User guide for PGM-C++ software

Installation: The installation is a very simple two step process. Step 1 : Unzip the source files into the directory of choice Step 2 : After ensuring that gcc/7.1+ is available on the system, from the "src" directory, run "make". After a few seconds the "pgm" executable will be generated.

Directory structure: The software is packaged into three directories. The directories and their contents are described below

a. *src:* This contains all the source C++ files and the header files necessary. "pgm.cpp" provides the entry point to the program. "options.cpp" takes care of all the program options, "graph.cpp" is where all the work such as reading/writing files, building models, label augmentation gets done. "distributions.cpp" handles all the random distributions related functions. Finally "utils.cpp" provides some helper functions.
b. *scripts:* This directory contains the Python script that can be used to compare the basic network properties of the source and the target datasets. The usage of this script is completely optional. If utilized, this script has some dependencies that are outlined previously.
c. *data:* This directory holds three test cases and the options file needed to run each of them.

Usage: The executable pgm takes in only one argument which is the options file. To run the program just type ***"./pgm path_to_options_file***". For example *"./pgm ../data/options-role.txt"*

Options file: The options file holds a lot of information that is needed by the attributed graph generation tool. The information is specified as key-value pairs, one per line, separated by a space. The various options are described below:

a. Working_Dir – This specifies the working directory which holds the input dataset files and to which the output target dataset files are written. The path is specified as an absolute path or relative to the directory in which the executable is present and invoked.

b. Source_Graph_File – This option specifies the name of the source graph edgelist file.

c. Source_Attributes_File – This option specifies the name of the source graph node attributes file.

d. Target_Graph_File – This option specifies the name of the output target graph edgelist file.

e. Target_Attributes_File – This option specifies the name of the output target graph node attributes file.

f. Number_Target_Graph_Nodes – This option specifies the requested number of nodes in the target graph.

g. Number_Target_Graph_Edges – This option specifies the requested number of edges in the target graph.

h. Number_Attributes – This option specifies the number of node attributes in the source and target datasets.

i. Attribute_Cardinalities – This options lists the cardinality of each of the node attributes, one number per node attribute, separated by spaces. For example, if the number of node attributes (previous option) is 3 and each of the attributes are binary, the attribute cardinalities will be represented as “2 2 2”

j. Node_Probability_Threshold - This option defines the threshold below which the probability of node attribute vector is ignored (set to 0). Typically, of the order of 1e-6 or even lower.

k. Edge_Probability_Threshold - This option defines the threshold below which the probability of edge attribute vector (concatenated node attribute vectors) is ignored (set to 0). Typically, of the order of 1e-7 or even lower.

l. Augment_Label_Cardinality – This option specifies the cardinality of the augmented label. When the attributes cannot completely explain the connectivity, node labels need to be augmented with labels based on degree distribution (reflecting node popularity). Typical values are between 4 and 10. The value is upper-bounded to 10 within to code to avoid dimensionality explosion.

m. Random_Seed – This option specifies the random seed used to initialize the distributions for repeatability.

n. Augmentation_Type – Type of augmentation used for the label augmentation process. Choices are “Linear” and “Logarithmic”. Logarithmic gives better results for power-law type degree distributions.

o. OpenMP – This option specifies the number of cores. Currently multi-threading is supported only for the target edge generation. Values from 1 to 20 have been tested.

p. Attributes_Input_Type – This option specifies the attribute file format. If this option is absent, the code assumes that the first column of the attributes file is the node id. If this option is specifies with value “Node_Id_Implicit”, the code provides the node indexing implicitly starting from ‘0’. In the examples provided, “role” and “pa” do not have this option and hence the first column of those attribute files include the node id while “netflow” includes this option for implicit indexing.

Source dataset file formats: In this section we describe the formats of the graph and attributes files.

The graph file is a simple edge list that contains one pair of nodes per line denoting the source and destination nodes for the directed edge. The node ids are assumed to be integers in the present version of the software tool. Implicit support for undirected graphs is not available in the current version of the software tool but will be supported in future versions.

The attributes file can be in one of the two formats (see the description of the option Attributes_Input_Type). In both the formats, the attributes are specified (as integers) – one per line. In the first format the node id is explicitly specified in the first column. This is the default format. In the second format, the node is not specified, and code implicitly provides the node id (starting from 0). In both the cases each attribute value for a node is an integer. It can take on values from 0 to the cardinality of that attribute minus 1. The attributes are separated by space.

Multi threading support: Currently the code works on a single node and multiple cores. Multi-threading is supported only for the target edge generation. Future versions will support multi-threading support for other routines such as model build and target node generation. The scaling performance of the code has been tested up to 20 cores. For large target datasets, significant amount of main memory is utilized. For example, generation of a target graph of size 25 million nodes and 1 billion edges, with two binary attributes per node, required about 30GB of main memory.