

Acknowledgments

The authors appreciate the comments and suggestions of all anonymous reviewers, whose comments significantly improved this paper.

The research is supported by the National Science Foundation of China (61472202, 61272229, 61003095) and the National Technological Support Program for the 12th-Five-Year Plan of China (2012BAJ03B07). The second author is also supported by the Chinese 973 Program (2010CB328003). The third author is also supported by the Chinese 863 Program (2012AA040902).

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] BuildingSMART, The IFC-Compatible Implementations Database Available from: <http://buildingsmart-tech.org/implementation/implementations2014>.
- [4] C. Eastman, J. Lee, Y. Jeong, J. Lee, Automatic rule-based checking of building designs, *Autom. Constr.* 18 (8) (2009) 1011–1033.
- [5] S. Jeong, Y. Ban, Computational algorithms to evaluate design solutions using space syntax, *Comput. Aided Des.* 43 (6) (2011) 664–676.
- [6] Z. Ma, Z. Wei, W. Song, Z. Lou, Application and extension of the IFC standard in construction cost estimating for tendering in China, *Autom. Constr.* 20 (2) (2011) 196–204.
- [7] Z. Hu, J. Zhang, BIM- and 4D-based integrated solution of analysis and management for conflicts and structural safety problems during construction: 2. Development and site trials, *Autom. Constr.* 20 (2) (2011) 167–180.
- [8] L. Zhang, R. Issa, Ontology based partial building information model extraction, *J. Comput. Civ. Eng.* 27 (6) (2013) 576–584.
- [9] J. Won, G. Lee, C. Cho, No-schema algorithm for extracting a partial model from an IFC instance model, *J. Comput. Civ. Eng.* 27 (6) (2013) 585–592.
- [10] 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, *Adv. Eng. Inform.* 27 (2) (2013) 189–205.
- [11] Y. Adachi, Overview of Partial Model Query Language Available from: <http://cic.vtt.fi/projects/ifcivr/tec/VTT-TEC-ADA-12.pdf#2002>.
- [12] T. Pazlar, Z. Turk, Evaluation of IFC optimization, Proceedings of CIB W78 Conference on Bringing ITC Knowledge to Work, 2007, pp. 61–66.
- [13] T. Pazlar, Z. Turk, Interoperability in practice: geometric data exchange using the IFC standard, *ITcon* 13 (2008) 362–380.
- [14] E. William East, N. Nisbet, T. Liebich, Facility management handover model view, *J. Comput. Civ. Eng.* 27 (1) (2013) 61–67.
- [15] J. Won, G. Lee, Algorithm for efficiently extracting IFC building elements from an IFC building model, Proceedings of ASCE International Workshop on Computing in, Civil Engineering, 2011, pp. 713–719.
- [16] S. Backas, SPADEX Final Report Available from: <http://cic.vtt.fi/vera/Documents/spadex-final-report.pdf>.
- [17] C. Kam, M. Fischer, Capitalizing on early project decision-making opportunities to improve facility design, construction, and life-cycle performance – POP, PM4D, and decision dashboard approaches, *Autom. Constr.* 13 (1) (2004) 53–65.
- [18] BuildingSMART, IFC Data File Formats and Icons Available from: <http://www.buildingsmart-tech.org/specifications/ifc-overview/ifc-overview-summary2014>.
- [19] Solibri, Solibri IFC Optimizer Available from: <http://www.solibri.com/solibri-ifc-optimizer.html#2014>.
- [20] G. Lee, J. Won, S. Ham, Y. Shin, Metrics for quantifying the similarities and differences between IFC files, *J. Comput. Civ. Eng.* 25 (2) (2011) 172–181.
- [21] P. Demian, P. Balatsoukas, Information retrieval from civil engineering repositories: importance of context and granularity, *J. Comput. Civ. Eng.* 26 (6) (2012) 727–740.
- [22] J. Beetz, J. Leeuwenand, B. Vries, IFCOWL: a case of transforming EXPRESS schemas into ontologies, *Artif. Intell. Eng. Des. Anal. Manuf.* 23 (1) (2009) 89–101.
- [23] L. Zhang, R. Issa, Development of IFC-based construction industry ontology for information retrieval from IFC Models, Proceedings of the 2011 EG-ICE Workshop, 2011.
- [24] H. Ma, K. Ha, C. Chung, R. Amor, Testing semantic interoperability, Proceedings of Joint International Conference on Computing and Decision Making in Civil and Building, Engineering, 2006, pp. 1216–1225.
- [25] G. Lee, What information can or cannot be exchanged? *J. Comput. Civ. Eng.* 25 (1) (2011) 1–9.
- [26] BuildingSMART, ifcXML Overview Available from: <http://www.buildingsmart-tech.org/specifications/ifcxml-releases2014>.
- [27] N. Nisbet, T. Liebich, ifcXML Implementation Guide (Version 2.0), 2007.
- [28] S. Sakr, Investigate state-of-the-art XML compression techniques Available from: <http://www.ibm.com/developerworks/library/x-datacompression/index.html#2011>.
- [29] 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.
- [30] T. Liebich, IFC 2x Edition 3 Model Implementation Guide (Version 2.0), 2009.