VISUAL GRAPH ABSTRACTION

Anna Mitelman
Wellesley College

A graph with three vertices and three edges can be displayed on a computer screen and easily understood by a user. If that graph, however, has hundreds (or even dozens) of vertices and edges, then the image on the screen would be difficult to display and understand. In my Honors Thesis, I plan to experiment with graph transformations that hide some graph details and highlight others. I plan to implement a system that not only allows a user to perform transformations by hand but which also allows her to specify rules for performing such transformations automatically.

Graph transformations fall into two broad categories. In the first category are appearance transformations that change only the appearance or geometry of the graph. For example, if a graph is bipartite, then redraw it indicating the independent sets. Also, if there exists a path between two specified vertices, then highlight that path. While these transformations do not change the vertex or edge sets of a graph, they do emphasize significant features of a graph.

In the second category are structural transformations that change the vertex and edge sets of the graph. A subset of these transformations hide details of the graph and we can refer to them as abstractions. For example, all the vertices and edges that have a given relationship could be combined into a composite vertex. A complete subgraph, for example, could be redrawn as a single composite vertex indicating that all the vertices in the composite vertex are adjacent. Also each instance of a specified subgraph could be replaced with a composite vertex. The number of vertices and edges in the graph will then be reduced and the location of each occurrence of the subgraph will be emphasized.

One important feature of all the abstractions I plan to study is closure, implying that after a transformation has been applied to a graph the result is a new graph that can also be transformed. Hence, multiple abstractions can be performed on a given graph. For example, a composite vertex could be treated as a regular vertex and new abstractions could be performed upon it. These abstractions also hinge on the ability to find subgraphs.

Once I have determined a rich and expressive set of transformations, I will design a language of graph transformations. In defining this language my primitives will be structural transformations and a small number of appearance transformations. I will test my design by implementing a system, the Graph Abstracter, that will apply these transformations to a given graph.

There are several important considerations in the implementation. First, many problems, such as finding subgraphs, are NP-Complete and therefore intractable. If transformations are automated by user-specification rules then each of these transformations should have a priority ensuring that the most useful transformations were the most likely to be applied.

The ultimate goal of my project is to make graphs simpler to understand, simpler to visualize, and more representative of their meaning and structure. The Graph Abstracter can help to make many kinds of graphs more understandable: e.g., data structures, work-flow diagrams, chemical models, and transportation maps.