Graph reader: Reading graph from graphml and gml file format into network simulation

Pham Nguyen Quang

April 1, 2017

1 Introduction

1.1 Context

Graph reader project is part of the internship of the project from Bui Quang Minh, Pham Nguyen Quang, Nguyen Chi Thanh and Vo Huynh Quang Kiet at Frankfrut university of applied sciences. This project is about building a generic network optimizing system using evolutionary algorithm (EA). The graph reader project enables the users to read graph files from the graphml or gml file format as an input for the simulation without the restriction on the number of nodes, edges and attributes that can be read from the file.

1.2 Requirements

- Programming Language: C++11
- Operating system: Linux 64 bit
- Compiler: Cross g++
- Dependencies: Tinyxml2 library
- Recommended IDE: Eclipse IDE for C++, version Neon.2
  - To install, simply open the terminal in linux and type:`sudo apt-get install libtinyxml2-dev`

2 Structure

All header files (.h) and source code file (.cpp) are in the src/ folder. The files for unit testing is also included in the same directory. The program reads graph files as the input and return a network topology as output, the class topology is declared in Topology.h and its functions are defined in Topology.cpp. Likewise, the class for node, edge and attributes are declared in AnyNode.h, AnyEdge.h, Attributes.h and their functions are defined in AnyNode.cpp, AnyEdge.cpp and Attributess.cpp respectively.

3 Implementation detail

3.1 Attribuute implementation

The implementation for the attributes are based on how the attributes are written in the graphml format. The Attributes class declaration and definition can be found in Attributes.h and Attributes.cpp, the class contains:

- Attribute key: that can be used to identify the attribute. Implemented as string to allow combination of letters and number.
- Attribute name: the name of the attribute, can be uniquely identified by attribute key. Implemented as string to allow combination of letters and number.
- Attribute data: The actual data stored in the attribute. Implemented as string to allow combination of letters and number. Actual data type can be retrieved by reading the Attribute type.
- Attribute type: type of the data stored. Implemented as string. This attribute type can be retrieved in order to get the actual type of data for further processing.
- Getter and setter functions (more details on usage).

3.2 Node Implementation

The node class for this program is called AnyNode, their declaration and definitions can be found in AnyNode.h and AnyNode.cpp. The class AnyNode contains mainly:

- Node ID: The name that unique for every nodes (ID information can be read to uniquely identify the node). Implemented as string in order to allow combination of numbers and letters.
- Array of attributes: A single node may have one or more attributes. This array of attributes is implemented using vector to allow the ease of adding different number of attributes for different nodes. Each attribute are the instance of the class Attributes mentioned above.
- Getter and Setter function (more details on usge)

3.3 Edge Implementation

The Edge class for this program is called AnyEdge, their declaration and definitions can be found in AnyEdge.h and AnyEdge.cpp. The class AnyNode contains mainly:

- Edge ID: : The name that unique for every edge (ID information can be read to uniquely identify the edge). Implemented as string in order to allow combination of numbers and letters.
- Node IDs from - to: Indicate which two nodes are connected by the edge. Logically, there must be two nodes which each of them have one ID that contained in this attribute.
- Array of attributes: A single edge may have one or more attributes. This array of attributes is implemented using vector to allow the ease of adding different number of attributes for different edges. Each attribute are the instance of the class Attributes mentioned above.
- Getter and Setter function (more details on usge)

3.4 Topology implementation

A topology, in the network context, is a graph consist of nodes, edges and any information that are related to network. Topology class related files are Topology.h and Topology.cpp. The main elements of this class are:

- Array of nodes: A topology have one or more nodes. This array of node is implemented using vector to allow the ease of adding different number of nodes from different file. Each node in the array is the instance of class AnyNode mentioned above.
- Array of edges: A topology have one or more edges. This array of node is implemented using vector to allow the ease of adding different number of edges from different file. Each edge in the array is the instance of class AnyEdge mentioned above.
- Array of attributes: Some optional description of the graph that can be found in graphml file. Implemented using vector.
- Getter and Setter function (more details on usge)

3.5 File reading implementation

This program takes graphml or gml file type as input, the is the network topology that contains nodes, edges and their attributes that are mentioned above. For graphml file format, this is done by using tinyxml2 library (mentioned in requirement). For GML file format, this is done by processing string. As there are no available library for reading GML, the implementation for GML reading is somewhat buggy.

4 Usage

Since this program takes graphml or gml file type as input, return the network topology that contains nodes as output. Topology must be initiailzed before reading the graph

- Example:
  - To read graphml file: `Topology t = read_graphml_file("/home/user/sample.graphml");`

To get the total number of nodes/edges from the file, call the function `get_node_size()` or `get_edge_size()` respectively, for example:

```
cout<<"total nodes: "<<t.get_node_size()<<endl;
cout<<"total edges: "<<t.get_edge_size()<<endl;
```

Figure 1: Priting total number of nodes and edges onto the command line

To get information that are related to the individual attribute of a specific node or edge:

- For calling a specific node, call the function `get_node()[index]`.
- For calling a specific edge, call the function `get_edge()[index]`.

Example:

```
for(int i = 0; i < 3; ++i)
{
    cout<<"edge "<<i<<" detail: " <<endl;
    cout<<"source: "<<t.get_edge()[i].get_From()<<" target: "<<t.get_edge()[i].get_To()<<endl;
    for(uint i1 = 0; i1 < t.get_node()[0].get_attribute_list().size(); ++i1)
    {
        cout<<"attribute "<<i1<<" name: "<<t.get_edge()[i].get_Attribute()[i1].get_attribute_nam
        cout<<"attribute "<<i1<<" key: "<<t.get_edge()[i].get_Attribute()[i1].get_attribute_id()
        cout<<"attribute "<<i1<<" data: "<<t.get_edge()[i].get_Attribute()[i1].get_attribute_dat
        cout<<"attribute "<<i1<<" type: "<<t.get_edge()[i].get_Attribute()[i1].get_attribute_typ
    }
}
```

Figure 2: Priting attributes of the first three edges onto the command line

```
for(int i = 0; i <  3; ++i)
{
    cout<<"node "<<i<<" detail: " <<endl;
    cout<<"id: "<<t.get_node()[i].get_ID()<<endl;
    for(uint i1 = 0; i1 < t.get_node()[0].get_attribute_list().size(); ++i1)
    {
        cout<<"attribute "<<i1<<" name: "<<t.get_node()[i].get_attribute_list()[i1].get_attribu
        cout<<"attribute "<<i1<<" key: "<<t.get_node()[i].get_attribute_list()[i1].get_attribut
        cout<<"attribute "<<i1<<" data: "<<t.get_node()[i].get_attribute_list()[i1].get_attribu
        cout<<"attribute "<<i1<<" type: "<<t.get_node()[i].get_attribute_list()[i1].get_attribu
    }
}
```

Figure 3: Priting attributes of the first three nodes onto the command line