1 Introduction

GT-ITM is a topology modeling package that produces graphs that *look* like wide-area internets. A brief description of graphs produced by GT-ITM is given below. More information about the GT-ITM graphs including the source code is available at `http://www.cc.gatech.edu/projects/gtitm`

1.1 Flat Random Graphs

The networking literature contains a variety of flat (i.e., non-hierarchical) random graphs used to model internetworks. All are variations on the standard random graph model that distributes vertices at random locations in a plane and then considers each pair of vertices; an edge is added between a pair of vertices with probability α. We call this the **Pure Random model** or simply the Random model. While it does not explicitly attempt to reflect the structure of real internetworks, it is attractive for its simplicity and is commonly used to study networking problems.

The other models also distribute the vertices randomly in the plane, however they alter the function used for the probability of an edge, in an attempt to better reflect real network structure. After the Pure Random Model, perhaps the most common random graph model is one proposed by Waxman [1], with the probability of an edge from u to v given by:

$$P(u, v) = \alpha e^{-d/(\beta L)}$$

where $0 < \alpha, \beta \leq 1$, d is the Euclidean distance from u to v and $L = \sqrt{2} \times scale$ is the maximum distance between any two nodes. An increase in α will increase the number of edges in the graph, while an increase in β will increase the ratio of long edges relative to shorter edges. We call this the **Waxman 1** model.

1.2 Hierarchical Models

Random models offer very little control over the structure of the resulting topologies. They do not capture the hierarchy that is present in real internetworks. We now describe a method of creating hierarchical topologies by connecting smaller components together according to a particular structure.

Transit-Stub

Our Transit-Stub model produces hierarchical graphs in a different way, by composing interconnected transit and stub domains. We first construct a connected random graph (using any one of the methods discussed earlier); each node in that graph represents an entire transit domain. Each node in that graph is then replaced by another connected random graph, representing the backbone topology of one transit domain. Next, for each node in each transit domain, we generate a number of connected random graphs representing the stub domains attached to that node. Finally, we add some number of additional edges between pairs of nodes, one from a transit domain and one from a stub, or one from each of two different stub domains.

2 Visualization of GT-ITM graphs

The aim of this project is to provide a *nice* front-end for GT-ITM graphs. Each vertex in a GT-ITM graph has x, y co-ordinates that are assigned as the vertex is created. This project should produce a (X11) based front end that can be used to visualize the graph, and make some changes to it. As a part of the project, a newer (more visually pleasing) vertex layout algorithm should be developed.

The front-end produced should allow the following interactions:

- Specification of which graph(s) to display
- Aggregation of nodes — a set of nodes should be able to be marked and aggregated as a macro-node. Possible ways to do this would include clicking at particular nodes with a mouse pointer or to outline a rectangle over a region of the graph which would then be abstracted as a macro-node.
- Addition of nodes and edges

The display should updated to reflect the changes.

Optionally, once macro-nodes have been identified, edges created/deleted, the visualizer should be able to produce the GRDEL(Section 7) program for the graph transformation. As the interactions map directly to GRDEL primitives, this step should be relatively straight forward.

3 Reading

The Stanford GraphBase is introduced and explained in `The Stanford GraphBase : a platform for combinatorial computing` by Donald Knuth, 1993 (`QA164.K6 1993`). Unless you are already familiar with The Stanford GraphBase, recommended sections to read are

- Chapter 4 — How to Read `CWEB` Programs
- Programs of the Stanford GraphBase
  - GB_BASIC
  - GB_IO

An alternate (more mortal-readable) graph format can also be used if that proves easier than the Stanford GraphBase. There are auxiliary routines available to convert SGB graphs to the alternate format and vice-versa.

4 Requisites

- Experience with a language/library for X11 graphics. Should be able to do elementary graphics programming, and write callback functions for the graph interactions.
- Elementary C programming — should be able to read/parse C function headers.
- Elementary background in graph theory will be useful. Exposure to algorithms to do planar embeddings may be useful (though most GT-ITM graphs are not planar) to produce nicer visualizations.

5 Implementation, and interface to GT-ITM

The GT-ITM graphs are Stanford GraphBase graphs that are created to have a set of properties. In general, the GT-ITM graphs already have x-y co-ordinates that are generated during graph creation. These co-ordinates can be used to visualize the graph to begin with. Though, it is desired that a nicer assignment of co-ordinates be developed as part of this project.

The programming in this project will include writing the front-end to visualize the graphs, and the subsequent interactions. This can be done in any reasonable language/library with graphical primitives and will mostly the choice of the student.

6 Evaluation

- Well-written, commented code for the visualization.
- A small report outlining your approach and anything interesting that you encountered. Include approaches taken to find a nice layout for the graphs and modifications you would like to see in GRDEL or GT-ITM or the visualization. Note that you will be primarily be evaluated on the work you do with GRDEL and GT-ITM and the front-end and not the report.

This will be a moderate project for one person with the pre-requisites.

7 The Little Language — GRDEL: GRaph DEfinition Language

The little language that will be used as the interface to GT-ITM graphs is partially specified below. The language will be an LISP like in "look-and-feel", as this is a convenient way to work with functions with variable number of arguments.

(**let** v s) Variable v is associated with node set s — i.e. node set s, which is may be a (null) set of nodes, macro-nodes, graphs, or variables are named v.

(**connect** a $\{b\}$ k) Vertices in node sets a and b are connected with a probability of k. If the second node set is null, then vertices within node-set a are connected (to each other) with the given probability.

(**new-vertex** v) A new vertex (referred to by variable v) is created.

(**edge-connect** a b k) Node sets a and b are k connected — i.e. there are at least k edge-disjoint paths between a and b.

(**vertex-connect** a b k) Node sets a and b are k vertex-connected — i.e. there are at least k vertex-disjoint paths between a and b.

References

[1] Bernard M. Waxman. Routing of multipoint connections. *IEEE Journal on Selected Areas in Communications*, 6(9):1617–1622, 1988.