University of Ljubljana
Institute of Mathematics, Physics and Mechanics
Department of Mathematics
Jadranska 19, 1000 Ljubljana, Slovenia

Preprint series, Vol. 39 (2001), 792

VISUALIZATION OF GRAPHS AND RELATED DISCRETE STRUCTURES IN *MATHEMATICA*

Tomaž Pisanski Marko Boben

Arjana Žitnik

ISSN 1318-4865

December 7, 2001

Ljubljana, December 7, 2001

Visualization of Graphs and Related Discrete Structures in *Mathematica*

Tomaž Pisanski*, Marko Boben and Arjana Žitnik,
University of Ljubljana, Slovenia

Abstract

Research in discrete mathematics can be conducted more efficiently if one can visualize discrete structures such as graphs, groups, polyhedra, maps, posets, lattices, tilings, incidence structures, etc. The *Mathematica* based computer package VEGA that our group is developing over the years has many automatic drawing programs built in. We review several methods for drawing graphs and present a way of computer representation of other discrete structures which can be visualized as graphs.

Sažetak

Istraživanja u diskretnoj matematici moguće je mnogo efikasnije provoditi ukoliko možemo vizualizirati diskretne strukture, kao što su grafovi, grupe, poliedri, preslikavanja, parcialno uređeni skupovi (posets), rešetke, popločenja (tilings), incidentne strukture, itd. Programski paket VEGA, baziran na *Mathematici*, kojeg već nekoliko godina razvija naša grupa, ima ugrađeno mnogo programa za automatsko crtanje. Prikazane su neke metode crtanja grafova, koje upotrebljava VEGA, te njihova reprezentacija, kao i reprezentacija drugih struktura, koje možemo vizualizirati pomoću grafova.

Key words: *Mathematica*, graphs, representation, visualization, drawing methods.
Math. Subj. Class. (2000): 05C62, 05C85, 68R05, 68R10.

1 Introduction

VEGA is a system for doing Discrete Mathematics. It is a *Mathematica* based collection of operations with interface to external packages and programs. In 1990 we started a project by adding an interface from "Combinatorica" by Steven Skiena [12, 13] to "Nauty" by Brendan McKay [6, 7]. Soon it became obvious that continuous additions and modifications of the project produced

*Tomaz.Pisanski@fmf.uni-lj.si, supported in part by Ministrstvo za šolstvo, znanost in šport Republike Slovenije, grant J1-6161, J2-6193. Part of the research was conducted while the author was visiting the DIMACS center in NJ. The paper is based on the invited lecture on `PrimMath[2001]`, Zagreb, September 27–28, 2001.

an entirely new system that we called VEGA. Tens of students and colleagues throughout the world have contributed to the VEGA project.

The ideas behind VEGA are simple. The project should be based on a powerful and machine independent system like *Mathematica* or *Maple*. It should provide an integrated and user friendly environment in which researchers, teachers and students of Discrete Mathematics, Theoretical Computer Science, Mathematical Chemistry or any other branch of science in which graphs are used, can quickly test ideas and hypotheses on small and mid-size examples. Most algorithms are written in *Mathematica*. If they are time consuming then they are replaced by efficient algorithms written in C, C++ or Pascal.

External programs are used for planarity testing, for finding Hamilton cycles in cubic graphs, for automatic drawings of graphs, for 2-factorization of a regular $2d$-valent graph, etc. Nauty is used for finding the automorphism group of a graph. There are also files providing the interface to other non-profit software like Nauty or GAP [3].

The documentation of VEGA is written in HTML and is available on the Internet [8]. All users and contributors to the VEGA Project can follow the development of VEGA by reading Vega News on `http://vega.ijp.si/Htmldoc/veganews/`.

Currently VEGA contains several data structures such as graph, poset, group, map, polyhedron, network, configuration, molecule, etc. with appropriate selectors, constructors and functors.

In this article we focus our attention to visualization of graphs. Graphs are visualized in $\mathbb{R}^2$ by means of a map $\rho : VG \to \mathbb{R}^2$, which is called a *representation* of a graph in $\mathbb{R}^2$ [4]. We present some automatic drawing routines which we use in VEGA. Later on we give examples of other data structures which are related to graphs. The notion of representation of graphs and related structures is much more explored in the survey [10].

2 Representations of graphs

Let $\mathbb{F}$ be any field and $m \geq 0$. We denote by $\mathbb{F}^m$ the m-dimensional vector space over $\mathbb{F}$. We define a *representation* ρ of a graph G in $\mathbb{F}^m$ to be a map ρ from the set of vertices VG into $\mathbb{F}^m$. If we regard vectors $\rho(u)$, $u \in VG$, as row vectors, we may represent ρ by $|VG| \times m$ matrix R with the images of the vertices of G as its rows. When we want to avoid confusion we call such a representation a *vector representation* of graphs. The representation ρ is *orthonormal* if $R^T R = I_m$. A representation ρ is called *balanced* if $\sum_{u \in VG} \rho(u) = 0$. The idea of graph representation goes back at least to Tutte [15, 16], where it is stated primarily for planar graphs.

The *energy* of the representation ρ is defined in general form to be the value:

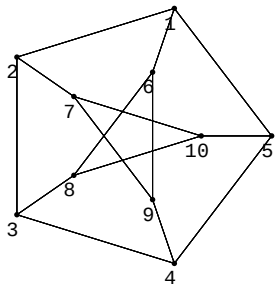
$$\mathcal{E}(\rho) = \sum_{uv \in EG} \omega_{uv} ||\rho(u) - \rho(v)||^2 \tag{1}$$

where $\omega : EG \to \mathbb{R}^+$ is a map defining an edge-weighted graph.

A representation ρ can be also regarded as a drawing or embedding of the graph G into $\mathbb{F}^m$. The vertices of a graph are clearly points in the vector space, whose position is determined by ρ. For a complete drawing of the graph one has to represent the edges of G in the same space. This is a natural motivation for

extending the concept of representation to edges. There are several "natural" edge-extensions of ρ. In the case $\mathbb{F} = \mathbb{R}$ the most natural representation for edges is $\rho(uv) = \mathrm{conv}(\rho(u), \rho(v))$, where the *convex hull* $\mathrm{conv}(X)$ of a set $X = \{x_1, x_2, ..., x_n\}$ is the set of all points $\sum_{i=1}^{n} \lambda_i x_i$ with $\sum_{i=1}^{n} \lambda_i = 1$. This idea was used, for instance by Tutte [16] under the name of *straight representation*, when embedding planar graphs with straight line segments in the plane $\mathbb{R}^2$. Since we are concerned with graphs (and not maps) we allow degeneracies (edge-crossings, extra vertices on the edge segment, etc.).

In the package VEGA, the data structure `Graph` consists of a list of adjacency lists of vertices and a list of coordinates of vertices, which gives us a representation of the graph in $\mathbb{R}^2$: `Graph[`adjacency lists, coordinates of vertices`]`. An example is given in Figure 1.




```
Graph[{{2,5,6},{1,3,7},{2,4,8},{3,5,9},{1,4,10},{1,8,9},{2,9,10},
  {3,6,10},{4,6,7},{5,7,8}},{{0.31,0.95},{-0.81,0.59},{-0.81,-0.59},
  {0.31,-0.95},{1.,0},{0.15,0.48},{-0.4,0.29},{-0.4,-0.29},
  {0.15,-0.48},{0.5,0}}]
```

Figure 1: Petersen graph and its representation in VEGA.

VEGA includes several automatic drawing routines which produce pleasing drawings of graphs in $\mathbb{R}^2$. They can be divided into two categories:

- Exact methods: Laplace method [9], Tutte method [16].
- Iterative methods: spring-embedders such as the one by Kamada and Kawai [5], Fruchterman and Reingold [2] or Shlegel diagram by B. Plestenjak [11].

They are all based on minimizing a given type of the energy of a representation [4]. Iterative methods are too time-consuming to implement in *Mathematica* and are implemented in other languages. Laplace method and Tutte method are particularly suited for *Mathematica*, since *Mathematica* contains all the necessary tools such as finding eigenvectors of a matrix and solving systems of linear equations.

The Laplace method. The matrix Q with the elements $q_{uv} = -\omega_{uv}, uv \in EG, q_{uv} = 0, uv \notin EG, q_{uu} = -\sum_{uv \in EG} q_{uv}$ is called the *generalized Laplacian* of an edge-weighted graph G.

Theorem 2.1 ([9, 4]). *Let G be an edge-weighted graph with edge-weights ω and the generalized Laplacian Q. Assume that the eigenvalues of Q are $\lambda_1 \leq \cdots \leq \lambda_n$ and that $\lambda_2 > 0$. The minimum energy of a balanced orthonormal representation of G in $\mathbb{R}^m$ equals $\sum_{i=2}^{m+1} \lambda_i$.*

Note that the orthonormal representation ρ of the above Theorem is given by the matrix $[x_2, \ldots, x_{m+1}]$ composed of orthonormal eigenvectors corresponding to $\lambda_2, \ldots, \lambda_{m+1}$. For $m = 2$ and $m = 3$ we get a graph drawing in $\mathbb{R}^2$ and $\mathbb{R}^3$, respectively. Examples of such drawings are shown in Figure 2. Any procedure that obtains a representation of a graph by solving the eigenvalue and eigenvector problem will be called the *eigenvector method*. In the above case, Theorem 2.1 guarantees that the eigenvector method produces a representation that minimizes the energy given by (1).

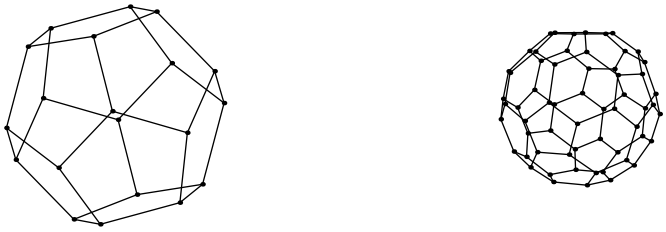


Figure 2: A $\mathbb{R}^2$ representation of the dodecahedron using second and third eigenvectors of the Laplace matrix Q and a $\mathbb{R}^3$ representation of the C_{60} fullerene, using second, third and fourth eigenvectors.

The Tutte method. A cycle C of G is called *peripheral* if no edge not in C joins two vertices in C and $G \setminus C$ is connected. For example, any face of a 3-connected planar graph can be shown to be a peripheral cycle.

We say, that a representation ρ of G is *barycentric* relative to a subset S of VG if for each $u \notin S$ the vector $\rho(u)$ is the barycenter of the images of neighbors of u.

Lemma 2.2. *Let G be a connected graph, let S be a subset of vertices of G, and let σ be a map from S into $\mathbb{R}^m$. If $G \setminus S$ is connected, there is a unique m-dimensional representation ρ of G that extends σ and is barycentric relative to S.*

Theorem 2.3 (Tutte). *Let C be a peripheral cycle in a connected graph G. Let σ be a mapping from VC to the vertices of a convex $|VC|$-gon in $\mathbb{R}^2$ such that adjacent vertices in C are adjacent in the polygon. The unique barycentric representation determines a drawing of G in $\mathbb{R}^2$. This drawing has no crossings if and only if the graph is planar.*

A barycentric drawing based on this theorem is obtained by solving the system of equations

$$\rho(v) = \frac{1}{deg(v)} \sum_{u \in N(v)} \rho(u), \qquad v \in VG \setminus S.$$

It is sometimes called the *Tutte drawing* of a graph.

The generalized Tutte alias Schlegel method. We have developed a similar approach. Let S be a subset of vertices VG. For each vertex $v \in VG$ let $\delta(v)$ denote the distance from S. Define $\omega_{uv} = \phi(\delta(u), \delta(v))$, for a suitable symmetric function ϕ such as $\omega_{uv} = 1 + |\delta(u) - \delta(v)|^p$, or $\omega_{uv} = \frac{1}{\max(\delta(u),\delta(v))^p}$, for some parameter $p \in \mathbb{R}$. Select a map σ from S into $\mathbb{R}^m$. The corresponding weighted barycentric representation ρ is called the *Schlegel representation* of G with respect to S. It is defined by

$$\rho(v) = \frac{1}{\omega_{vv}} \sum_{u \in N(v)} \omega_{uv}\rho(u), \qquad v \in VG \setminus S.$$

Figure 3 shows Tutte and Schlegel drawings of graphs.

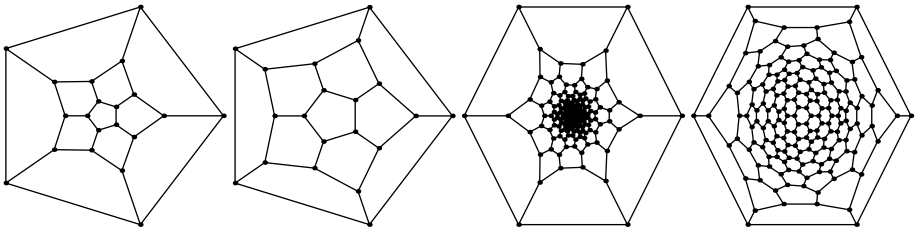


Figure 3: A $\mathbb{R}^2$ representation of the dodecahedron – Tutte method and Schlegel method with $\omega_{uv} = \frac{1}{\max(\delta(u),\delta(v))}$ and a $\mathbb{R}^2$ representation of $Le(C_{60})$ – Tutte method and Schlegel method with $\omega_{uv} = \frac{1}{\max(\delta(u),\delta(v))^{2.5}}$.

3 Other data structures in VEGA

Several other data structures that we use for description of various mathematical structures such as maps, networks, configurations, automata, posets, molecules, knots, etc. are implemented in VEGA. All implementations follow the same concept.

1. The data type is determined by the corresponding `Head` of *Mathematica* expression, such as `Graph`, `MapGraph`, `Network`, `Configuration`, `Automaton`, `Poset`, `Molecule`, etc.

2. The structure is manipulated via predefined operations: *selectors* such as `AdjacencyLists`, `Edges`, `Coordinates`, ... , and *constructors* such as `FromAdjacencyLists`, `FromEdges`, ... in case of data structure `Graph`. No other direct access to the implementation is allowed. Users of VEGA have no need to study internal representation of data. When developing VEGA we made several changes in the internal representation of graphs and other data structures and only kernel functions had to be re-written.

3. Various *functors* transform objects from one data structure to another one. For instance, the forgetful functor `Graph[net_Network]` forgets the values on the edges of a network `net` and returns the underlying graph.

4. The `Format` feature of *Mathematica* makes objects user friendly since they are displayed automatically during the interactive session. For each structure there exists a default drawing routine which is used by `Format`.

5. There exist functions which enable the user to export the figures of graphs and other structures as short Encapsulated PostScript files.

6. Documentation is extracted in a form of interlinked HTML files and is updated after new version is released.

Since many of these structures (networks, automata, posets, molecules) are naturally connected to graphs, the problem of their visualization can be reduced to the problem discussed in the previous section.

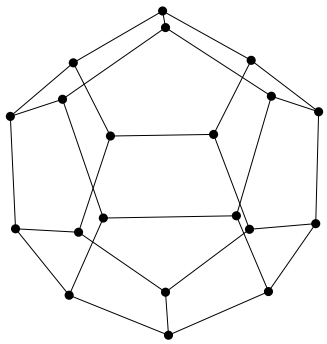
In the case of configurations, the algorithm from [1] is used. For (v_3) configurations it produces a representation of the structure in the plane using at most one curved line which reflects a well known result obtained by Steinitz [14].

In the remainder of the section we give a description of data structures which are used to represent certain mathematical structures in VEGA. Of course, the available space does not allow us to present the subject in its full range. For details, the reader is invited to read the Vega Manual pages on the Internet [8].

3.1 Representation of maps

Data structure: `MapGraph[{`# of vertices, # of edges, # of faces`}`, edges, faces, coordinates`]`

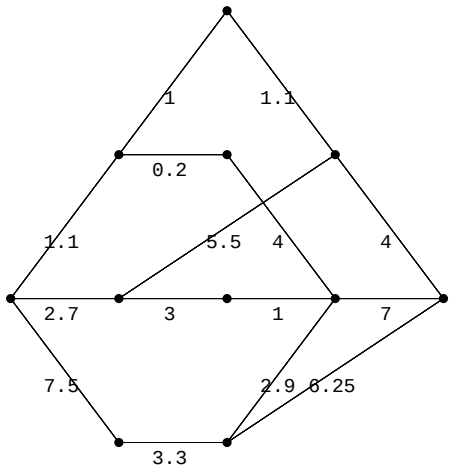
Example: Dodecahedron.



3.2 Representation of networks

Data structure: `Network[`weighted edges, coordinates`]`

Example:

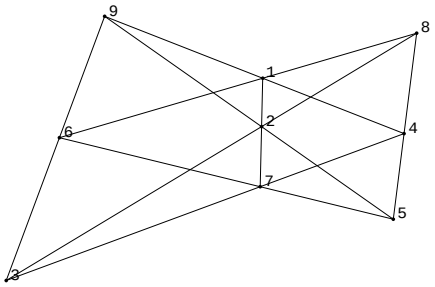



```
Network[{{1,2,1.1},{1,3,2.7},{1,4,7.5},{2,5,1},{2,6,0.2},
  {3,9,3},{3,7,5.5},{4,8,3.3},{5,9,1.1},{6,10,4},{7,10,1},
  {8,10,2.9},{8,11,6.25},{9,11,4},{10,11,7}},
 {{0,0},{1,1},{1,0},{1,-1},{2,2},{2,1},{2,0},{2,-1},{3,1},
  {3,0},{4,0}}]
```

3.3 Representation of configurations

Data structure: `Configuration`[list of lines, possible additional information (projective, affine coordinates, projection matrix, etc.)]

Example: Pappus configuration.

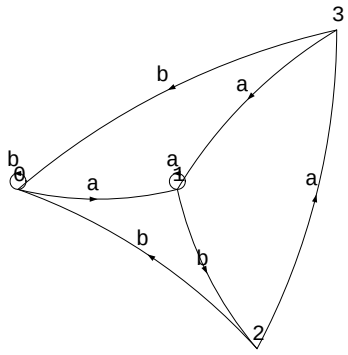



```
Configuration[{{1,2,7},{3,4,7},{5,6,7},{1,6,8},{2,3,8},
   {4,5,8},{1,4,9},{3,6,9},{2,5,9}},
  PrMatrix->{{0.17,0.5,0.55},{0.66,0.11,0.92},{1.14,0.87,0.6}},
  Coordinates->{{0.37,0.,0.93},{-0.27,-0.67,-0.69},
   {-0.28,-0.68,0.68},{0.24,-0.78,-0.58},{0.29,-0.96,0},
   {1.,0,0},{0,1.,0},{0,0,1.},{0.58,-0.58,0.58}}]
```

3.4 Representation of automata

Data structure: `Automaton`[DFA/NDFA, list of states, list of transitions, initial state, list of final states, alphabet, coordinates]

Example: Automaton recognizing a language of strings over $\{a, b\}$ ending with aba.

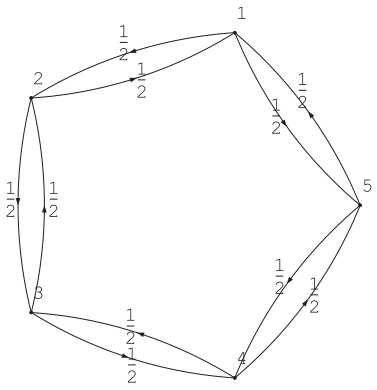



```
Automaton[DFA,{0,1,2,3},{{{"b",1},{"a",2}},{{"b",3},{"a",2}},
 {{"b",1},{"a",4}},{{"b",1},{"a",2}}},1,{4},{"a","b"},
 {{0,0},{1,0},{1.5,-1},{2,1}}]
```

3.5 Representation of Markov chains

Data structure: `MarkovChain`[list of transition triples, coordinates]

Example: A walk on a cycle.

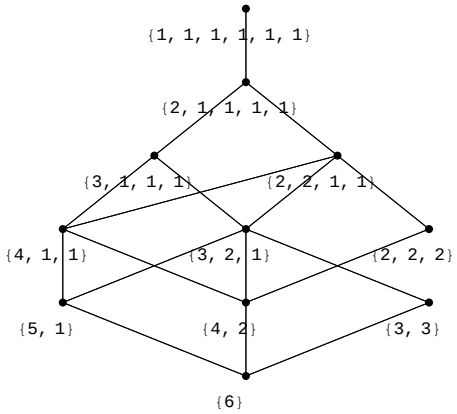



```
MarkovChain[{{1,2,1/2},{1,5,1/2},{2,3,1/2},{2,1,1/2},{3,4,1/2},
  {3,2,1/2},{4,5,1/2},{4,3,1/2},{5,1,1/2},{5,4,1/2}},
  {{0.31,0.95},{-0.81,0.59},{-0.81,-0.59},{0.31,-0.95},{1.,0}}]
```

3.6 Representation of posets

Data structure: `Poset`[poset adjacencies, coordinates, labels]

Example: The lattice of partitions of the number 6.

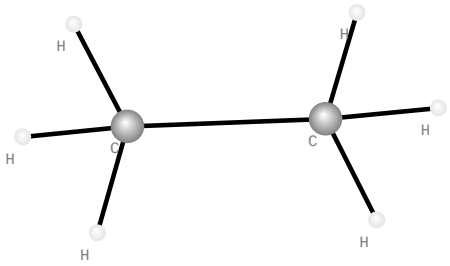



```
Poset[{{},{1},{1},{1},{3,2},{4,3,2},{3},{6,5},{7,6,5},{9,8},{10}},
 {{2/3,1/7},{1/3,2/7},{2/3,2/7},{1,2/7},{1/3,3/7},{2/3,3/7},
  {1,3/7},{1/2,4/7},{5/6,4/7},{2/3,5/7},{2/3,6/7}},
 {{6},{5,1},{4,2},{3,3},{4,1,1},{3,2,1},{2,2,2},{3,1,1,1},
  {2,2,1,1},{2,1,1,1,1},{1,1,1,1,1,1}}]
```

3.7 Representation of molecules

Data structure: `Molecule[Graph` object, labels]

Example: Ethane.

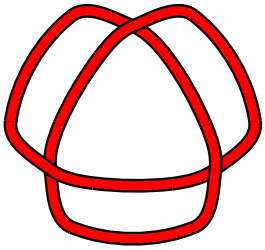



```
Molecule[Graph[{{2,3,4,5},{1},{1},{1},{1,6,7,8},{5},{5},{5}},
 {{0.78,0.80},{0.65,1.05},{0.53,0.78},{0.70,0.55},{1.25,0.82},
  {1.32,1.07},{1.37,0.58},{1.52,0.85}}],
  {"C","H","H","H","C","H","H","H"}]
```

3.8 Representation of knots

Data structure: `ComputedKnot[`list of x and y coordinates and the tangent angle in degrees for every crossing or auxiliary point, list determining bridges and tunnels, list of labels together with their coordinates]

Example:



```
ComputedKnot[{{{0.00,0.78,-157.},{-0.27,0.53,-127.},
  {-0.60,-0.02,-113.},{-0.68,-0.39,-83.3},{-0.5,-0.87,-22.},
  {0.5,-0.87,22.},{0.68,-0.39,83.2},{0.56,-0.03,113.},
  {0.28,0.53,127.},{0.00,0.78,157.},{-0.5,0.86603,-142.},
  {-1.,0.,-98.},{-0.68,-0.39,-36.8},{-0.33,-0.50,-6.83},
  {0.32,-0.51,6.53},{0.68,-0.39,36.7},{1.,0.,98.},
  {0.5,0.87,142.}}},{{1,0,0,-1,0,0,1,0,0,-1,0,0,1,0,0,-1,0,0}},
 {{1,1,{0.00,0.78}},{6,2,{-0.68,-0.39}},{11,3,{0.68,-0.39}}}]
```

References

[1] J. Bokowski, B. Sturmfels, *Computational Synthetic Geometry*, Lecture Notes in Mathematics 1355, Springer, Heidelberg, 1989.

[2] T. M. J. Fruchterman and E. M. Reingold, *Graph drawing by force–directed placement*, Software Practice and Experience 21 (1991) 1129–1164.

[3] The GAP group, *GAP – Groups, Algorothms and Programming*, see `http://www.math.rwth-aachen.de/~GAP/`.

[4] C. Godsil and G. Royle, *Algebraic Graph Theory*, Springer, New York, 2001.

[5] T. Kamada and S. Kawai, *An algorithm for drawing general undirected graphs*, Inform. Process. Lett. 31 (1989) 7–15.

[6] B. McKay, *Nauty*, see `http://cs.anu.edu.au/~bdm/nauty/`.

[7] B. McKay, *Practical Graph Isomorphism,* Congressus Numerantium 30 (1981) 45–87.

[8] T. Pisanski & coworkers, VEGA, a programming tool for manipulating discrete mathematical structures, see `http://vega.ijp.si`.

[9] T. Pisanski and J. Shawe-Taylor, *Characterizing Graph Drawing with Eigenvectors*, J. Chem. Inf. Comput. Sci. 40 (2000) 567–571.

[10] T. Pisanski and A. Žitnik, *Representation of Graphs and Maps*, work in progress.

[11] B. Plestenjak, *An Algorithm for Drawing Planar Graphs*, Software – Practice and Experience 29 (1999) 973–984.

[12] S. Skiena, *Combinatorica*, see
`http://www.cs.sunysb.edu/~skiena/combinatorica/index.html`.

[13] S. Skiena, *Implementing Discrete Mathematics: Combinatorics and Graph Theory in Mathematica,* Advanced Book Division, Addison-Wesley, Redwood City CA, June 1990.

[14] E. Steinitz, *Über die Construction der Configurationen* n_3, Inaugural-Dissertation, Breslau (1894).

[15] W. T. Tutte, *Convex representations of graphs*, Proc. London Math. Soc. 10 (1960) 304–320.

[16] W. T. Tutte, *How to draw a graph*, Proc. London Math. Soc. 13 (1963) 743–767.