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

Preprint series, Vol. 40 (2002), 828

INTERACTIVE CONJECTURING WITH VEGA

Tomaž Pisanski   Marko Boben
Arjana Žitnik

ISSN 1318-4865

July 4, 2002

Ljubljana, July 4, 2002

Interactive conjecturing with Vega

Tomaž Pisanski*, Marko Boben and Arjana Žitnik
University of Ljubljana, IMFM
Jadranska 19, 1000 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. The expert knowledge built in Vega is intended for interactive work that usually helps formulating and solving research problems.

Key words: *Mathematica*, graphs, representation, visualization, drawing methods, configurations, Balaban graph, Dyck graph, groups, Laplace matrix, Schlegel diagram, poset, knot.
Math. Subj. Class. (2000): 05C62, 05C85, 68R05, 68R10.

1 Introduction

In this article we try to present Vega, a *Mathematica* [30] based system for manipulating discrete mathematical structures. In the first part we present a history of Vega, its basic features and principles which we try to follow in the development of the system. In the second part we focus to the visualization of graphs and give some theory. We also describe some data structures which are used in Vega to represent graphs and other related objects such as maps, networks, configurations, posets, knots, etc. Finally we present an example that shows how new knowledge can be obtained using Vega.

2 An Overview of Vega

In 1991 Steve Fisk was visiting Slovenia and we first got hold of *Mathematica* due to Marko Petkovšek, who brought good knowledge of *Mathematica* with him after completing his Ph. D. at CMU and after working for Wolfram Research where he developed the package `RSolve`. We also knew Nauty [16, 17] by

*`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.

Brendan McKay. The first idea was to write an interface that would allow Steve Skiena's package `Combinatorica` [25, 26] to communicate with Nauty and thus improve significantly the isomorphism checking for graphs.

The next step was to use the `Format` feature to improve the way for a user to understand the graphs that he or she is working with in a *Mathematica* session.

Soon we began to use this hybrid in our teaching and research process. Students in various levels of their studies were asked to choose a project from a growing list of *projects to do*.

Several students implemented various graph drawing algorithms that were written in Pascal and then linked to *Mathematica* by a simple mechanism:

- Write data D on a temporary file.
- Call an external program P that reads data D and writes the results on another file R.
- Read results from R.

Two of the most successful external programs are `NiceGraph` and `Sclegel-Diagram`.

At that point we decided to give our package a name. We have chosen the name VEGA honoring Slovenian mathematician and scientist Baron Jurij Vega (1754–1802) who was the first person to calculate 140 digits of π and held world record for over fifty years.

When working with sparse graphs we noticed that the idea used in `Combinatorica` of storing graphs as adjacency matrices was not the most efficient. We replaced `AdjacencyMatrix` by `AdjacencyList` internal representation.

The replacement of the internal representation was done in two steps.

- We made sure that the data structure `Graph` has a number of *constructors* and *selectors* that were written on a separate file. These were the only programs that were allowed to work directly with the internal structure of the graph. For instance: `AdjacencyMatrix[g_Graph] := g[[1]]`.
- All other programs that are dealing with a graph may do that only via selectors and constructors. A typical call would look like:

```
MyGraphInvariant[g_Graph] :=
  Module[{al = AdjacencyLists[g], coo = Coordinates[g]},...]
```

Our work with chemists forced us to expand our graphs to three dimensions. We first introduced a separate data structure `Graph3D` which is now obsolete but was followed by numerous useful data structures, some of which are presented in this paper.

Some of these data structures were quite general. For instance, we introduced groups and maps: `Group`, `MapGraph`. The group is given by a set of generators and the group operation. The maps are a generalization of polyhedra. The main difference is that we keep faces abstract and apart from vertex coordinates.

Some of the data structures were introduced for a specific purpose or as a tool for solving a specific research problem. For instance, we have permutation groups, voltage graphs (used in covering graph constructions), colored graphs, etc.

We considered our data structures as categories and provided basic functors that allow us to transform an object belonging to one category to an object from a different category. For instance, the forgetful functor `Graph` takes a map `m_MapGraph` and constructs its one-skeleton graph `g = Graph[m]`.

A big step forward was done by Bor Plestenjak, the author of the `View` external program. He wrote a series of other programs, for instance, the program `Fullerene` that generates fullerenes with some prescribed properties at random using the so-called Stone-Wales transformations [4].

VEGA was growing so fast that our students wrote a series of utility programs for automatic documentation first in `techinfo` and later in `html`.

Bor Plestenjak also wrote a program `WriteEPS` for exporting graph drawings as a short and concise EPS file. This was the first in a series of programs for drawing VEGA data structures in PostScript. Now we were able to use VEGA for support in our paper writing.

Is there a future for VEGA? It has grown so big that it should be totally rewritten. In the process of developing VEGA we have learned a lot and with an appropriate financial support would be quite capable to design and implement a much more up to date version of VEGA.

We always believed it is better to use a good product than to re-invent and re-implement it. That is why we immediately used Nauty. We also wrote and interface to GAP [6] when we first obtained a copy of it. We also considered an interface with LEDA [13] and MuPad [18] but we were never able to find a practical solution for it. Any future development of VEGA should include an interface to Magma [15].

For a few years we have frozen our development of VEGA but we develop smaller packages in addition to it. For instance, we have a successful package for working with configurations and another one for the so-called flag systems.

3 Representations 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 $\mathbb{R}^2$, compare [7, 14]. The notion of representation of graphs and related structures is much more explored in the survey [23].

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 [28, 29], where it is stated primarily for planar graphs.

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

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

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 $X = \{x_1, x_2, \dots, x_n\}$ 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 [29] 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).

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 [22], Tutte method [29].
- Iterative methods: spring-embedders such as the one by Kamada and Kawai [9], Fruchterman and Reingold [5] or Schlegel diagram by B. Plestenjak [24].

They are all based on minimizing a given type of the energy of a representation [7].

Laplace method and Tutte method are particularly suited for implementation in *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.

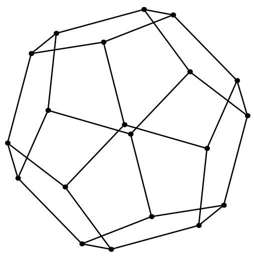
Theorem 3.1 ([22, 7]). *Let G be an edge-weighted graph with edge-weights ω and the generalized Laplacian Q. Assume that the eigenvalues of Q are $\lambda_1 \leq \cdots \leq \lambda_n$ and that $\lambda_2 > 0$. The minimum energy of a balanced orthonormal representation of G in $\mathbb{R}^m$ equals $\sum_{i=2}^{m+1} \lambda_i$.*

Note that the orthonormal representation ρ of the above Theorem is given by the matrix $[x_2, \dots, x_{m+1}]$ composed of orthonormal eigenvectors corresponding to $\lambda_2, \dots, \lambda_{m+1}$. 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 1. 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 3.1 guarantees that the eigenvector method produces a representation that minimizes the energy given by (1).

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 3.2. *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*



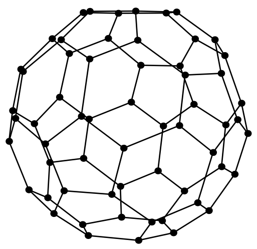


Figure 1: A $\mathbb{R}^2$ representation of the dodecahedron using second and third eigenvectors of the Laplace matrix Q and a $\mathbb{R}^3$ representation of the C_{60} fullerene, using second, third and fourth eigenvectors.

m-dimensional representation ρ of G that extends σ and is barycentric relative to S.

Theorem 3.3 (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

$$\rho(v) = \frac{1}{\deg(v)} \sum_{u \in N(v)} \rho(u), \qquad v \in VG \setminus S, \tag{2}$$

where $N(v)$ denotes the set of neighbors of v. 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 2 shows Tutte and Schlegel drawings of graphs.

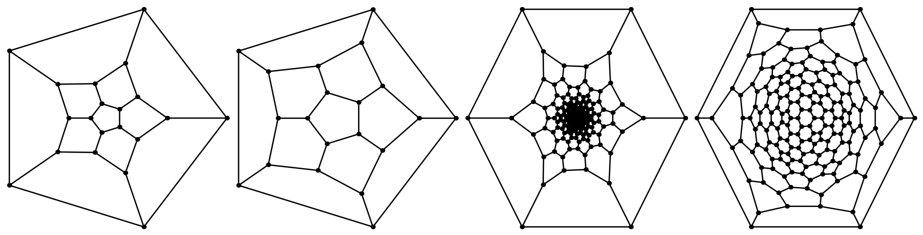


Figure 2: A $\mathbb{R}^2$ representation of the dodecahedron – Tutte method and Schlegel method with $\omega_{uv} = \frac{1}{\max(\delta(u),\delta(v))}$ and a $\mathbb{R}^2$ representation of $Le(C_{60})$ – Tutte method and Schlegel method with $\omega_{uv} = \frac{1}{\max(\delta(u),\delta(v))^{2.5}}$.

Another interpretation of the Tutte method. Suppose that we fix a representation ρ of the vertices in $S \subseteq VG$ of a connected graph G. Now, let us consider the following random process (Markov chain) on VG: the process stays in a vertex $v \in S$ with probability 1, otherwise it moves from $v \in VG \setminus S$ to any neighbor with probability $1/\deg(v)$. After a sufficient number of steps the process reaches a state $v \in S$ with probability 1 (and stays there). If we denote by p^n_{vu} a transition probability from v to u after n steps and with p^∞_{vu} the limit $\lim_{n\to\infty} p^n_{vu}$, then $p^\infty_{vu} = 0$ for $u \notin S$. The idea for extending representation ρ to the vertices not in S is to use limit probabilities p^∞_{vu} for the weights:

$$\rho(v) = \sum_{u\in S} p^\infty_{vu}\, \rho(u). \tag{3}$$

The following theorem says that we actually obtain the Tutte drawing of G.

Theorem 3.4. *The representation defined by* (3) *is the same as the Tutte representation obtained from* (2).

Remark. This theorem admits a straight-forward generalization to the Schlegel method representation.

It is perhaps of interest to note that the extension of ρ can be expressed using formal coordinates $\rho(u)$ for the vertices $u \in S$ since the weights p^∞_{vu} in (3) are independent of $\rho(u)$, $u \in S$.

The idea of this Theorem is due to Tom Tucker when he attended a talk in which the Schlegel method was explained. Both representations were tested on numerous examples. This helped us to find the proof of the Theorem.

Note that L. Lovász independently considers the Schlegel generalization of the Tutte method [14].

4 Representations of discrete structures in Vega

Currently Vega implements several data structures used for description of graphs, maps, networks, configurations, automata, posets, molecules, knots, etc. All implementations follow the same concept.

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

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

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.

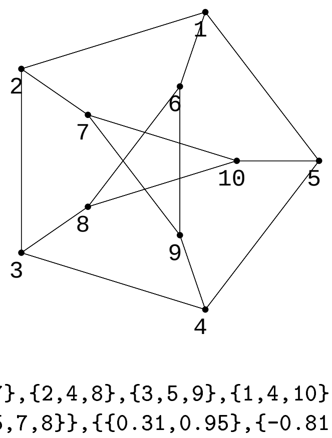
In the case of configurations, the algorithm from [2] is used. For (v_3) configurations it produces a representation of the structure in the plane using at most one curved line which reflects a well known result obtained by Steinitz [27].

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 [19], see also [21].

4.1 Representation of graphs

Data structure: `Graph`[adjacency lists, coordinates of vertices]

Example: Petersen graph.

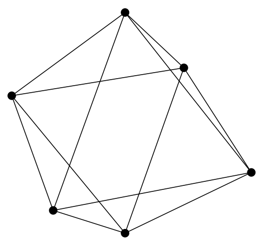


```
Graph[{{2,5,6},{1,3,7},{2,4,8},{3,5,9},{1,4,10},{1,8,9},{2,9,10},
  {3,6,10},{4,6,7},{5,7,8}},{{0.31,0.95},{-0.81,0.59},
  {-0.81,-0.59},{0.31,-0.95},{1.,0},{0.15,0.48},{-0.4,0.29},
  {-0.4,-0.29},{0.15,-0.48},{0.5,0}}]
```

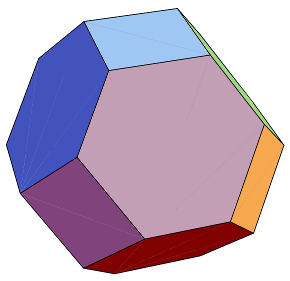
4.2 Representation of maps

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

Example: Octahedron.



Octahedron



Truncated octahedron
(polyhedral representation)

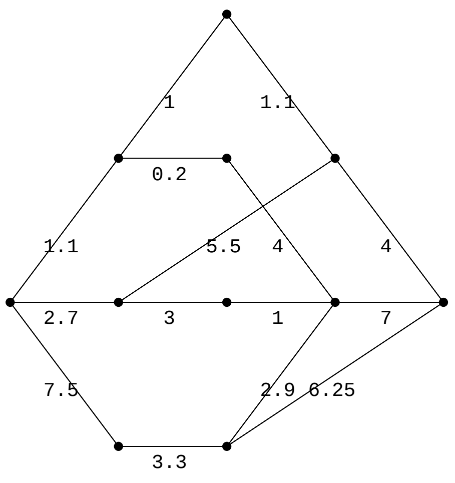
```
MapGraph[{6,12,8},{{{{2,1},{3,1},{4,1},{5,1}}},
  {{{5,1},{6,1},{3,1},{1,1}}},{{{2,1},{6,1},{4,1},{1,1}}},
  {{{3,1},{6,1},{5,1},{1,1}}},{{{4,1},{6,1},{2,1},{1,1}}},
  {{{2,1},{5,1},{4,1},{3,1}}}},
 {{6,2},{6,5},{6,4},{6,3},{5,4},{5,2},{5,1},{4,3},{4,1},{3,2},
  {3,1},{2,1}},
 {{{{4,6},1},{{6,5},1},{{5,4},1}},{{{3,6},1},{{6,4},1},{{4,3},1}},
  {{{2,5},1},{{5,6},1},{{6,2},1}},{{{2,6},1},{{6,3},1},{{3,2},1}},
  {{{1,2},1},{{2,3},1},{{3,1},1}},{{{1,3},1},{{3,4},1},{{4,1},1}},
  {{{1,4},1},{{4,5},1},{{5,1},1}},{{{1,5},1},{{5,2},1},{{2,1},1}}},
 {{0.,0.,1.41421},{1.41421,0.,0.},{0.,1.41421,0.},{-1.41421,0.,0.},
  {0.,-1.41421,0.},{0.,0.,-1.41421}}]
```

Remark: This structure also allows a description of non-orientable surfaces, pseudo-surfaces, and non-simple graphs (loops and parallel edges). In addition, there exist numerous transformations on maps such as dual, medial, leapfrog, truncation, etc. Conversion of a `MapGraph` structure to a polyhedron (`Graphics3D` object) is also possible.

4.3 Representation of networks

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

Example:

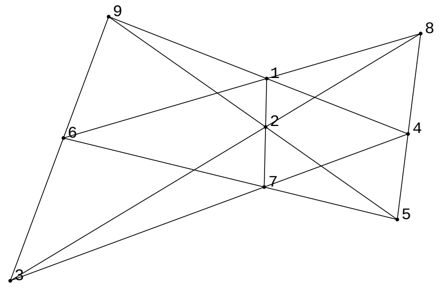


```
Network[{{1,2,1.1},{1,3,2.7},{1,4,7.5},{2,5,1},{2,6,0.2},
  {3,9,3},{3,7,5.5},{4,8,3.3},{5,9,1.1},{6,10,4},{7,10,1},
  {8,10,2.9},{8,11,6.25},{9,11,4},{10,11,7}},
 {{0,0},{1,1},{1,0},{1,-1},{2,2},{2,1},{2,0},{2,-1},{3,1},
  {3,0},{4,0}}]
```

4.4 Representation of configurations

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

Example: Pappus configuration.

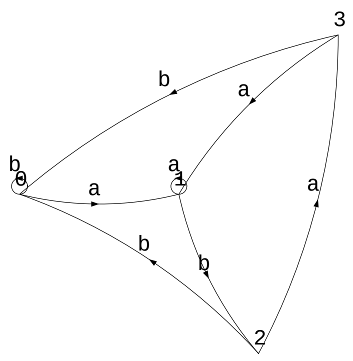


```
Configuration[{{1,2,7},{3,4,7},{5,6,7},{1,6,8},{2,3,8},
   {4,5,8},{1,4,9},{3,6,9},{2,5,9}},
  PrMatrix->{{0.17,0.5,0.55},{0.66,0.11,0.92},{1.14,0.87,0.6}},
  Coordinates->{{0.37,0.,0.93},{-0.27,-0.67,-0.69},
   {-0.28,-0.68,0.68},{0.24,-0.78,-0.58},{0.29,-0.96,0},
   {1.,0,0},{0,1.,0},{0,0,1.},{0.58,-0.58,0.58}}]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

4.5 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.

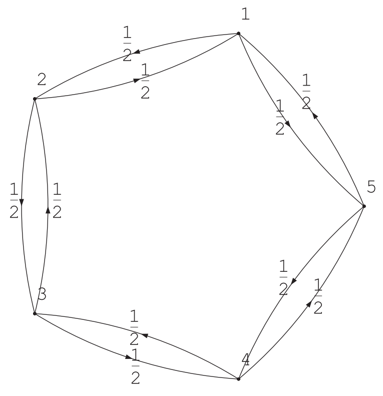


```
Automaton[DFA,{0,1,2,3},{{{"b",1},{"a",2}},{{"b",3},{"a",2}},
 {{"b",1},{"a",4}},{{"b",1},{"a",2}}},1,{4},{"a","b"},
 {{0,0},{1,0},{1.5,-1},{2,1}}]
```

4.6 Representation of Markov chains

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

Example: A walk on a cycle.

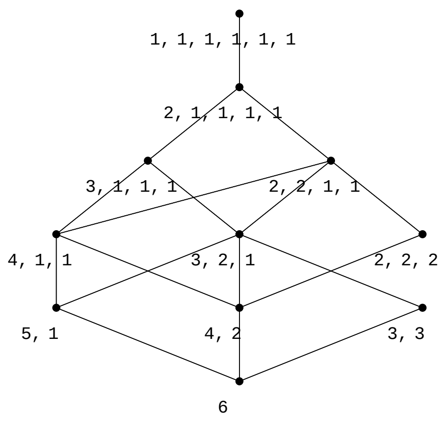


```
MarkovChain[{{1,2,1/2},{1,5,1/2},{2,3,1/2},{2,1,1/2},{3,4,1/2},
  {3,2,1/2},{4,5,1/2},{4,3,1/2},{5,1,1/2},{5,4,1/2}},
  {{0.31,0.95},{-0.81,0.59},{-0.81,-0.59},{0.31,-0.95},{1.,0}}]
```

4.7 Representation of posets

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

Example: The lattice of partitions of the number 6.

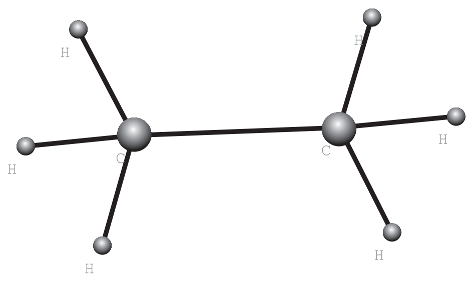


```
Poset[{{},{1},{1},{1},{3,2},{4,3,2},{3},{6,5},{7,6,5},{9,8},{10}},
 {{2/3,1/7},{1/3,2/7},{2/3,2/7},{1,2/7},{1/3,3/7},{2/3,3/7},
  {1,3/7},{1/2,4/7},{5/6,4/7},{2/3,5/7},{2/3,6/7}},
 {{6},{5,1},{4,2},{3,3},{4,1,1},{3,2,1},{2,2,2},{3,1,1,1},
  {2,2,1,1},{2,1,1,1,1},{1,1,1,1,1,1}}]
```

4.8 Representation of molecules

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

Example: Ethane.

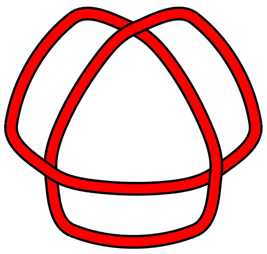


```
Molecule[Graph[{{2,3,4,5},{1},{1},{1},{1,6,7,8},{5},{5},{5}},
 {{0.78,0.80},{0.65,1.05},{0.53,0.78},{0.70,0.55},{1.25,0.82},
  {1.32,1.07},{1.37,0.58},{1.52,0.85}}],
  {"C","H","H","H","C","H","H","H"}]
```

4.9 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:

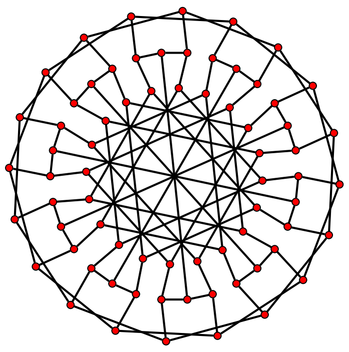


```
ComputedKnot[{{{0.00,0.78,-157.},{-0.27,0.53,-127.},
  {-0.60,-0.02,-113.},{-0.68,-0.39,-83.3},{-0.5,-0.87,-22.},
  {0.5,-0.87,22.},{0.68,-0.39,83.2},{0.56,-0.03,113.},
  {0.28,0.53,127.},{0.00,0.78,157.},{-0.5,0.86603,-142.},
  {-1.,0.,-98.},{-0.68,-0.39,-36.8},{-0.33,-0.50,-6.83},
  {0.32,-0.51,6.53},{0.68,-0.39,36.7},{1.,0.,98.},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

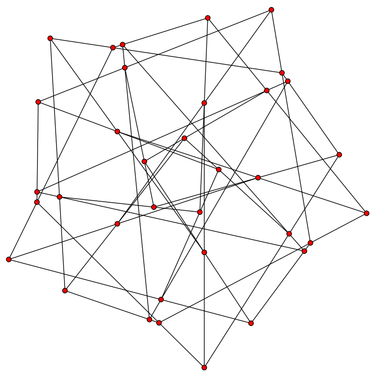
5 Examples

The power of Vega is explained here by two simple examples. The details regarding the first one can be found in [1, 20].

The smallest cubic graph of girth 10 is called a 10-cage. There exist three non-isomorphic 10-cages. One of them is known as the *Balaban graph* [8]. Since it is bipartite it is a Levi graph (incidence graph) of a (v_3) configuration, which we call the *Balaban configuration*. See Figure 3.



(a)



(b)

Figure 3: The Balaban graph (a) and the self-dual Balaban configuration (b).

Not every (v_3) configuration can be drawn with straight lines in a plane. Using the theory from [1] and Vega one can determine rational coordinates of the points of the Balaban configuration that show it can be drawn with straight lines.

The second example is related to the to the *Dyck graph* on 32 vertices. Recently it was extensively studied by R. B. King [10, 11, 12] in connection with a possible infinite carbon molecular structures with negative curvature. Its finite quotient is a regular $\{8,3\}_6$ map on a Riemann surface of genus 3 [3]. The Foster census [3] representation is shown in Figure 4(a). By the same argument as above it gives rise to a (16_3) self-dual configuration which we call the *Dyck configuration*, Figure 4(b). Again, using VEGA, we were easily able to construct a symmetric straight-line drawing of the configuration.

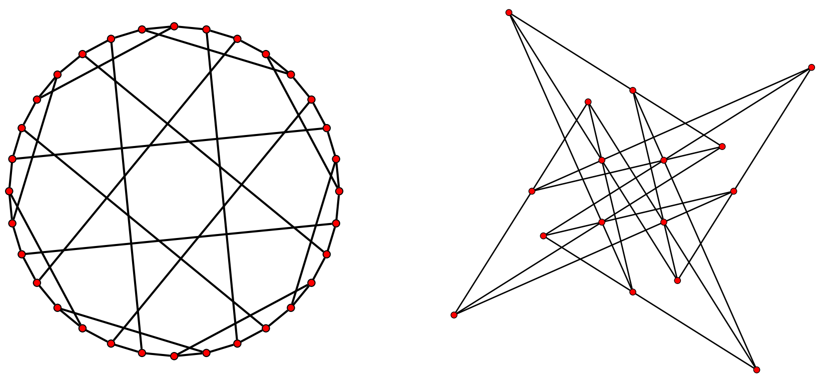

(a) (b)

Figure 4: The Dyck graph (a) and the Dyck configuration (b).

Acknowledgement

The authors would like to thank to Alen Orbanić for his careful reading of the paper.

References

[1] M. Boben, T. Pisanski, *Polycyclic configurations*, submitted.

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

[3] I. Z. Bouwer, W. W. Chernoff, B. Monson, Z. Star, The Foster Census, Charles Babbage Research Centre, 1988.

[4] P. Fowler, D. Manolopoulos, R. Ryan, *Stone-Wales pyracylene transformations of the isomers of C84*, J. Chem. Soc. Chem. Comm. 5 (1992) 408–410.

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

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

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

[8] N. Hartsfield, G. Ringel, Pearls in graph theory, Academic Press, Boston, 1994.

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

[10] R. B. King, *Automorphism groups and spectra of highly symmetrical graphs generating possible carbon and boron nitride structures by leapfrog transformations: the Klein and Dyck graphs*, Match 44 (2001) 237–250.

[11] R. B. King, *Novel highly symmetrical trivalent graphs which lead to negative curvature carbon and boron nitride chemical structures*, Discrete Math. 244 (2002) 203–210.

[12] R. B. King, *Riemann surfaces as descriptors for symmetrical negative curvature carbon and boron nitride structures*, Croat. Chem. Acta 75 (2002) 447–473.

[13] LEDA, `http://www.algorithmic-solutions.com/`.

[14] L. Lovász and K. Vesztergombi, *Geometric representations of graphs*, in: Paul Erdős and his Mathematics, Proc. Conf. Budapest, 1999.

[15] Magma, the Computational Algebra System `http://magma.maths.usyd.edu.au/magma/`.

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

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

[18] MuPad, `http://www.mupad.de/`.

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

[20] T. Pisanski, M. Boben, D. Marušič, A. Orbanić, *The Generalized Balaban Configurations*, submitted.

[21] T. Pisanski, M. Boben and A. Žitnik, Visualization of Graphs and Related Discrete Structures in *Mathematica*, `PrimMath[2001]`, Zagreb, 2001.

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

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

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

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

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

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

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

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

[30] S. Wolfram, The Mathematica book, Cambridge University Press, 1999.