SIMULTANEOUS EDITING AND MULTILABELLING OF GRAPHS IN SYSTEM newGRAPH

Vladimir Brankov, Dragoš Cvetković, Slobodan Simić, Dragan Stevanović

In our research in spectral graph theory we often encounter the need for the simultaneous editing of two or more interdependent graphs (e.g. a graph and its line graph), together with multiple labellings of their vertices and edges. Occasionally, labellings are of such kind that it could be beneficial to permit the user to modify the labelling and test whether it still satisfies a given property. Here we develop a methodology for treating such situations, which is implemented in system newGRAPH.

1. INTRODUCTION

Let $G = (V, E)$ be a graph with $n$ vertices. The characteristic polynomial $\det(xI - A)$ of the adjacency matrix $A$ of $G$ is called the characteristic polynomial of $G$ and denoted by $P_G(x)$. The zeros of $P_G(x)$ are called the eigenvalues of $G$, and the spectrum of $G$ is the set of all eigenvalues of $G$. The eigenvalues of $G$ are real, since the adjacency matrix $A$ is symmetric. Spectral graph theory studies the connections which exist between a graph and its spectrally defined invariants. Some general references for spectral graph theory are [3], [4], [6].

Nowadays, there is an abundance of computer programs supporting, in various ways, research in graph theory. Majority of these systems were presented and surveyed in three workshops on “Computers and Discovery” held at DIMACS in 2001 [3], at HEC Montréal in 2004 and at the University of Ghent in 2006.

The oldest of these systems is an interactive programming package called GRAPH, an expert system for graph theory. It was developed at the University of
Simultaneous editing and multilabelling of graphs in system newGRAPH

of Belgrade, Faculty of Electrical Engineering, during the period 1980-1984 [1], [2], [5], GRAPH was designed to support research in graph theory, among other things, by helping to pose, verify and disprove conjectures. Graph theory results, obtained by several researchers in the period 1982-2001 with the support of the expert system GRAPH, have been reviewed in [8] and [9]. Most of these results belong to the theory of graph spectra.

Recently, we began the Java implementation of system newGRAPH (which can be downloaded from http://www.mi.sanu.ac.yu/newgraph/), which retains the spirit of GRAPH, but uses more sophisticated tools in order to provide more comfortable work environment. Above all, newGRAPH is modular and adaptive to the needs of a specific researcher. Using the concept of plug-ins and extensive library, the user is able to write his/her own graph invariants, graph generators and graph actions or import existing ones.

Concerning a spectral graph theorist, one often encounters the need for the simultaneous editing of two or more interdependent graphs (e.g. a graph and its line graph), together with multiple labellings of their vertices and edges. Occasionally, labellings are of such kind that it could be beneficial to permit the user to modify the labelling and test whether it still satisfies a given property. Here we develop a methodology for treating such situations, which is implemented in system newGRAPH.

2. INSPIRATION: LINE GRAPHS, PETALS, STAR PARTITIONS

Here we illustrate the types of visualization problems that we encounter when we want to support research in spectral graph theory.

An important part of research in spectral graph theory consists of studying generalized line graphs and star partitions.

Let $K_n$ denote the complete graph on $n$ vertices. The cocktail-party graph $CP(n)$ is the unique regular graph with $2n$ vertices of degree $2n - 2$; it is obtained from $K_{2n}$ by deleting $n$ mutually non-adjacent edges. The line graph $L(G)$ of a graph $G$ is defined as follows: the vertices of $L(G)$ are the edges of $G$ and two vertices of $L(G)$ are adjacent whenever the corresponding edges of $G$ have a vertex of $G$ in common.

An elementary observation that line graphs have the least eigenvalue greater than or equal to $-2$ raised interest in the study of graphs with such a remarkable property with a natural problem to give their full characterization. It appeared that line graphs share this property with generalized line graphs and with some exceptional graphs. Much information on these problems can be found in the book [7].

A generalized line graph $L(G; a_1, \ldots, a_n)$ is defined for graphs $G$ with vertex set $\{1, \ldots, n\}$ and non-negative integers $a_1, \ldots, a_n$ by taking the graphs $L(G)$ and $CP(a_i)$ ($i = 1, \ldots, n$) and adding extra edges: a vertex $e$ in $L(G)$ is joined to all vertices in $CP(a_i)$ if $i$ is an end-vertex of $e$ as an edge of $G$. We include as special
cases an ordinary line graph \((a_1 = a_2 = \cdots = a_n = 0)\) and the cocktail-party graph \(CP(n)\) \((n = 1\) and \(a_1 = n)\).

In newGRAPH, in order to generate a generalized line graph from the existing graph \(G\) the user is required first to provide a vertex labelling. Moreover, changes in \(G\) should automatically reflect in its line graph. This leads us to the first two problems.

**Problem A.** Enable the user to define his/her own labelling of vertices and edges, and its further use in defining new graph objects.

**Problem B.** Provide the visualization of a graph \(f(G)\) depending on a graph \(G\), which will observe the changes in \(G\) and update automatically. More generally, enable the same feature for a graph \(f(G_1, G_2, \ldots, G_m)\) depending on graphs \(G_1, G_2, \ldots, G_m, m \geq 1\).

Further, we can introduce the concept of the root graph of a generalized line graph. Let a petal be a pendant double edge. A blossom \(B_n\) consists of \(n\) \((n \geq 0)\) petals attached at a single vertex. (An empty blossom \(B_0\) has no petals and is reduced to the trivial graph \(K_1\).) A graph in which to each vertex a blossom (possibly empty) is attached is called a graph with blossoms or a \(B\)–graph. A graph \(H\) is a generalized line graph if \(H = L(G)\) is the line graph of a \(B\)–graph \(G\) called the root graph of \(H\). The definition of \(L(G)\) remains as given above. We have \(L(B_n) = CP(n)\). If one changes the generalized line graph arbitrarily, then the root graph need not always exist. Thus, the concept of the root graph suggest the third problem.

**Problem C.** Provide the interactive visualization of \(H = f(G)\), such that the changes in \(H\) cause the changes in \(G\) in order to keep the equality \(H = f(G)\).

Another important concept in spectral graph theory are star partitions. Let \(G\) has distinct eigenvalues \(\mu_1, \mu_2, \ldots, \mu_m\), where \(\mu_i\) has multiplicity \(k_i, i = 1, 2, \ldots, m\). If we delete a single vertex \(v\) from \(G\) then the multiplicity of \(\mu_i\) is at least \(k_i - 1. A partition \(V_1 \cup V_2 \cup \cdots \cup V_m\) of the vertex set of \(G\) is a star partition of \(G\) if and only if, for each \(i = 1, 2, \ldots, m, \mu_i\) is not an eigenvalue of \(G - V_i\). More information on star partitions can be found in [6].

A star partition provides a natural way (in general non-unique) of ascribing the eigenvalues of a graph to its vertices, where each of the vertices of \(V_i\) is labelled by \(\mu_i\). Obtaining one star partition is possible with newGRAPH and that is an example of a read-only, graph-defined labelling. The user may want to change the star partition found by program and test whether a new partition is still a star partition, which leads to the last problem in this section.

**Problem D.** Enable the user to copy a graph-defined labelling into another labelling which may be freely modified.
3. THE RESOLVING FRAMEWORK

3.1. Labellings

So far, newGRAPH had three main plug-in ingredients:

- **generators**, which use the supplied parameters to create a particular graph from a specified family of graphs;
- **invariants**, of which selected ones are shown in a table below the graph drawing and are automatically recalculated whenever the graph changes;
- **actions**, which perform various modifications of the whole or part of a graph.

In order to solve problems A and D from the previous section, we introduce another plug-in type: **labellings**. Labelling may be either a vertex or an edge labelling. Each graph $G$ in newGRAPH has an associated natural labelling of vertices with numbers $1, 2, \ldots, |V(G)|$, which is used internally.

The user creates a new labelling by issuing the **Create** command from the Labellings menu. In the **Create new labelling** dialog that appears, the user chooses whether he/she wants an empty labelling, in which case he/she has to select the labelling data type (integers, reals or strings), or whether he/she wants to copy the values of a selected invariant into a new labelling. (The natural labelling of vertices is, a bit arbitrarily, considered an invariant. Perhaps a better way would be to call it a graph property.) Additionally, the user may make a copy of the existing labelling with a **Copy labelling** command from Labellings menu.

Labellings are displayed in a table below the graph drawing, together with the selected invariants. Initially, the labelling is not visible in the drawing. In order to achieve this, the user has to check the **Visualize** box next to the name of the labelling in the table. In the drawing of the graph, each vertex is represented by a rectangle containing all labellings to be visualized, one in each row corresponding to the order of labellings in the table, and each edge has a similar rectangle placed in its middle (see Fig. 1). Further help in visualizing labellings is color-coding: the user may modify the font color for each labelling by issuing the **Color** command from the Labellings menu.

Since most of the labellings used in spectral graph theory use real numbers, not all of their digits may be packed easily into a drawing. In order to keep the (vertex) rectangle small, at most five symbols from each labelling are shown, and the rest of the symbols may be obtained either from a fast tooltip, which appears as a full-sized vertex rectangle after approximately 0.25 second, or by issuing a command **List labels** from the Labellings menu, which list all labels in newGRAPH console window. In case the labels are long enough, the user may use the console to compare the full values of labels for different vertices by issuing **List labels** command in turn.
3.2. Testing Methods

Although testing methods would be rather rarely used for most of the graph properties, a researcher studying star partitions will often enough want to change the labelling given by a star partition (move the vertices from one to another part) and test whether a new labelling is still a valid star partition. In order to enable this behaviour, one needs to be able to implement a method which takes a labelling as a parameter and checks whether it satisfies a particular property. In newGRAPH this is similar to writing a new invariant (for more information, check out http://www.mi.sanu.ac.yu/newgraph/). A testing method is implemented in a class having the following, self-explanatory structure:

```java
import scg.ac.mi.pmf.newgraph.<All necessary classes>;

public class <TestingMethod>
    implements <ReturnTypeClass>, Parameterizable
{
    public String getName() {
        return "<Short name>";
    }

    public String getDescription() {
        return "<A longer description of the testing method.>";
    }

    public <returnType> calculate(GraphData data) {
        // Body of the testing method.
    }
}
```

Figure 1: Visualization of different graph labellings.
public String checkParameters() {
    // returns null whenever
    // the supplied parameters have sensible values,
    // and an error message otherwise.
}

private <LabellingType> labelling;

public int getLabelling() {
    return labelling;
}

public void setLabelling(<LabellingType> labelling) {
    this.labelling = labelling;
}

Here, the labelling is given as a parameter that is determined at the runtime. Each parameter (and there may be more than one) has the standard JavaBean methods for getting and setting the parameter value:

<type> get<parameter_name>

and

void set<parameter_name>(<type>)

This way, the user may implement even stronger testing methods, which need not only test one labelling, but which may take into account some other invariants or labellings of a graph and return other results than simple yes/no.

3.3. Interdependent Graphs

Here we describe our solutions to problems B and C from the previous section. The user can create a new graph that depends on one or more of the existing graphs by issuing a command **Dependent graph...** from the **New** submenu of the **File** menu. The **New dependent graph...** dialog that appears consists of a list of operators defining the dependency, e.g., a complement, a blossom graph, a generalized line graph, a product of two graphs. In Figures 2 and 3 we provide screenshots of the system where the dependent graph is the sum of two graphs and the generalized line graph of a graph, respectively.
Figure 2: The dependent graph is the sum of two graphs.

Since newGRAPH is written so that it can be easily extended by its users, each operator is defined in a class that has the following general structure:

```java
import scg.ac.ni.pmf.newgraph.<All necessary classes>;

public class <Operator>
    implements GraphGenerator, Parametrizable
{
    public String getName() {
        return "<Short name>";
    }

    public String getDescription() {
        return "<A longer description of the operator.>";
    }

    public void newGraph(Graph graph) {
        // Body of the dependent graph generator.
    }
}
Simultaneous editing and multilabelling of graphs in system newGRAPH

Figure 3: The dependent graph is the generalized line graph of a graph.

```java
public String checkParameters() {
    // returns null whenever
    // the supplied parameters have sensible values,
    // and an error message otherwise.
}
```

// Parameter section

After the user selects an operator from the **New dependent graph...** dialog, another dialog appears which provides a field for each parameter of the operator. (Supplying graph names is easiest by using drag & drop technique.) The graph is then created by calling the `newGraph` method in the above class.

The parameters that the graph depends on are observed by newGRAPH and whenever one of them is changed, the `newGraph` method is called again.

This way we can also deal with chains of dependencies—for example, the user may create a graph $G$, its line graph $L(G)$ and its line graph $L(L(G))$. Whenever
changes, both \( L(G) \) and \( L(L(G)) \) get updated.

At certain time, the user may want to break the dependence between graphs and continue to edit freely the resulting graph \( G \). This can be achieved by issuing the \texttt{Get independent} command from the Edit menu. The graphs that \( G \) depended on no longer have an influence on \( G \); however, the graphs that depended on \( G \) are still updated when \( G \) changes.

In cases when the graph depends only on one root graph, it may be possible to obtain a root graph from the dependent graph (this is always possible if the dependent graph is a complement, and sometimes possible if the dependent graph is a line graph). If the class \texttt{<Operator>} defines method

```java
public Graph invert() {
    // Body of the root graph generator.
}
```

the system newGRAPH tries to work also in reverse: a change to the dependent graph results in a call to \texttt{invert}, which returns its root graph. If such a graph does not exist, the root graph is set to \texttt{null}, and its editing is not possible until either the user selects the \texttt{Undo} command from the Edit menu, or the user modifies the dependent graph such that \texttt{invert} returns non-null value.

4. CONCLUSION

Features of the system newGRAPH explained here are inspired by the specific needs of spectral graph theorists. However, we proposed a general framework for their implementation, which enables other researchers in graph theory to visualize and study other labellings, such as, for example, graph colorings. The general framework also enables easier modifications and extensions in the future.

REFERENCES

Simultaneous editing and multilabelling of graphs in system newGRAPH


