A Graph Theory Package for Maple

Jeffrey B. Farr[*], Mahdad Khatirinejad[*],
Sara Khodadad[*], Michael Monagan[*]
Department of Mathematics
Simon Fraser University
Burnaby, BC
Canada V5A 1S6

e-mail:
{jfarr,mahdad,skhodada,mmonagan}@cecm.sfu.ca

Abstract

We present a new graph theory package for Maple. The package is presently intended for teaching and research usage and is designed to handle graphs of up to 1000 vertices. Most of the standard operations for graphs are available in this package, and we describe some of them in this paper. A full list of the currently supported commands is given in the final section. The package also includes a drawing component.

1 Introduction

We describe in this paper a new graph theory package for Maple. The package we are developing is presently intended for teaching and research use and is designed to handle graphs of up to 1000 vertices. The current package in Maple for solving problems in graph theory is the networks package. This package is over ten years old and was designed primarily with applications of networks in mind. Its data structure is too heavy and cumbersome to

[*]Supported by the MITACS NCE of Canada and NSERC of Canada.

treat some elementary graph theory problems. One design criterion for the new GraphTheory package is that it must posses a simple, yet flexible, data structure designed primarily for solving problems related to graphs rather than networks. Most of the standard operations for graphs, including a planarity test, maximum clique, hamiltonicity, Tutte polynomial, and chromatic number, are available in this package. The package also includes a drawing component.

2 The data structure

The data structure of a (di)graph has six arguments and is as follows:

$$\mathit{GRAPHLN}(D, W, V, A, T, EW),$$

where D and W are of type symbol, V is of type list, A is of type Array, T is of type table, and EW is of type Matrix. Two symbols, namely *directed* or *undirected*, are possible for D, and two symbols, *weighted* or *unweighted*, can be used for W. The list V stores the labels of the vertices of the graph. The array A contains the set of neighbors for each vertex. Each element of A is of type set(posint). The table T stores some of the graph properties which are expensive to find but are cheap to store. The sparse matrix EW stores the edge weights of the graph if the graph is weighted; otherwise, EW is not a matrix and is set to 0. The following example shows the data structure of the unweighted graph $G = (V, E)$ where $V = \{a, b, c\}$ and $E = \{\{a, b\}, \{b, c\}\}$.

```
> with(GraphTheory):
> G := Graph([a,b,c],{{a,b},{b,c}});
```

$$\mathit{GRAPHLN}(\mathit{undirected}, \mathit{unweighted}, [a, b, c], [\{2\}, \{1, 3\}, \{2\}], \mathit{table}([]), 0)$$

Notice that the internal representation for the graph uses integers $1, 2, 3$ rather than the user labels a, b, c. This is for efficiency and canonicality. Once a graph G is created, the user cannot remove or insert new vertices without making a new copy of G. However, the user can remove or insert new edges.

```
> DeleteEdge(G,{a,b}):  G;
```

$$\mathit{GRAPHLN}(\mathit{undirected}, \mathit{unweighted}, [a, b, c], [\{\}, \{3\}, \{2\}], \mathit{table}([]), 0)$$

Suppose we insert/delete an edge $\{i, j\}$ to/from a graph G. Because we use an array of sets of integers to represent G, the cost of edge insertion/deletion is $O(\max\{deg_G(i), deg_G(j)\})$.

3 Graph constructor command

In order to construct a (di)graph, one may use the constructor commands `Graph()` or `Digraph()`. Various combinations of the parameters V, E, A, D and/or W may be used to construct a graph, where V is the set or list of vertices or the number of vertices, E is the set of edges, A is the adjacency matrix (possibly containing edge weights), D is a symbol of the form directed/undirected or directed=true/false, and W is a symbol of the form weighted/unweighted or weighted=true/false. The input parameters may appear in any order, however they should be compatible. The following examples show some of the different ways of constructing a (di)graph.

```
> V := [a,b,c,d]:
> E1 := {{a,b},{a,c},{b,d}}:
> E2 := {[[a,b],1.0],[[a,c],2.3],[[b,d],3.1], [[b,a],4]}:
> A := Matrix([[0, 1, 1, 0], [1, 0, 0, 1], [1, 0, 0, 0], [0, 1,
0, 0]]):
> Edges(Graph(V));
```

$$\{\}$$

```
> AdjacencyMatrix(Graph(E1)), AdjacencyMatrix(Graph(V, E2));
```

$$\begin{bmatrix} 0 & 1 & 1 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}, \begin{bmatrix} 0 & 1.0 & 2.3 & 0 \\ 4 & 0 & 0 & 3.1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

```
> Edges(Graph(A));
```

$$\{\{1, 2\}, \{1, 3\}, \{2, 4\}\}$$

```
> Edges(Graph(directed, A, weighted));
```

$$\{[[1, 2], 1], [[1, 3], 1], [[2, 1], 1], [[2, 4], 1], [[3, 1], 1], [[4, 2], 1]\}$$

We emphasize that E must of be of type set. Any input of type list will be interpreted as the labels of vertices. This restriction enables the user to create graphs with vertex labels of type set. Also, notice that inputs of the form `Graph(E,A)` and `Graph(V,E,A)`, where V is of type set, are not allowed. The problem is that this type of input cannot be interpreted unless an expensive isomorphism test is run.

4 Commands and features

4.1 Maximum clique

The problem of finding a maximum clique (or a maximum independent set) of a graph is NP-hard. We have implemented a backtracking (branch and bound) algorithm to solve this problem; the bounding function that we have used is the greedy coloring of a graph (see [4]).

```
> G := RandomGraph(15,.4):
> MaximumClique(G);
```

$$[2, 4, 10, 15]$$

```
> MaximumIndependentSet(G);
```

$$[1, 2, 6, 9, 14]$$

4.2 Planarity test

To test whether a graph on n vertices is planar or not, we have implemented an algorithm of Demoucroun, et al. (see [3]), which has running time $O(n^2)$. There are some algorithms such as the Hopcroft-Tarjan algorithm which are linear time. However, the algorithm we have used is much simpler and seems to be more efficient for graphs with less than 1000 vertices. If a graph is planar, we output the plane embedding of the graph as a set of faces. This information is useful for drawing the planar graph.

```
> IsPlanar(PetersenGraph());
```

$$\text{false}, \{\}$$

```
> IsPlanar(DodecahedronGraph());
```

$$\begin{aligned}\text{true}, \{&[15,6,11,16,20],[9,4,5,10,14],[19,14,10,15,20],\\&[20,16,17,18,19],[7,2,3,8,12],[5,1,6,15,10],\\&[3,2,1,5,4],[17,12,8,13,18],[13,9,14,19,18],\\&[1,2,7,11,6],[11,7,12,17,16],[3,4,9,13,8]\}\end{aligned}$$

4.3 Traveling salesman problem

Given a collection of cities and the cost of travel between each pair of them, the traveling salesman problem, is to find the cheapest way of visiting all of the cities and returning to your starting point. The `TravelingSalesman` command will return two objects. The first argument is the optimal value for the traveling salesman problem, and the second is a Hamiltonian cycle that achieves the optimal value. The strategy is a branch and bound algorithm using the so-called Reduce bound (see [4]).

```
> G := CompleteGraph(10):
> M;
```

$$\begin{bmatrix} 0 & 68 & 37 & 95 & 57 & 30 & 1 & 25 & 71 & 84 \\ 68 & 0 & 9 & 26 & 90 & 26 & 97 & 29 & 47 & 78 \\ 37 & 9 & 0 & 84 & 59 & 11 & 67 & 61 & 75 & 35 \\ 95 & 26 & 84 & 0 & 1 & 99 & 55 & 63 & 19 & 8 \\ 57 & 90 & 59 & 1 & 0 & 61 & 66 & 18 & 7 & 48 \\ 30 & 26 & 11 & 99 & 61 & 0 & 93 & 10 & 14 & 54 \\ 1 & 97 & 67 & 55 & 66 & 93 & 0 & 47 & 20 & 95 \\ 25 & 29 & 61 & 63 & 18 & 10 & 47 & 0 & 28 & 52 \\ 71 & 47 & 75 & 19 & 7 & 14 & 20 & 28 & 0 & 92 \\ 84 & 78 & 35 & 8 & 48 & 54 & 95 & 52 & 92 & 0 \end{bmatrix}$$

```
> TravelingSalesman(G,M);
```

142, [1, 7, 9, 5, 4, 10, 3, 2, 6, 8]

4.4 Maximum flow

The basic problem of finding a maximal flow in a network occurs not only in transportation and communication networks, but also in currency arbitrage, image enhancement, machine scheduling and many other applications. To find the maximum flow of a network on n vertices and m edges, we have implemented the so called preflow-push (push-relabel) algorithm. This algorithm runs in $O(n^2 m)$ time, an improvement over the $O(nm^2)$ augmenting path algorithms, *e.g.* Edmonds-Karp, which are often used. This algorithm is also used to find the vertex connectivity and edge connectivity of a graph.

```
> N := Digraph([s,v1,v2,v3,v4,t],
{[[s,v1],16],[[s,v2],13],[[v1,v2],10],[[v2,v1],4],[[v1,v3],12],
[[v3,v2],9],[[v2,v4],14],[[v4,v3],7],[[v3,t],20],[[v4,t],4]}):
> MaxFlow(N,s,t);
```

23, table([(1,3)=10, (2,3)=1, (5,6)=4, (3,5)=11, (5,4)=7, (4,6)=19, (2,4)=12, (1,2)=13])

4.5 Coloring

A proper coloring of a graph G is a labelling of $V(G)$ so that adjacent vertices do not have the same label. A graph is k-colorable if it has a proper coloring with k labels, and the chromatic number $\chi(G)$ is the minimum k such that V is k-colorable. Computing $\chi(G)$ is an NP-hard problem. To compute the chromatic number, the `GraphTheory` package first computes a fast bound for the region in which $\chi(G)$ must lie, then uses a successive backtrack approach to find the exact value. A greedy coloring is used for the upperbound, and either the maximum clique size $\omega(G)$ or $|V|/\alpha(G)$, where $\alpha(G)$ is the independence number, whichever is easier to compute for a given graph, is used for the lowerbound. The coloring commands in `GraphTheory` return both the number of colors used and the (zero-based) coloring itself.

```
> G := CycleGraph(5):  GraphTheory:-GreedyColor(G));
```

3, [0, 1, 0, 1, 2]

```
> CliqueNumber(G), NumberOfVertices(G) / IndependenceNumber(G);
```

2, 5/2

```
> H := ShrikhandeGraph():GraphTheory:-GreedyColor(H));
```

$$4,\ [0, 0, 1, 2, 0, 0, 2, 1, 1, 2, 3, 3, 2, 1, 3, 3]$$

```
> CliqueNumber(H), IndependenceNumber(H);
```

$$3,\ 4$$

So, in both the cases above the greedy coloring actually finds an optimal coloring. A trivial way to augment a graph so that the chromatic number increases is to add cliques as subgraphs of the graph; however, the lower-bound for the chromatic number of the graph would increase as well. The Mycielski construction (see, for example, [6]) gives a way to build up triangle-free graphs with increasing chromatic number. Figure 1 shows the Mycielski construction applied to the five-cycle.

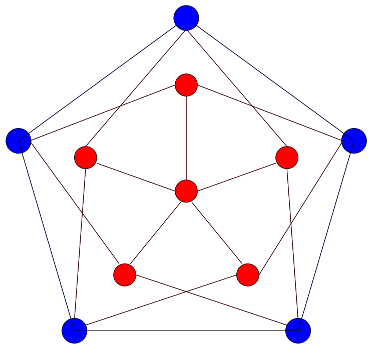


Figure 1: Mycielski construction applied to C_5

```
> M := Mycielski(G):
> ChromaticNumber(M);
```

$$4,\ [0, 1, 0, 1, 2, 0, 1, 0, 1, 2, 3]$$

```
> M2 := Mycielski(M):
> ChromaticNumber(M2);
```

$$5,\ [0, 1, 0, 1, 2, 0, 1, 0, 1, 2, 3, 0, 1, 0, 1, 2, 0, 1, 0, 1, 2, 3, 4]$$

4.6 Polynomials

There are several polynomials associated with graphs. The chromatic polynomial, when evaluated at the variable p, counts the number of proper colorings of a graph using p colors. Hence, the smallest positive integer that is not a root of the polynomial is the chromatic number. In other words computing the chromatic polynomial involves finding the chromatic number as a subproblem. As an illustration, it is evident that $\chi(G) = 3$ in the example below.

```
> G := RandomGraph(10,0.3):
> ChromaticPolynomial(G,p);
```

$$p^3(p-1)(p^2-4p+5)(p-2)^4$$

The chromatic polynomial is one of several polynomials that are special cases of the bivariate Tutte polynomial of $G = (V, E)$ which is defined by

$$T(G,x,y) = \sum_{E' \subseteq E} (x-1)^{\rho(E)-\rho(E')}(y-1)^{|E'|-\rho(E')},$$

where $\rho(E')$ is the number of vertices minus the number of components of the graph $G[E']$. All these polynomials follow the general recursion

$$\texttt{TuttePolynomial}(\texttt{G},\texttt{x},\texttt{y}) = \texttt{TuttePolynomial}(\texttt{DeleteEdge}(\texttt{G},\texttt{e}),\texttt{x},\texttt{y})$$

$$+\texttt{TuttePolynomial}(\texttt{Contract}(\texttt{G},\texttt{e}),\texttt{x},\texttt{y}),$$

and we use this recursion to compute them. Other polynomials in this family that are included in `GraphTheory` are: `AcyclicPolynomial`, `FlowPolynomial`, `RankPolynomial` and `SpanningPolynomial`. In certain cases we are able to speed up the algorithms by taking advantage of some graph isomorphisms which are not too expensive to check; `SpanningPolynomial` is one such example.

```
> TuttePolynomial( CompleteGraph(4) );
```

$$y^3 + 3y^2 + 4yx + 2y + 2x + 3x^2 + x^3$$

```
> G := RandomGraph(10,0.8):  f := SpanningPolynomial(G,p);
```

$$f := 132240p^{37} - 4164696p^{36} + 63358344p^{35} - \cdots - 22436993424p^{12} + 3308086227p^{11} - 317966985p^{10}$$

```
> eval(f,p=0.5)
```

$$0.9116$$

Other graph theory polynomials of interest included in `GraphTheory` are the well-known characteristic polynomial and the more recently studied graph polynomial. The characteristic polynomial of a graph is simply the characteristic polynomial of the adjacency matrix of the graph, and the following are only a few of the results relating roots of this polynomial (still called eigenvalues in this setting) to other graph parameters.

Theorem 1.

(i) `Diameter(G)` $<$ *the number of distinct eigenvalues of* G *(for connected* G*);*

(ii) `MinimumDegree(G)` $\leq$ *the largest eigenvalue of* $G \leq$ `MaximumDegree(G)`*;*

(iii) [7] $\chi(G) - 1 \leq$ *the largest eigenvalue of* G*.*

Occasionally, (iii) in the above theorem gives a better upperbound for $\chi(G)$ than a greedy coloring does (as in the example below), but usually greedy coloring is a better strategy for bounding.

```
> G := RandomGraph(15,0.15):
> f := CharacteristicPolynomial(G):
> Ev :=  solve( f=0 ) :
> EvList := [ seq( evalf(simplify(Ev[j])), j=1..nops(Ev) ) ]:
> max(op(EvList))+1;
```

$$3.647463440$$

```
> GraphTheory:-GreedyColor(G);
```

$$4, [0, 0, 0, 1, 0, 0, 1, 1, 0, 2, 1, 2, 0, 2, 3]$$

The so-called graph polynomial of $G = (V, E)$ is the n-variate polynomial defined by $\prod_{\{i,j\}\in E, i<j}(x_i - x_j)$. A result due to Alon and Tarsi [1, 2] says that G is not k-colorable if and only if `GraphPolynomial(G)` $\in \langle x_1^k - 1, x_2^k - 1, \ldots, x_n^k - 1\rangle$ over the complex numbers. Using this result and using the new `PolynomialIdeals` package in Maple ([5]), we can determine that the

chromatic number of the following graph is three without actually finding a 3-coloring.

```
> E := {{2, 6}, {3, 5}, {4, 5}, {3, 4}, {4, 6}, {5, 6}, {1, 4}}
> G := Graph(E):
> f := GraphPolynomial(G);
```

$$f := (x_2 - x_6)(x_3 - x_5)(x_4 - x_5)(x_3 - x_4)(x_4 - x_6)(x_5 - x_6)(x_1 - x_4)$$

```
> GraphTheory:-GreedyColor(G);
```

$$4,\ [0,\ 0,\ 0,\ 1,\ 2,\ 3]$$

```
> with(PolynomialIdeals):
> n := nops(Vertices(G)):
> for k from 2 to 3 do
>  xvars := seq( x[j], j=1..n):
>  printf("%d-colorability:\n", k);
>    J := < seq( xvars[j]^k -1, j=1..n) >;
>  NormalForm(f,J,plex(xvars));
> end do;
2-colorability:
```

$$J := < x_6^2 - 1, x_5^2 - 1, x_4^2 - 1, x_3^2 - 1, x_1^2 - 1, x_2^2 - 1 >$$

$$0$$

```
3-colorability:
```

$$J := < x_1^3 - 1, x_6^3 - 1, x_2^3 - 1, x_4^3 - 1, x_5^3 - 1, x_3^3 - 1 >$$

$$x_5 x_6^2 x_1 + \cdots + x_6^2 x_4^2$$

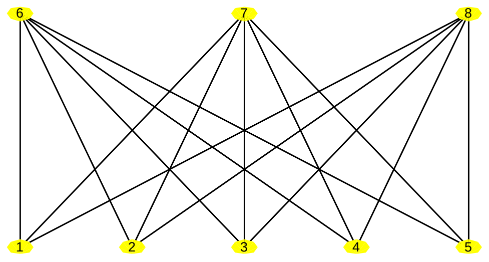
5 Drawing the graph

The graph visualization component of `GraphTheory` consists of two modules, namely *layout* and *display* modules. The layout module gets a `Graph` data structure and determines how to arrange the vertices and edges. It also determines in what color each vertex or edge should be displayed. The display module gets the output of the Layout module and displays the graph accordingly. Currently our display module uses MAPLE's `plot` command to display a graph.

The advantage of this modular structure is that we can replace one module by another existing module or implement a different module for a special purpose. For example GRAPHVIZ is a software package capable of displaying different types of graphs and diagrams in various ways, e.g. PostScript, GIF/JPEG/PNG, MetaPost, and simple text format. We could use GRAPHVIZ as a layout module by getting the text output of it—which contains information on the layout of the input—and write an adapter that converts this text output to what our display module expects it to be. Another example would be implementing a display module which gets the layout output and generates a metapost file to be inserted into a LaTeX document.

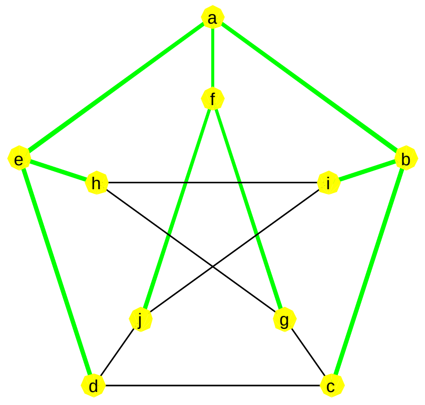
`GraphTheory` package includes a command named `DrawGraph` for displaying graphs. `DrawGraph` distinguishes three types of graphs: *trees*, *bipartite* graphs and *special* graphs such as the Petersen, Dodecahedron, Icosahedron, Octahedron and the so-called Clebsch graph. If the graph to be drawn does not fall into one of these categories, then the vertices are equally spaced in a circle. The user can force `DrawGraph` to display the graph using a specific style by writing `DrawGraph(G, style='s')` where `s` has one of these three values: `circle, tree, bipartite`. `DrawGraph` also detects all components of a disconnected graph and displays each component separately.

```
> G := CompleteGraph(5, 3):
> DrawGraph(G);
```



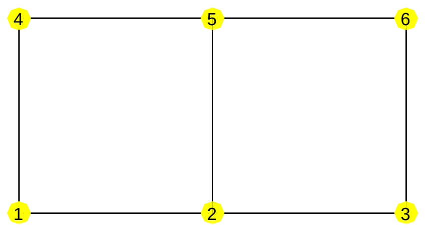
There are some cases when the user may want to mark an edge or a set of edges of a graph. If S is a subgraph of G, then `Highlight(G, S)` achieves this result. The color used for highlighting may be specified by writing `Highlight(G, S, COLOR(RGB, r, g, b)`.

```
> G := PetersenGraph():
> S := SpanningTree(G):
> Highlight(G, S):
> DrawGraph(G);
```



One other useful command when drawing graphs is `SetVertexPos` which gets three arguments: a graph, a vertex label and the position for that vertex. `SetVertexPos` can be used when the user wants to specify how to arrange the vertices of a graph. The following example displays the input graph as a grid.

```
> G := Graph({{1,2},{1,4},{4,5},{2,5},{2,3},{3,6},{5,6}}):
> p := [[0,0], [0.5,0], [1,0], [0,0.5], [0.5,0.5], [1,0.5]]:
> for i to nops(p) do
>  SetVertexPos(G, i, p[i]):
> end do:
> DrawGraph(G);
```



A command named `SetOption` will be available to set the output of commands generating a `Graph` data structure. For example, one of the options would be to set the output to be the plotted graph by writing `SetOption(GraphOutput='Plot')`.

In the future we will add the possibility of drawing directed or weighted graphs. Also, we are going to support more *special* graphs and some other general graphs such as `Grid` graphs and `BiConnected` graphs. Also, one other feature that can be added in future is supporting 3-dimensional plots of graphs.

6 List of commands of the GraphTheory package

AcyclicPolynomial	AddArc	AddEdge
AddVertex	AdjacencyMatrix	AllPairs
Arrivals	BiConnectedComponents	CharacteristicPolynomial
ChromaticNumber	ChromaticNumberBound	ChromaticPolynomial
ClassicDrawGraph	ClebschGraph	CliqueNumber
CompleteBinaryTree	CompleteGraph	CompleteBinaryTree
Connect	ConnectedComponents	Contract
CopyGraph	CycleBasis	CycleGraph
Deck	Degree	DegreeSequence
DeleteArc	DeleteEdge	DeleteVertex
Departures	Diameter	Digraph
Distance	DodecahedronGraph	DrawGraph
EdgeConnectivity	Edges	FlowPolynomial
FundamentalCycle	Girth	Graph
GraphComplement	GraphDifference	GraphIntersection
GraphJoin	GraphPower	GraphRank
GraphSum	GraphUnion	GridGraph
Head	Highlight	HyperCubeGraph
IcosahedronGraph	IncidenceMatrix	IncidentEdges
InDegree	IndependenceNumber	InducedSubgraph
IsAcyclic	IsBipartite	IsClique
IsConnected	IsCutSet	IsDirected
IsEulerian	IsGraphicSequence	IsHamiltonian
IsPlanar	IsRegular	IsTree

IsWeighted	LineGraph	MakeDirected
MakeWeighted	MaxFlow	MaximumClique
MaximumDegree	MaximumIndependentSet	MinimumDegree
MinimumSpanningTree	MycielskiGraph	Neighbors
NumberOfEdges	NumberOfTrees	NumberOfVertices
OctahedronGraph	OutDegree	PathGraph
PayleyGraph	PermuteVertices	PetersenGraph
RankPolynomial	RandomDigraph	RandomGraph
RandomTournament	RelabelVertices	SeidelSpectrum
SequenceGraph	SetOption	SetVertexPos
ShortestPath	ShrikhandeGraph	SpanningPolynomial
SpanningTree	Spectrum	StandardGraph
Subdivide	Subgraph	Switch
Tail	TetrahedronGraph	TopologicSort
TravelingSalesman	TuttePolynomial	UnderlyingGraph
VertexConnectivity	Vertices	WheelGraph

References

[1] N. Alon and M. Tarsi, Colorings and orientations of graphs, *Combinatorica* **12** (1992), no. 2, 125–134.

[2] Noga Alon and Michael Tarsi, A note on graph colorings and graph polynomials, *J. Combin. Theory Ser. B* **70** (1997), no. 1, 197–201.

[3] Alan Gibbons, *Algorithmic Graph Theory.* Cambridge University Press, 1985.

[4] D. L. Kreher, D. R. Stinson, *Combinatorial algorithms: generation, enumeration, and search.* CRC Press, 1999.

[5] M. Monagan and R. Pearce, The PolynomialIdeals Maple package, *Proceedings of the 2004 Maple Summer Workshop.*

[6] Douglas B. West, *Introduction to graph theory.* Prentice Hall, Inc., Upper Saddle River, NJ, 1996.

[7] H. S. Wilf, The eigenvalues of a graph and its chromatic number, *J. London Math. Soc.* **42** (1967) 330–332.