Automated Schedule Quality Assessment for 4D Models using Industry Foundation Classes

Parth Sachdeva (40081568)

Advisor: Dr. Mazdak Nik-Bakht

Report

ENGR 6991 Project and Report III

Department of

Building, Civil and Environmental Engineering

Submitted for Partial Fulfillment of Requirement for the Degree of

Master of Engineering in Civil Engineering

December 2019

ACKNOWLEDGEMENT

I would like to express my sincere gratitude to my advisor, Dr. Mazdak Nik-Bakht for providing his invaluable guidance, comments and suggestions throughout the course of the research. I would like to offer my sincere appreciation for providing learning opportunities through this research work.

I would also like to thank Ph. D student Abdelhady Hosny for his continuous support in the research. I am thankful to him for providing his time to solve the problems during the research.

It was impossible to deliver the content of this research work clearly without the help of Dr. Mazdak Nik-Bakht and Ph.D. student Abdelhady Hosny. Thank you again for providing your valuable time and support.

TABLE OF CONTENT

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

1.1 General

Building Information Modeling (BIM) goes way back in 1980s, however it was only in December 16, 2002, issue of LaiserinLetter™ which could be called as first widely popularized article on the topic, in which a debate conducted by industry analyst Jerry Laiserin, helped in spreading of new term or acronym to describe the emerging design technology in order to replace computer-aided design (CAD) [1]. Even though BIM became a buzz after this, the development of it in Canada was almost stagnant till 2017, According to National BIM Standard International BIM Report 2017, 78% Canadians think that BIM is the future for managing project information and 67% are currently using BIM [2]. BIM is an approach to design, construct, own, manage, operate, maintain, use, demolish or reuse a building by compiling data in one or more than one electronic format [3].

The growth in Canadian construction industry have been prolific in the recent years. All these construction projects are complex and requires planning and management from the beginning to the construction completion. Construction planning involves identifying all the required steps to build a structure, splitting them into defined activities and predicting all the uncertainties possible. In order to do this planning in a construction project, schedules are built, these schedules consist of specific tasks and/or activities to be achieved by a certain time arranged in a systematic manner. All this done by a key responsible party, mainly, a project planner. Now, the question arises - how can a project manager judge whether the schedule prepared by the planner is complete, accurate and workable?

Several studies in AEC industry show that with a good schedule the difference between completion times as well as costs are drastic for a same project. The development in Building Information Modeling (BIM) technologies have allowed to link the time schedule with the three-dimensional model to create 4D model. This 4D model allows planner to visualize the hierarchy of each activity in the whole process and then further helps in detecting trades clashes occurring due to overlapping at same time in general helps in better communication of the project's trade information and sequencing data.

1.2 Industry Foundation Classes (IFC)

Since the very beginning, the collaboration and the information exchange in BIM have been a major challenge and research aspect, as various disciplines use different types of software and platforms for their required use. After several efforts by various countries and/ or organizations, recently International Alliance for Interoperability (IAI) (now known as BuildingSMART), a subdivision of ISO with 10 chapters and more than 600 member organizations in 24 countries, came up with industry foundation classes (IFCs) (IAI 2003) IFC is defined using the ISO 10303 suite of specifications for data modelling and exchange, also known as STEP (Standard for the Exchange of Product Data). STEP consists of a range of specifications, a language for specifying data schema (STEP/Express, in which the IFC language is defined), a mapping (Part-21) for text file representation of models conforming to that schema, a mapping (StepXML) for XML file representation of models, and mappings to Application Programming Interface (API) for accessing models programmatically (Part-22, Standard Data Access Interface, or SDAI). The Part-21 mapping defines the IFC file format [4].

.

The latest version of IFC is IFC 2 X Edition 4 release and covers several domains of building construction, including architecture; structural engineering; heating, ventilation, and air conditioning (HVAC); facilities management; plumbing; and fire protection. They have a set of predefined rules that further defines the building data of a model. These rules help in forming a common and neutral interface to define classes of components for construction [5]. An IFC file consists of geometrical data and informational data or metadata. Many of the software applications or BIM tools currently used in AEC industry support import - export of IFC files. Also, this IFC file could be used for various another BIM uses.

1.3 Motivation

"Even if you're on the right track, you'll run over if you just sit there." – Will Rogers.

The information exchanges in construction project, required during planning phase is tremendous. Compiling and working with them manually is a hassle. This unease to work with data manually led us towards the path of Building Information Modeling (BIM), as BIM can handle all the information digitally.

Further, despite years of findings and research on Building Information Modeling in construction industry the major research focus has always been on the 2D schedules, whereas the 4D aspect of it which is combined 3D and schedule in IFC format has been neglected. All this generated an ample motivation to work upon generating automated schedule assessment models to review the 4D schedule in IFC format[3].

1.4 Problem Statement

IFC is an open system used for collaborating various information in a BIM project. IFC file format consists of temporal (schedule) and spatial information (geometry) of the 4D model. In this research instead on assessing the quality of a simple 2D schedule the focus is on a 4D schedule. A 4D model generally is an animation (in our project generated in Synchro) which is the 3D model geometry appearing in the sequence of attached activities, however, for large scale projects, any naked eye cannot detect the faults in the 4D model. Thus, automated schedule quality assessment of a 4D IFC schedule is a tool which can automatically access the quality of 4D IFC schedule which the responsible party at the construction site can use to access the quality of 4D schedule at site.

1.5 Objective

The main goal of the research is to develop a platform to assess quality of the schedule in the 4D IFC environment. The following are the set of sub-objectives defined in order to achieve the primary objective.

a) To conduct comprehensive review of previous research in 4D Model done using IFC file.
b) To find a software platform which imports and exports IFC 4D format, require for our study.
c) To study the semantics of selected software exported IFC file and recognize the various classes and attributes present in it which can be used in order to check the quality of a schedule.
d) To manually check the quality schedule for 4D models through an IFC File.
e) To develop a framework in order to automate the quality check algorithms into python functions.

1.6 Scope of report

The review of previous research of 4D model in IFC format from earlier literature and industry schedule analysis tools are presented in Chapter 2. It also highlights the semantics of an exported IFC file used to evaluate the schedule quality. Chapter 3 presents propose methodology of the research work starting with developing selecting 4D IFC platform, validating the metrics previously developed, manually validating the obtained algorithms. After that, the framework for the formation of automated models from the manually validated algorithms is described in chapter 4. The Chapter 5 consists the limitation of IFC and software applications. The conclusions of study are given in the Chapter 6. The recommendation for the future works is also presented in this chapter.

2 LITERATURE REVIEW

2.1 General

S,Daum and A.Borrmann , 2013 talk about developing a BIM query system in which they used ifcXML. By developing algorithms to form complex Ifc data into more meaning data. They developed algorithms of eliminating the middle component or the one which forms relationships and directly joining the components such as 3D geometry with schedule star and schedule finish. For example:- in Ifc file a component like IfcWallStandardCase is joined with the help of Relating and Related Object to Ifctask. And this IfcTask is connected to IfcTaskTime.

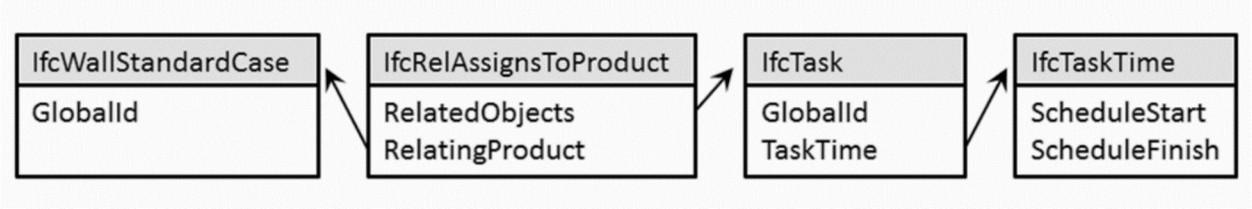

Figure 1 Relations between components and their construction times in the IFC [6]

They connected IfcWallStandard directly with IfcTaskTime by simplifying the links and forming algorithms to connect overlapping elements. They developed the patterns for the various 3D elements and then ran test to find them[6].

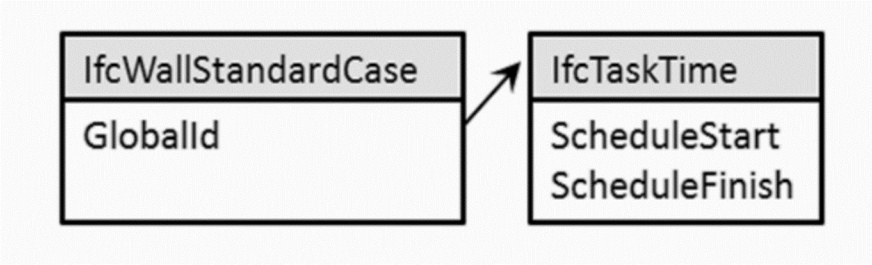

Figure 2 Simplified structure of components and their construction times[6]

An IFC file has various classes that support the transfer of scheduling, estimating, and resource data essential to 4D/5D BIMs. Scheduling information in the IFC file is related through set of tasks and their corresponding control information. These sets are modeled by instances of IfcTask, which describes any independent task in construction project; a task can further contain subtasks. Each IfcTask is distinguished from others by its globally unique ID (GUID). The schedule control is defined through subtypes of IfcControl, such as IfcScheduleTimeControl, which consist all scheduling attributes. The links between 3D elements and the tasks assigned to them are defined by subtypes of IfcRelationship [7].

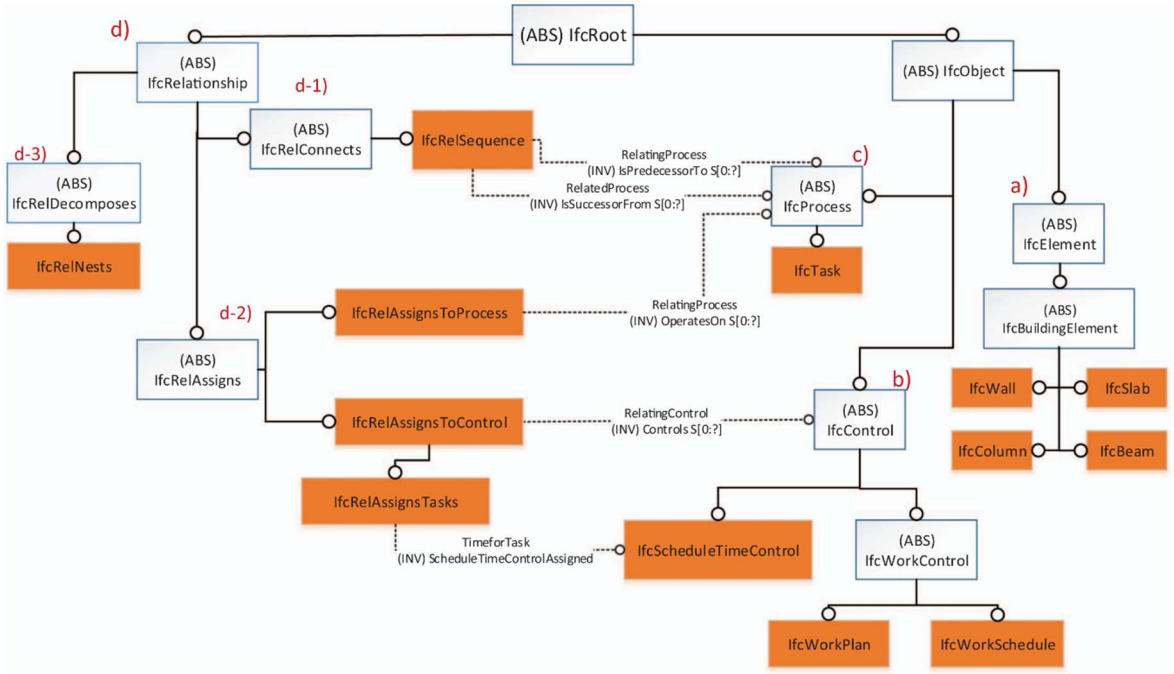

Figure 3 Hierarchical information of IFC entities[7]

These relationships define how each task is scheduled, to which building component(s) it is assigned, and how it relates to other tasks and subtasks. Fig. illustrates the relationships between various IFC classes, their attributes and their relationships with one another [7].

Jongsung and Ghang, 2013, penned down a paper on extracting a partial model from an IFC Model. In this they talk about filtering the IFC file, in which they take out required data from the file only. For this purpose, they developed algorithms in C# programming language. The purpose of their study was to get only required data from an Ifc file as an Ifc file is vast and Complex[8].

Further, Bansal (2011) utilized Geographic Information System (GIS) for space planning. In order to do workspace management at a jobsite they came up with the usage of GIS linking to the schedule component and then apply validation. It enabled time space conflict resolution prior to the construction. It would be difficult to enter space related data

And Lastly, the previous research done by Ashok and Rahul, 2018, define the checks need to develop to validate a 4D scheduled. In their research they used previous 2D model validation researches and came up with sets of metrics and developed pseudocodes in order to check the quality of the schedule. In their research, they talk about converting the various pseudocode for a programming language[9] .

3 METHODOLOGY

3.1 Selection of 4D IFC platform

With the increasing of BIM vendors in the market, there are quite a few numbers of 3D and 4D modeling software vendors in the market. BuildingSmart maintains a huge data base of all these software's IFC interaction, almost all the 3D software import IFC and all the 4D software import. However, from our project's aspect we were interested in finding a 4D software platform which can import and export IFC file. After conducting, an elaborate technological survey, which included contacting vendors, Synchro was the online platform which provided free access and could import and export IFC file.

Figure 4 4D model made in Synchro

However, even in Synchro IFC2x3 Coordination View 2.0 format could be exported. Thus, a 4D model in synchro was used in the project. The 4D model of sample office building in which schedule was attached to its activities. Then 4D model was exported in IFC format to receive IFC files. Note, these IFC files were exported in default settings of Synchro.

3.2 Metrics Validation

After receiving 4D IFC, the IFC file was studied properly. In the previous research (Ashok and Rahul, 2018) the metrics were divided as follow:

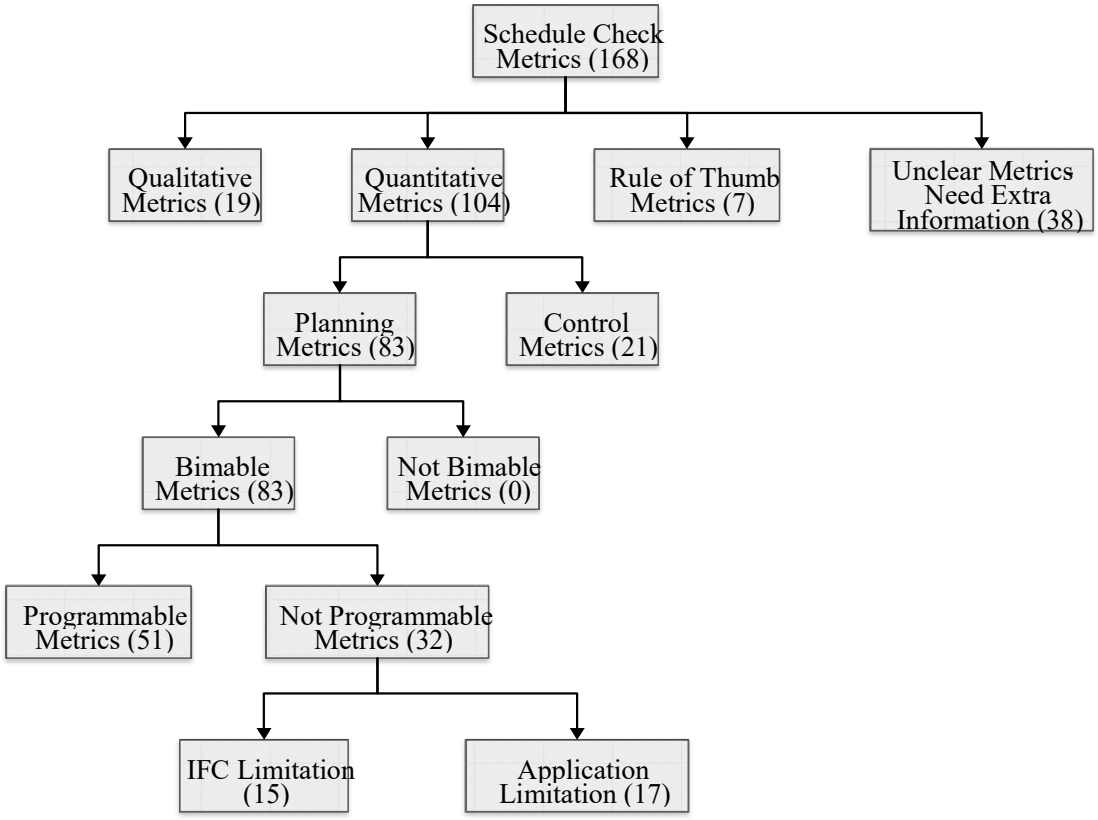

Figure 5 Classification of schedule check metrics[9]

These 51 metrics were validated again to verify if there is any change in the developed metrics or Synchro IFC export. The metrics were further divided into Parameter it was using, IFC entity required for the specified parameter and if its is exported via Synchro (Appendix A). In this validation it was found out that 76 parameters were exported by synchro. These parameters could only check 18 types of checks for which pseudo codes were written earlier.

3.3 Manual Validation of Algorithms

In order to verify the schedule quality assessment, algorithms were written in pseudocodes previously. These pseudocodes were further divided into two subtypes. Global functions and Main functions. In total there were 3 Global functions and 21 Main functions. For each function manually IFC file was read through and studied.

IFC file were read in a third-party application called "STEP File Analyzer". This software exports CSV and Excel files which helps you interpret data more clearly. TXT files readable in notepad were used for our project. The Table below explains meaning of certain symbols and words.

Table 1 Pseudocode terms [9]

Pseudocode terms	**Description (Meaning)**
X == Y	If X equals Y
X != Y	If X Not equals Y
X > Y	If X greater than Y
BEGIN FUNCTION	It begins an independent function which will be used in some other functions when necessary
BEGIN MAIN	It is the main (core) function of that particular metric

" "	Anything between " " these brackets will be printed exactly same.
PRINT "Message"	It will print word Message
PRINT Name	It will print whatever information is stored in Name

[ ]	Anything between [ ] these brackets is giving you information
Task [5]	Go to **5th** row in List called Task
Task [i]	Go to **i** row, where i is generally a variable like x
X [ID]	The type of data in **X** are ID
X [Name]	The type of data in **X** are Name
DECLARE	Create new
DECLARE LIST: X	Create new LIST called **X**
DECLARE NUMBER: Y	Create new Number called **Y**
STRING	A **STRING** is any finite sequence of characters (i.e., letters, numerals, symbols and punctuation marks)
DOUBLE	Number with decimal points
IfcTask.Name or X.Object	Anything after dot (.) is an entity e.g. IfcTask is an attribute and Name is an entity , X is a list and Object is a subtype
IfcTask.Name[i]	Go to **IfcTask** and Pick the **Name** in **i** row
If X.Name[i] == Y.Name[j]	When **i** row of **Name** in **X** equals **j** row of **Name** in **Y**
INPUT	What are the specific inputs required in function (along with attribute)
OUTPUT	What will be the outputs of function
OUTPUT and PRINT are not Same.	

Below are examples of manual validation of a global function ‘Total Task’ and a main function ‘Number of Logic Links with Lag’:

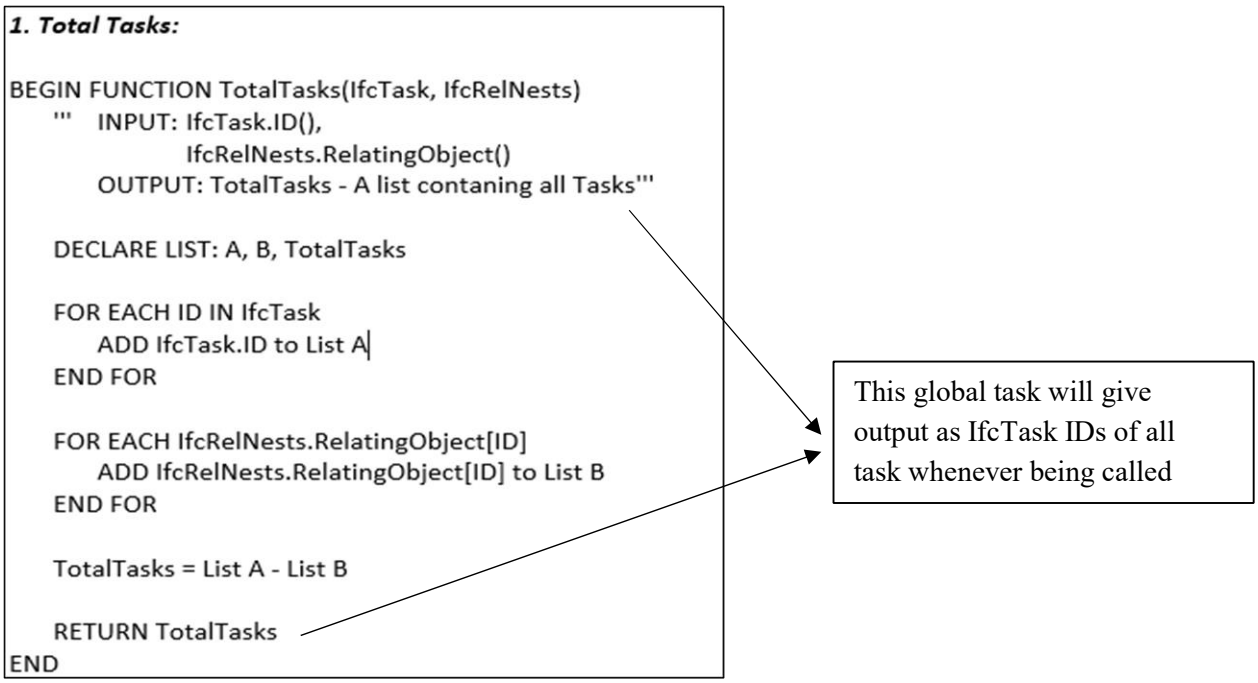

Figure 6 Pseudocode for Total Tasks [9]

1. The algorithm needs two IFC attributes as arguments IFCTASK and IFCRELNESTS.
2. The inputs are IfcTask.ID() and IfcRelNests.RelatingObject().
3. Declare lists A,B and Total Tasks.
4. List A consists of the attribute of 'id' from all 'ifctask' class of IFC file, which is also the line number all ‘IFCtasks’
5. And List B Contains the attribute 'Relatingobject' of all 'IFCrelnests' class of IFC file, which is the ‘5th’ entity in that attribute Below is how the semantics of a single component of an attribute looks like. The highlighted part is what we are looking for in the algorithm.

```
#801765      IFCTASK('2rNxWCGkH1GB$1IJpUfLZ1',#5,'Office    Building,    Core    &
Shell','1',$,'1F8503E8-5083-439F-9F27-57640A607DE0',$,$,.F.

#801867
IFCRELNESTS('25CLRnziP9JAVShyuk0LcZ',#5,$,$,#801774,(#801783,#801792,#801807,#801
822,#801837,#801852));
```

6. Further, output is list "Total Tasks = List A – List B"

2. Number of Logic links with Lag /

Lag % = # of Logic links with lag × 100/ # of logic links

```
BEGIN MAIN LagLogic (IfcRelSequence, IfcTask, AllowablePercent = 5)

    ''' INPUT: IfcRelSequence.TimeLag()
               IfcRelSequence.RelatingProcess()
               IfcRelSequence.RelatedProcess()
               IfcTask.ID()
               IfcTask.Name()
               IfcTask.TaskID()'''

    DECLARE LIST: Temp[RelatingProcess, RelatedProcess]
    DECLARE NUMBER: Lag = 0, NoOfLogicLinks = 0

    NoOfLogicLinks = Count Total IfcRelSequence.ID

    FOR EACH TimeLag IN IfcRelSequence
        if(IfcRelSequence.TimeLag[i] > 0)
            ADD one to Lag
            ADD IfcRelSequence.RelatingProcess[i], IfcRelSequence.RelatedProcess[i] to Temp
            END IF
        END FOR

    PRINT CALL FUNCTION MeasureCheck(Numerator: Lag, Denominator: NoOfLogicLinks, value:
    AllowablePercent, Condition: >, MetricName: "Lag% =")

    FOR EACH item IN Temp
            FOR EACH item IN IfcTask.ID
                IF (IfcTask.ID == Temp.RelatingProcess)
                    PRINT IfcTask.Name, IfcTask.TaskID
                END IF

                IF (IfcTask.ID == Temp.RelatedProcess)
                    PRINT IfcTask.Name, IfcTask.TaskID
                END IF
            END FOR
    END FOR
END
```

Printing the tasks that have missing logic

Figure 6 Pseudocode for Number of Logic Links with Lags [9]

1. The algorithm needs three arguments IfcRelSequence, IfcTask
2. And a predefined allowable percentage which is an integer value.
3. From these attributes the inputs required are IfcRelSequence.TimeLag(), IfcRelSequence.RelatingProcess(), IfcRelSequence.RelatedProcess(), IfcTask.ID(), IfcTask.Name(), IfcTask.TaskID().
4. Further, it declares a list Temp(Relating Process, Related Process) and two numbers Lag and no. of logic links.
5. The algorithm proceeds with a for loop of adding the 1 to the number lag every time timelag (shown shaded below) is greater than 0.

```
#803494=          IFCRELSEQUENCE('0NKTGP8d14gOrt4ftqngtD',#5,$,$,#801783,#801792,-
0.,.FINISH_START.);
```

6. Now, add the corresponding Relating Process and Related Process to the list Temp. As shown shaded in the examples below respectively.
7. Further, once the total number of lag is there, function 'MeasureCheck' is called and measure is checked, which is Lag% = number of logic links*100/ number of logic links is calculated, and it should be within the defined condition.

```
#803494=          IFCRELSEQUENCE('0NKTGP8d14gOrt4ftqngtD',#5,$,$,#801783,#801792,-
0.,.FINISH_START.);
#803494=          IFCRELSEQUENCE('0NKTGP8d14gOrt4ftqngtD',#5,$,$,#801783,#801792,-
0.,.FINISH_START.);
```

8. Once the list is complete another for loop to find any IfcTask.Id() equal to Ifc.RelatingProcess() IfcTask.Id() equal to Ifc.RelatedProcess() is run, if any of them is equal the corresponding Relating/Related process's name and Id are printed.

In this way each function developed were validated manually. Although majority of the functions were found to be correct, however, functions such as number of activities with out-of-range total float, number of activities with Negative total float and near Criticality Rate cannot be verified because while exporting IFC file from synchro IFC attribute such as free floats are not exported.

4AUTOMATIC MODEL VALIDATION FRAMEWORK

After the manual validation of algorithms were done, it was found out that to automate all these metrics, the inputs required from an IFC file are to be extracted first. For that some, predefined functions were necessary in the first place. Which was done by developing two functions in PYTHON. Namely, ParseIFCfile and Fetcher. These two functions were used in data mining of an IFC file.

4.1 ParseIfcFile

This function is developed with the idea of extracting the data of an attribute type from a large IFC file. The output from this function is a python dictionary which looks like this (in general):- {Global unique ID : [lists containing attribute entities]}. There is no working model, you cannot validate the model. So, As discussed earlier, we used a third-party application called "STEP File Analyzer" to generate .txt file as an input for these functions. In this function, there are two arguments that need to be defined, .txt file location, for example `D:\parth.txt` and the attribute name you are looking for , for example:- `IFCDATEANDTIME` for the sake of simplicity the function is broken down and explained in parts below. The full function is provided in appendix

1. To begin the function arguments are defined as mentioned above
2. Python regex module is imported to help in data mining.
3. Ifc.txt file is opened and is searched for the attribute data we are looking for.
4. Once the data is found it is further broken down into two parts global id or the line number in .txt file which would be the key of the dictionary and the body which consist of the entities of the defined attribute the values of the dictionary.
5. Once the Id or line numbers are extracted as "ID" and set aside.

```
def ParseIfcFile(file = "file_location.txt", attribute = "IFCATTRIBUTE"):
      import re
      IFC_FILE = open(file, "r")
      string_IFCFILE= "".join(IFC_FILE)
      lines = re.findall(r"#\d+= "+attribute+"\(.+",string_IFCFILE)
      string_lines = "\n".join(lines)
      ID1 = re.findall(r"\d+=",string_lines)
      string_ID1 = "\n".join(ID1)
      ID = re.findall(r"\d+",string_ID1)
      body1 =  re.findall(r""+attribute+"\(.+",string_IFCFILE)
      string_body1 = "\n".join(body1)
      body = re.findall(r"\(.+",string_body1)
      string_body = "\n".join(body)
      slice1 = re.split(",\(",string_body)
      t = string_body.count("))")
      l = string_body.count("),")
      print(len(ID))
      #print(l)
      #print(len(slice1))
```

6. Now, the body of entity is mined depending on the type of attribute. Each schedule and 3D element information are category one, second category consists of attributes consists of relationships between these scheduled tasks and their sub tasks and the third category is all other relationships between 3D and schedule.
7. The first category looks something like this:

   IFCTASK('00_P8aGHP20AWe0efTOMjx',#5,'ProjectTurnover','1.4.7',$,'ST00960',$,$,.F.,$);
8. Now there is a condition in the function that if there is no parenthesis inside the parenthesis (because it is not a relationship) it will work as condition one.
9. The regex finds all the patterns inside the parenthesis and splits them with the specified delimiter which in our case is ",".
10. Further with 'ID' as key and the split entities as values a dictionary is formed.

```
if (len(slice1) ==1):
    body = re.findall(r"\(.+",string_body1)
    string_body = "".join(body)
    slice2 = string_body.split("(")
    listw = [slice2[i+1] for i in range(0,len(body))]
    string_slice2 = "".join(listw)
    slice3 = string_slice2.split(")")
    #print(len(slice3))
    string_slice3 = "\n".join(slice3)
    #print(slice3)
    #print(listw)
    bodysplit = re.split(r"\,|\n;",string_slice3)
    #print(bodysplit)
    for elem in body:
        a = elem.count(",")
        b = a+1
    lista = [bodysplit[i:i+b] for i in range(0,len(bodysplit),b)]
    #print(len(lista))
    IFCdict = {}
    x = 0
    for eachid in ID:
        IFCdict[eachid] = lista[x]
        x+=1
    print(IFCdict)
    result = IFCdict
    return result
```

11. The second category is this:

```
IFCRELNESTS('1n5DUv3$n2PB2RP_h3Lb_u',#5,$,$,#803245,(#803254,#803269,#803284
,#803299));
```

12. Now in this condition the regex looks for the number of parenthesis “(“ and “)” in the body if there is any but the closing parenthesis “)” are together it works through condition two.
13. The regex finds all the patterns inside the parenthesis and splits them with the specified delimiter which is “(“ split then splits with “,” until it finds any “,(“ , once it does, it stops there and forms another list and starts putting the elements inside it and then stops at the “)”
14. Further with ‘ID’ as key and the split entity lists as values a dictionary is formed.

```
elif(t >>0):
    lista = []
    body = re.findall(r"\(.+",string_body1)
    string_body = "\n".join(body)
    bodysplit = re.split(",\(|\n",string_body)
    list1 = [bodysplit[i] for i in range(0,len(bodysplit),2)]
    string_list1 = "".join(list1)
    list3 = string_list1.split("(")
    list4 = [list3[i+1] for i in range(0,len(list1),1)]
    #print(list4)
    string_list4 = "\n".join(list4)
    list5 = re.split("\,|\n", string_list4)
    #print(len(list5))
    for elem in list4:
        r = elem.count(",")
    #print(r)
    y = r+1
    list6 = [list5[i:i+y] for i in range(0,len(list5),y)]
    #print(list6)
    #print(lista)
    list2 = [bodysplit[i+1] for i in range(0,len(bodysplit),2)]
    string_list2 = "\n".join(list2)
    list8 = re.split("\)\)|\n",string_list2)
    list9 = [list8[i] for i in range(0,len(list8),2)]
    #print(list9)
    i = 0
    listd = [[(list6[i]), [list2[i]]] for i in range(0, len(list6))]
    print(len(listd))
    IFCdict = {}
    x = 0
    for eachid in ID:
        IFCdict[eachid] = listd[x]
        x+=1
    print(IFCdict)
    result = IFCdict
    return result
```

15. The third category looks like this:

```
IFCRELASSIGNSTASKS('10TCReMfP0avkcGT6czY_n',#5,$,$,(#803324),$,#801751,#803325)
;
```

16. Now in this condition the regex looks for the number of parenthesis “(“ and “)” in the body if there is any but the closing parenthesis “)” are not together it works through condition three.
17. The regex finds all the patterns inside the parenthesis and splits them with the specified delimiter which is “(“ split then splits with “,” until it finds any “,(“ , once it does, it stops there and forms another list and starts putting the elements inside it and then stops at the “),” and again forms a list and starts putting elements in this list and at last stops at “)”.
18. Further with ‘ID’ as key and the split entity lists as values a dictionary is formed.

```
elif (l >>0):
    body = re.findall(r"\(.+",string_body1)
    string_body = "\n".join(body)
    body2 = re.findall(r"\(.+,\(",string_body1)
    string_body2 = "".join(body2)
    body3 = string_body2.split(",(")
    string_body3 = "".join(body3)
    string_body31 = "\n".join(body3)
    body31 = re.split("\n", string_body31)
    #print(body31[0])
    string_body32 = "".join(body31[0])
    k = string_body32.count(",")
    #print(k)
    d = k+1
    body4 = re.split(r"\(|,",string_body3)
    a = len(body4)
    c = a-1
    list1 = [body4[i+1] for i in range(0,c,1)]
    #print(string_body31)
    list2 = [list1[i:i+d] for i in range(0,len(list1),d)]
    #print(len(list2))
    para1 = re.findall(r",\(.+\),", string_body)
    string_para1 = "".join(para1)
    #print(string_para1)
    para2 = re.split(r",,", string_para1)
    string_para2 = "".join(para2)
```

```
            para3 = re.findall(r"\d+",string_para2)
            #print(len(para2))
            para4 = re.findall(r"\),.+\)", string_body)
            string_para4 = "".join(para4)
            para5 = re.split(r"\),", string_para4)
            list3 = [para5[i+1] for i in range(0, len(para4),1)]
            e = list3[0]
            u = e.count(",")
            w = u+1
            #print(e)
            string_list3 = "".join(para5)
            para6 = re.split(r"\,|\)", string_list3)
            list4 = [para6[i:i+w] for i in range(0,len(para6),w)]
            #print(list4)
            board = [[list2[i],[para3[i]],list4[i]] for i in range(0, len(list2))]
            #print(board)
            IFCdict = {}
            x = 0
            for eachid in ID:
                IFCdict[eachid] = board[x]
                x+=1
            print(IFCdict)
            result = IFCdict
            return result

ParseIfcFile()
```

4.2 FetchEntity

After parsing the IFC.txt file it is necessary to extract the particular entity of the attribute which would be needed in the metric check. For this the position of the entity inside the parenthesis is noted, if it is at 4th position an integer y = 5 (because numbering in python starts from 0) is added as argument, however if the attribute is a relationship attribute containing nested parenthesis, then z = is the parenthesis number ranging from 0 to 2 and y is the position of attribute in that parenthesis. The output of this function is a list of required entity.

```
import parseIFCfile
a = parseIFCfile.parseIFCfile()
def fetchEntity(a, y = 0 , z= 0):
    d = a.values()
    for each in d:
        entity = each[y]
        #print(entity)
        if (len(entity)>>0):
            print(entity[z])
        else:
            print(entity)
fetchEntity(a)
```

4.3 Model Validation Automation

After the above-mentioned functions were developed, the process of developing automated schedule checking functions was begun. Which is still in the process. The developed codes are in the appendix.

5 LIMITATIONS

As mentioned above in the project the IFC version taken for our project is not the latest version, which is why the patterns are prone to change once other software vendors start selling 4D export platforms. Also, the limitedness of the available software in another limitation as synchro is not capable to export some of the major IFC attributes such as resources. many IFC attributes such as critical activity, floats etc. are not exported by synchro.

In the data mining the key patterns developed and used for python module is working on patterns which are put together by synchro while exporting, if similar patterns are there in the model before exporting then the data mined most probably won't be the same. For example- if a task name consists of "," in its name then in python the name would also be split while splitting the entities. Which is why it necessary to carefully examine this thing in the model before exporting.

6 CONCLUSION AND RECOMMENDATION

6.1 Conclusion

The previous study on 4D validation have been the fundamental aspect of this research. IFC export from Synchro was studied via a third-party application STEP File Analyzer. After analyzing the IFC file the metrics developed previously were shortlisted based on the attributes available to test. Once the metrics were decided. All the pseudocodes developed were manually validated. All the data mining in the IFC.txt file was done manually, and all the algorithms were validated. Once, the codes were manually validated, it was identified that primary functions were needed to do automation of all the codes. Two primary functions namely ParseIfcFile and FetchEntity were developed in Python for the same. On the development of these functions, the automation of earlier mentioned pseudocodes has been made easier. Future research would involve working on this and developing a fully automated 4D scheduled Checker.

6.2 Recommendation

The current focuses only on schedule so, various other attributes involving geometry checks can be added for checking and more successful on-site efficiency could be achieved. The research can further be expanded to work more on the IFC semantics and importing back to the modeling software, in a way moving the focus of designing things manually in the software to textually typing the details into the IFC file and then importing back to the model. Interoperability, of 4D models which as mentioned in the research can be improved, Data mining done by the developed application can be used for further analytics of the construction industry and statistically analyzing except the checks what are the other reasons and metrics that affect construction projects.

REFERENCES

[1] Jerry laiserin ," comparing Pommes and Naranjas " LasierinLetter, no.15(2002) Available: www.Laiserin.com [Accessed 8 November 2019].

[2] Royal Architectural Institute of Canada, "BIM Explained," 2016. Available : https://www.raic.org/raic/bim-explained. [Accessed 8 November 2019].

[3] D. Smith, M. Tardif; “Building information modeling : a strategic implementation guide for architects, engineers, constructors, and real estate asset managers”.

[4] J. Steel, R. Drogemuller and B. Toth, "Model Interoperability in Building Information Modeling," *Software & Systems Modeling,* vol. 11, no. 1, pp. 99-109, 2012.

[5] “Definition and Implementation of Temporal Operators for a 4D Query Language,” *Journal of Computing in Civil Engineering,* Vol. 30, no., pp, , 2013

[6] “Definition and Implementation of Temporal Operators for a 4D Query Language,” *Journal of Computing in Civil Engineering,* Vol. 30, no., pp, , 2013

[7] H. Hamledari , B. McCabe, S. Davari and A. Shahi “Automated Schedule and Progress Updating of IFC-Based 4D BIMs,” *Journal of Computing in Civil Engineering*, Vol. 33, no. 4, pp, 159-171,2014.

[8] M. Laakso and A. Kiviniemi, "The IFC standard - a review of history, development, and standardization," *Journal of Information Technology in Construction,* vol. 17, pp. 134-161, 2012.

[9] A. Kavad and R. Dharsandia, “Schedule Quality Assessment for 4D Models using Industry Foundation Classes”, M.Engg. Project Report, Concordia University, Montreal, 2019

[10] BuildingSMART, "Open Standards - the basics," 2018. Available: https://www.buildingsmart.org/standards/technical-vision/open-standards/. [Accessed 30 October 2019].

APPENDIX A: METRIC VALIDATION

Sl. No.	Parameter	IFC entity	IFC Export (Yes/No)
1	Number of tasks	ifctask	yes
2	Number of tasks missing predecessors	ifcrelsequence	yes
3	Number of tasks missing successors	ifcrelsequence	yes
4	Number of incomplete tasks	ifcscheduletimecontrol, ifctask	yes
5	Total non summary tasks	ifcrelassigns, ifctask	No
6	Number of tasks with no predecessors	ifcrelsequence	yes

7	Number of tasks with no successors	ifcrelsequence	yes
8	Duration of each task	ifcscheduletimecontrol	yes
9	Total number of incomplete tasks with high duration	ifcscheduletimecontrol	yes
10	Duration of project	ifcscheduletimecontrol	yes
11	Number of logic links with FS relationships	ifcrelsequence	yes
12	Total number of logic links (FS, SS, FF, FS)	ifcrelsequence	yes
13	Relationship for each activity	ifcrelsequence	yes
14	Number of relationships between each activity	ifcrelsequence	yes

15	Total number of summary tasks	ifcrelassignstask, ifctask	yes
16	Total number of summary tasks with successors tasks	ifcrelassignstask, ifctask	yes
17	Duration of foundation activity		yes
18	Duration of framing activity		yes
19	Duration of curtain wall activity		yes
20	Duration of architectural activities	ifctask,	yes
21	Duration of electrical activities		yes
22	Duration of HVAC		yes

23	Duration of firefighting activities		yes
24	Duration of elevator activities		yes
25	Amount/status of activity completed		yes
26	Dangling tasks (tasks missing links to start or finish)	ifcrelsequence	yes
27	Incomplete effort tasks	ifcresource, ifcscheduletimecontrol	No
28	Total non-FS predecessor tasks	ifcrelsequence	yes
29	Total number of activities without finish relationship	ifcrelsequence	yes
30	Status of all activities	ifcscheduldetimecontrol	yes

31	The resources of each activity	ifcresource	yes
32	Number of labourers for each activity	ifcresource	yes
33	Based scheduled summed slack time	ifcscheduletimecontrol	yes
34	Evaluated scheduled summed slack time	ifcscheduletimecontrol	yes
35	Number of Resources for each task	ifcresource	No
36	Total summary tasks	ifctask, ifcrelassignstask	yes
37	Summary tasks with resource assigned to them	ifctask, ifcresource, ifcrelassignstask	No
38	Total effort tasks	ifcresource	No

39	Number of resource dependent activities	ifctask, ifcresource	No
40	Number of resource dependent activities missing resources	ifctask, ifcresource	No
41	Number of time dependent activities/ fixed start/finish date	ifcscheduletimecontrol	yes
42	Start and finish date of resource	ifcscheduletimecontrol, ifcreltask, ifcresource	No
43	Start and finish date of task	ifcscheduletimecontrol, ifcreltask	yes
44	Critical path length	ifcrelassignstask, ifcreltask, ifcscheuletimecontrol	yes
45	Total float	ifcscheduletimecontrol	yes
46	Project finish date	ifcscheduletimecontrol	yes

47	Delays added	ifcscheduletimecontrol	yes
48	Planned finish date	ifcscheduletimecontrol	yes
49	No. of critical activities	ifcscheduletimecontrol, ifcreltask	yes
50	Duration of critical activities	ifcscheduletimecontrol, ifcreltask	yes
51	Duration of activities	ifcscheduletimecontrol	yes
52	No. of near critical activities	Ifcrelassigntask, ifctask	yes
53	Predecessor of each critical activity	ifcrelsequence	yes
54	Number of critical paths	Ifcrelassigntask, ifctask	yes

55	Total number of incomplete tasks with negative float	ifcscheduletimecontrol, ifctask	yes
56	Total slack/float of each activity	ifcscheduletimecontrol, ifctask	yes
57	Number of logic links with lead/ -ve lag	ifcrelsequence, ifctask	yes
58	Total number of logic links	ifcrelsequence	yes
59	Number of logic links with lags	ifcrelsequence, ifctask	yes
60	Number of relationships with odd lag	ifcrelsequence, ifctask	yes
61	Activities with fixed logic finish to start, start to start, start to finish and finish to finish (fixed constraints)	ifcscheduletimecontrol	yes
62	Total constraints	ifcscheduletimecontrol	yes

63	Total non summary tasks	ifcrelassigns	yes
64	Early start, late start, early finish and late finish dates of all activities	ifcscheduletimecontrol	yes
65	Actual start and finish dates of all tasks	ifcscheduletimecontrol	yes
66	Planned start and finish dates of all tasks	ifcscheduletimecontrol	yes
67	Total number of tasks with actual date past baseline date	ifcscheduletimecontrol, ifctask	yes
68	Number of tasks with baseline finish date on or before status	ifcscheduletimecontrol, ifctask	yes
69	Total missing baseline tasks	ifcscheduletimecontrol	yes
70	Total non summary tasks	ifcrelassignstask	yes

71	Total tasks in future	ifcscheduletimecontrol	yes
72	Total actual finish before actual start tasks	ifcscheduletimecontrol	yes
73	Total number of resource assignments with misaligned dates	ifcresrource	No
74	Total number of resource assignments	ifcresrource	No
75	Total number of tasks complete before now	ifcscheduletimecontrol, ifctask	yes
76	Total number of tasks missing baseline finish date	ifcscheduletimecontrol, ifctask	yes
77	Number of Completed tasks	ifcscheduletimecontrol, ifctask	yes
78	Task planned to have been finished	ifcscheduletimecontrol	yes

79	Number of tasks in progress	ifcscheduletimecontrol, ifctask	yes
80	Total number of task without baseline	ifcscheduletimecontrol, ifctask	yes
81	Total number of task with late actual project start date	ifcscheduletimecontrol, ifctask	yes
82	Project duration	ifcscheduletimecontrol	yes
83	Total Number of milestones in the schedule	ifctask	yes
84	Cost of each task	ifccostvalue	No
85	Cost of work performed	ifccostvalue	No
86	Cost of work scheduled	ifccostvalue	No

87	Project calendar	Ifccalendardates	yes
88	Duration of each activity/phase	ifcscheduletimecontrol	No
89	Resources on milestones (in any)	ifctask, ifcresource	No
90	Maximum number of workers per square meter	ifcresource	No
91	Total effort tasks	ifctask, ifcresource	No
92	Activity/ schedule submission date	ifcscheduletimecontrol	yes
93	Total missing WBS tasks	Ifcrelnests, ifctask	yes
94	Cost of each resource	ifccostvalue	No

95	Total number of incomplete tasks with high cost	ifccostvalue	No
96	Activity expenses	ifccostvalue	No
97	Value of expenses	ifccostvalue	No
98	Total cost of activity	Ifccostvalue, ifctask	No
99	Budgeted cost of activity	Ifccostvalue	No
100	Actual cost of activity	Ifccostvalue	No
101	Number of labourers for critical activities	ifcrelassignstask, ifctask, ifcresource	No
102	Total project labourers	ifcresource	No

103	Start and finish dates of prime contractors	Ifceresource, ifccalendar, ifcscheduletmecontrol	No
104	Task under each milestone	ifctask	yes

APPENDIX B: AUTOMATIC MODEL VALIDATION FRAMEWORK

1. Function 1 - ParseIfcFile

```
def parseIFCfile(file = "D:\parth.txt", attribute = "IFCTASK"):
    import re
    IFC_FILE = open(file, "r")
    string_IFCFILE= "".join(IFC_FILE)
    lines = re.findall(r"#\d+= "+attribute+"\(.+",string_IFCFILE)
    string_lines = "\n".join(lines)
    ID1 = re.findall(r"\d+=",string_lines)
    string_ID1 = "\n".join(ID1)
    ID = re.findall(r"\d+",string_ID1)
    body1 =  re.findall(r""+attribute+"\(.+",string_IFCFILE)
    string_body1 = "\n".join(body1)
    body = re.findall(r"\(.+",string_body1)
    string_body = "\n".join(body)
    slice1 = re.split(",\(",string_body)
    t = string_body.count("))")
    l = string_body.count("),")
    #print(len(ID))
    #print(l)
    #print(len(slice1))
    if (len(slice1) ==1):
        body = re.findall(r"\(.+",string_body1)
        string_body = "".join(body)
        slice2 = string_body.split("(")
        listw = [slice2[i+1] for i in range(0,len(body))]
        string_slice2 = "".join(listw)
        slice3 = string_slice2.split(")")
        #print(slice3)
        string_slice3 = "\n".join(slice3)
        #print(slice3)
        #print(listw)
        bodysplit = re.split(r"\,|\n;",string_slice3)
        #print(bodysplit)
        for elem in body:
            a = elem.count(",")
            b = a+1
        lista = [bodysplit[i:i+b] for i in range(0,len(bodysplit),b)]
        #print(len(lista))
        IFCdict = {}
        x = 0
        for eachid in ID:
            IFCdict[eachid] = lista[x]
```

```
            x+=1
        #print(IFCdict)
        result = IFCdict
        return result
    elif(t >>0):
        lista = []
        body = re.findall(r"\(.+",string_body1)
        string_body = "\n".join(body)
        bodysplit = re.split(",\(|\n",string_body)
        list1 = [bodysplit[i] for i in range(0,len(bodysplit),2)]
        string_list1 = "".join(list1)
        list3 = string_list1.split("(")
        list4 = [list3[i+1] for i in range(0,len(list1),1)]
        #print(list4)
        string_list4 = "\n".join(list4)
        list5 = re.split("\,|\n", string_list4)
        #print(len(list5))
        for elem in list4:
            r = elem.count(",")
        #print(r)
        y = r+1
        list6 = [list5[i:i+y] for i in range(0,len(list5),y)]
        #print(list6)
        #print(lista)
        list2 = [bodysplit[i+1] for i in range(0,len(bodysplit),2)]
        string_list2 = "\n".join(list2)
        list8 = re.split("\)\)|\n",string_list2)
        list9 = [list8[i] for i in range(0,len(list8),2)]
        #print(list9)
        i = 0
        listd = [[(list6[i]), [list2[i]]] for i in range(0, len(list6))]
        #print(len(listd))
        IFCdict = {}
        x = 0
        for eachid in ID:
            IFCdict[eachid] = listd[x]
            x+=1
        #print(IFCdict)
        result = IFCdict
        return result
    elif (l >>0):
        body = re.findall(r"\(.+",string_body1)
        string_body = "\n".join(body)
        body2 = re.findall(r"\(.+,\(",string_body1)
        string_body2 = "".join(body2)
        body3 = string_body2.split(",(")
```

```
        string_body3 = "".join(body3)
        string_body31 = "\n".join(body3)
        body31 = re.split("\n", string_body31)
        #print(body31[0])
        string_body32 = "".join(body31[0])
        k = string_body32.count(",")
        #print(k)
        d = k+1
        body4 = re.split(r"\(|,",string_body3)
        a = len(body4)
        c = a-1
        list1 = [body4[i+1] for i in range(0,c,1)]
        #print(string_body31)
        list2 = [list1[i:i+d] for i in range(0,len(list1),d)]
        #print(len(list2))
        para1 = re.findall(r",\(.+\),", string_body)
        string_para1 = "".join(para1)
        #print(string_para1)
        para2 = re.split(r",,", string_para1)
        string_para2 = "".join(para2)
        para3 = re.findall(r"\d+",string_para2)
        #print(len(para2))
        para4 = re.findall(r"\),.+\)", string_body)
        string_para4 = "".join(para4)
        para5 = re.split(r"\),", string_para4)
        list3 = [para5[i+1] for i in range(0, len(para4),1)]
        e = list3[0]
        u = e.count(",")
        w = u+1
        #print(e)
        string_list3 = "".join(para5)
        para6 = re.split(r"\,|\)", string_list3)
        list4 = [para6[i:i+w] for i in range(0,len(para6),w)]
        #print(list4)
        board = [[list2[i],[para3[i]],list4[i]] for i in range(0, len(list2))]
        #print(board)
        IFCdict = {}
        x = 0
        for eachid in ID:
            IFCdict[eachid] = board[x]
            x+=1
        #print(IFCdict)
        result = IFCdict
        return result

parseIFCfile()
```

2. Function 2 – FetchEntity

```
import parseIFCfile
a = parseIFCfile.parseIFCfile()
def fetchEntity(a, y = 0 , z= 0):
    d = a.values()
    for each in d:
        entity = each[y]
        #print(entity)
        if (len(entity)>>0):
            print(entity[z])
        else:
            print(entity)
fetchEntity(a)
```