19

PIGALE

Hubert de Fraysseix
CNRS UMR 8557. Paris

Patrice Ossona de Mendez
CNRS UMR 8557. Paris

19.1 Introduction

This chapter gives an overview of the *Public Implementation of a Graph Algorithm Library and Editor* (`Pigale`). `Pigale` integrates a graph algorithm library written in `C++` and a graph editor based on the `Qt`© and `OpenGL`™ libraries. This program runs under `Linux`, `Mac OS X`™ and `Windows`™ platforms. It is particularly intended for academic researchers working on topological graph theory.

`Pigale` is available under `GPL`[1] license and may be downloaded on `sourceforge.net` at `http://pigale.sourceforge.net`. `Pigale` may be used as a library, as a graph editor or as a multi-threaded graph algorithm server.

The GNU General Public License is a free, copyleft license for software and other kinds of works. the GNU General Public License is intended to guarantee your freedom to share

™`OpenGL` is a trademark of Silicon Graphics, Inc.; `Mac OS X` is a trademark of Apple Inc.; `Windows` is a trademark of Microsoft.
[1] `GNU` General Public License.

and change all versions of a program — to make sure it remains free software for all its users (see `http://www.gnu.org/licenses/gpl.html`).

The library is built on an original data structure. This data structure optimizes operations performed on static graphs.

19.1.1 Why GPL?

Free software has the following advantages, which we believe are essential for academic software:

- It increases the exchanges between research centers and facilitates the integration of algorithms originating from several contexts into a coherent framework, thus inducing *de facto* new standards in the concerned field.
- It increases the visibility of the laboratories's skills, thus offering a showcase toward potential industrial partners and allows the development of industrial software based on well-designed license-free libraries.
- It allows to reduce the economic gap between rich and poor countries and contributes to the competitiveness of local laboratories and companies by reducing the cost linked to the acquisition of foreign licenses.
- It allows the users to control the source code of sometimes strategic modules of their projects and suppresses the dramatic dependence on a single software provider, which ties the users to the perennity and the goodwill of a particular actor.

19.1.2 Chapter Organization

The rest of this chapter is organized as follows. Section 19.2 discusses data structures for representing graphs and their embeddings. In Section 19.3, we describe fundamental graph algorithms provided by `Pigale`. The map generators available in `Pigale` are outlined in Section 19.4. In Section 19.5, we present the drawing algorithms supported by `Pigale`. The implementation of `Pigale`, including the graphical interface for creating graphs in `Pigale` is illustrated in Section 19.6. Finally, in Section 19.7, we show an example of use of `Pigale` as a software libary.

19.2 Data Structures

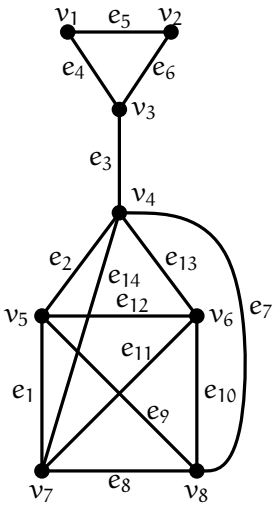
In this section, we present the graph model and data structures we have developed in `Pigale`.

19.2.1 The Topological Quasi-Static Model

`Pigale` provides two main graph data structures, depending on whether one considers dense graphs or sparse ones:

- For dense graphs, a matrix is used, which represents the adjacency relation among vertices or the vertex-edge incidence relation;
- For sparse graphs, either a list of incidences (i.e., a list of all edges with vertex incidences) or lists of adjacencies for the vertices are used.

Although the matrix encoding allows constant-time adjacency testing, it does not allow to list the edges incident to a vertex in constant time per incident edge. Also, this encoding needs space quadratic in the number of vertices. As the `Pigale` software is mainly concerned with topological graph algorithms, particularly traversal-based algorithms, it has been a natural choice to consider list encodings of graphs. On the one hand, we shall allow to input graphs encoded as a list of edge incidences in order to simplify the interface to other software (see Figure 19.1). On the other hand, the internal representation of graphs is tailored to fit the types of topological graph algorithms we mainly consider.



$$\mathrm{L} = \big((5,7),(5,4),(4,3),(1,3),(1,2),(2,3),(4,8),(8,7),(8,5),(8,6),(6,7),(6,5),(6,4),(4,7)\big)$$

Figure 19.1 Encoding of a graph by a list of incidences

Internal graph representation is a major issue for the efficiency of graph algorithms. Although most of the data structures used by graph algorithm libraries are oriented to fully dynamic graphs, thus offering constant-time insertion and deletion operations, the `Pigale` data structure is oriented to *quasi-static* graphs, that is, graphs on which only few modifications are done. Moreover, these modifications mainly correspond to a sequence of additions and (after some computations) of deletions of the added elements. In such a context, it is of particular interest to index vertices by consecutive integer values from 1 to n (where n is the order of the graph) and edges by consecutive integer values from 1 to m (where m is the size of the graph).

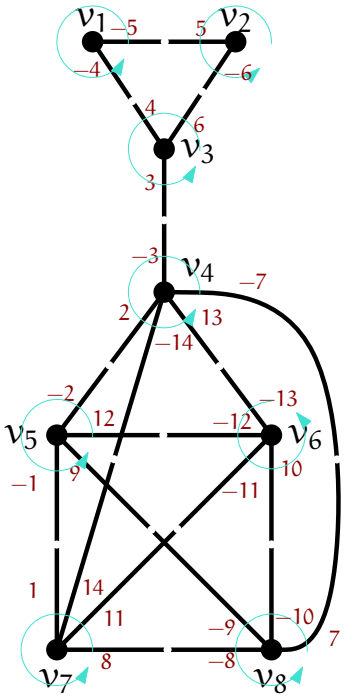
Since `Pigale` is designed to ease the writing of topological graph algorithms, the data structure is based on the mathematical notion of combinatorial map. A *combinatorial map* is a triple (B, τ, σ), where B is a set of half-edges, each called a *brin* (also sometimes called *flag* or *dart*), τ is a fixed point free involution of B whose orbits are the edges of the map, and σ is a permutation of B whose orbits are the vertices of the map.

This combinatorial structure is particularly efficient for map traversals. However, edges and vertices only have an implicit description in this model. This is the reason why `Pigale`'s graph description slightly differs from the one of the combinatorial map. The structure describing maps in `Pigale` is based on the functions shown in Table 19.1, where $V = \{1, \ldots, n\}$ is the index set of the vertices, $E = \{1, \ldots, m\}$ is the index set of the edges, and $B = \{-m, \ldots, -1, 1, \ldots, m\}$ is the index set of the brins.

Note that for technical reasons, the vertex set, the edge set and the brin set are actually $\{0,\dots,n\}$, $\{0,\dots,m\}$ and $\{-m,\dots,m\}$. The operators are extended to 0 with reserved values cir[0] = acir[0] = vin[0] = pbrin[0] = 0. (see Figure 19.2).

Operator	Domain	Description
$-b$	$B \to B$	brin opposite to b ($\tau(b)$)
cir[b]	$B \to B$	brin next to b in circular order ($\sigma(b)$)
acir[b]	$B \to B$	brin before b in circular order ($\sigma^{-1}(b)$)
$\lvert b\rvert$	$B \to E$	edge containing b
vin[b]	$B \to V$	vertex incident to b
e	$E \to B$	first brin of edge e
pbrin[v]	$V \to B$	first brin incident to vertex v

Table 19.1 Functions of the data structure for maps in `Pigale`.



brin #	Cir	Acir	Vin
−14	13	2	4
−13	−12	10	6
−12	−11	−13	6
−11	10	−12	6
−10	−9	7	8
−9	−8	−10	8
−8	7	−9	8
−7	−3	13	4
−6	5	5	2
−5	−4	−4	1
−4	−5	−5	1
−3	2	−7	4
−2	−1	12	5
−1	9	−2	5
1	8	14	7
2	−14	−3	4
3	6	4	3
4	3	6	3
5	−6	−6	2
6	4	3	3
7	−10	−8	8
8	11	1	7
9	12	−1	5
10	−13	−11	6
11	14	8	7
12	−2	9	5
13	−7	−14	4
14	1	11	7

Figure 19.2 Encoding of a graph by a combinatorial map.

19.2.2 Graph Properties

Since in our model vertices, edges and brins are represented by integer values, most of the properties attached to the elements of the graph will be scalar. In order to reduce the slow down of calls to constructors and destructors of complex types, it has been decided to favor scalar properties.

Since the graph structure is a very general abstract one, most algorithms and applications need more or less specific properties to be added to vertices, edges or brins. It appears that class derivation, which is suitable in contexts where a limited number of distinct sets of properties are meaningful, does not work well in our case. This is the reason why we have opted for a more flexible framework in which properties may be added or suppressed dynamically. Then only a few subsets of properties have to be distinguished, the subsets corresponding to coherent views of a graph as a mere *graph* (i.e., a list of edge to vertex incidences), a *topological graph* (where circular orders around the vertices are defined) or a *geometric graph* (where vertices have coordinates, labels, colors, ...), leading to three *logical views* in `Pigale`, namely: `Graph`, `TopologicalGraph`, and `GeometricGraph`, of the set of graph properties stored in a `GraphContainer` data structure.

19.3 Basic Graph Algorithms

In this section, we describe the implementation of several basic graph algorithms in `Pigale`.

19.3.1 Depth-First Search

Depth-First Search (DFS) is central to the planarity algorithm implemented in `Pigale`. It is responsible for a sensible percentage of the execution time. Thus, the optimization of this particular algorithm has strong consequences on the efficiency of other important algorithms.

One of the main characteristics of DFS is that the DFS-tree it builds is traversed several times and that the tree/cotree partition it induces is intensively used in the planarity testing algorithm. For these reasons, it appeared that an efficient optimization stands in the renumbering of the vertices and the edges of the graph using the following scheme (see Figure 19.3):

- the vertices are numbered $1, \ldots, n$ in the order of first discovery by the DFS;
- the tree edges are numbered $1, \ldots, n-1$ in the order of first traversal by the DFS. Precisely, brin i is adjacent to the parent of vertex $(i+1)$ and brin $-i$ is adjacent to vertex $(i+1)$;
- the cotree edges are numbered $n, \ldots, m$ in order opposite to the order in which their low incidences are met by the DFS. The positive brin is incident to the lower vertex according to tree order.

From the above numbering, it follows that a traversal of the edges in DFS order may be simulated using a simple `for(e=1; e<n; e++)` loop. Also, testing if an edge belongs to the tree is performed by a simple `(e<n)` test.

19.3.2 Planarity and Nonplanar Subgraph Exhibition

The linear-time planarity testing algorithm implemented in `Pigale` is based on the characterization by de Fraysseix and Rosenstiehl [FR85, FR82, FR83a, FR83b] and its improvement [FOdMR06, FOdM12, Fra08]. This algorithm is currently the fastest-implemented planarity testing algorithm [BCPD04].

A linear-time algorithm to find a Kuratowski subdivision in a nonplanar graph (see Figure 19.4) has been implemented in `Pigale`, based on a theoretical characterization of DFS cotree-critical graphs [FOdM01a, FOdM02, FOdM03].

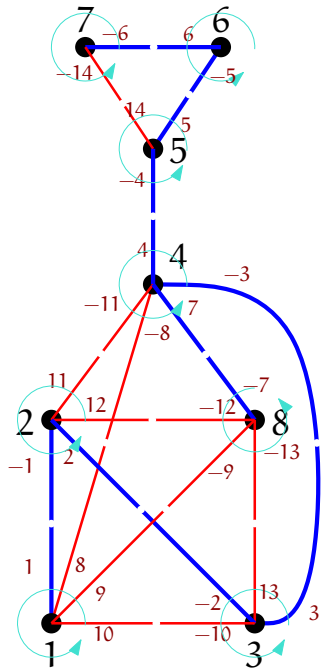


Figure 19.3 DFS numbering of a combinatorial map.

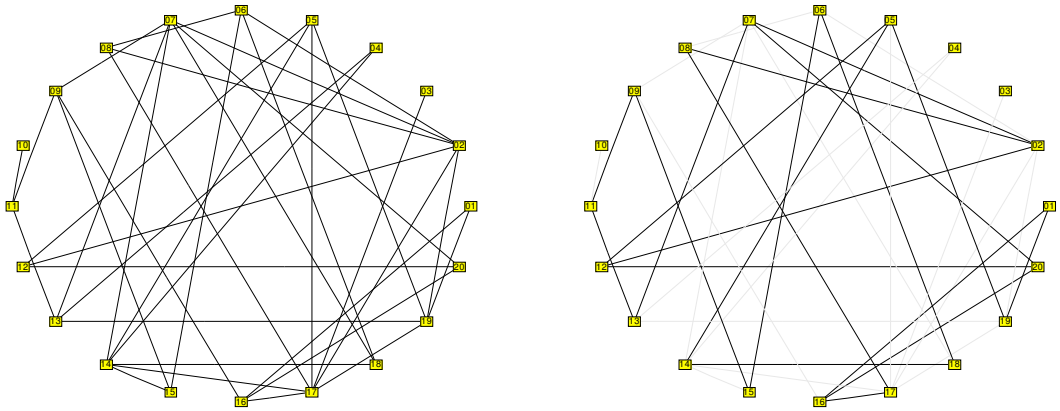


Figure 19.4 A Kuratowski subdivision in a nonplanar graph.

This algorithm relies on the concept of DFS cotree-critical graphs, which is a by-product of our planarity testing algorithms. Roughly speaking, a DFS cotree-critical graph is a simple graph of minimum degree 3 having a DFS tree, such that any nontree (i.e., cotree) edge is critical, in the sense that its deletion would lead to a planar graph. A first study of DFS cotree-critical graphs appeared in [FR83a], where it is proved that a DFS cotree-critical graph is either isomorphic to K_5 or includes a subdivision of $K_{3,3}$ and no subdivision of K_5.

The algorithm consists of two steps:

1. Extraction of a DFS cotree-critical subgraph by a case analysis algorithm; and
2. Extraction of a Kuratowski subdivision from the DFS cotree-critical subgraph.

Step 2 is performed by an algorithm whose simplicity contrasts with the complexity of its theoretical justification (which relies on the full characterization of DFS cotree-critical graphs proved in [FOdM03]. This algorithm roughly works as follows:

- It first computes the set of the critical edges of a graph, using the property that a tree edge is critical if and only if it belongs to a fundamental cycle of length 4 of some cotree edge to which it is not adjacent.
- Then, three pairwise non-adjacent non-critical edges are found to complete a Kuratowski subdivision isomorphic to $K_{3,3}$.

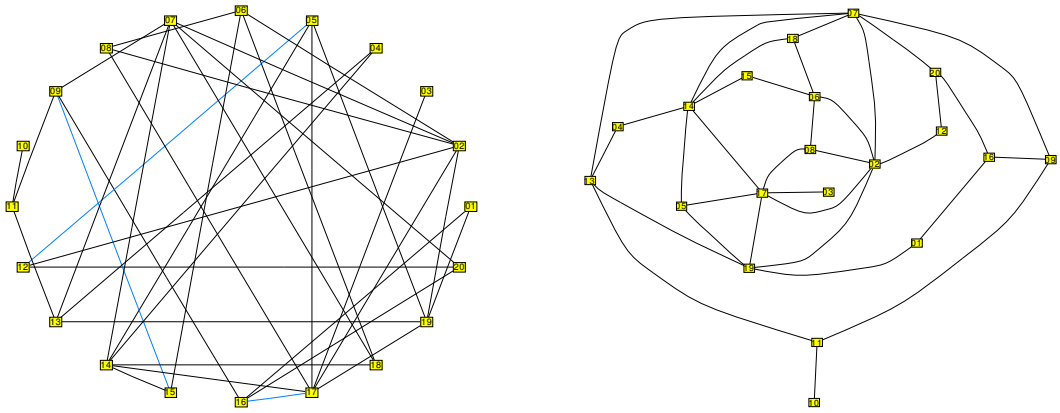


Figure 19.5 Finding a minimal subset of nonplanar edges. On the right is a Bezier drawing of the planar graph obtained after deletion of the computed set of nonplanar edges.

This algorithm is the central routine of a heuristic for exhibiting an inclusion-minimal set of edges whose deletion ensures the planarity of the graph (see Figure 19.5).

19.3.3 Connectivity Tests

Based on properties of regular orientations of planar graphs, `Pigale` offers a linear-time algorithm to test whether a planar graph is 3-connected and a linear-time algorithm to test whether a maximal planar graph is 4-connected [FOdM01b, FOdM04, FOdM01d]. The study of graphs by means of special orientations is relatively recent. For instance, bipolar orientations have become a basic tool in many graph drawing problems [OdM94, FOdMR95].

Constrained orientations (i.e., orientations with bounded indegrees) lead to new characterizations of connectivity for planar undirected graphs. Although standard 3-connectivity testing algorithms for planar graphs are heavily related to planarity testing algorithms (see [HT73, Tar74] and PQ-tree algorithms), the algorithm in `Pigale` assumes that the input graph is already embedded in the plane so the problem drastically reduces to the acyclicity testing of a particular orientation. Concerning the 4-connectivity testing of a maximal planar graph, the use of an indegree bounded orientation was already used in [CE91] to enumerate triangles. In `Pigale`, the use of a specific orientation allows to further simplify the algorithm. The 4-connectivity test itself also reduces to an acyclicity test. It should be noted that no special data structure is used for these algorithms since in the planar case the acyclicity of an orientation can be efficiently tested using a dual topological sort.

19.3.4 Augmentation of Planar Graphs

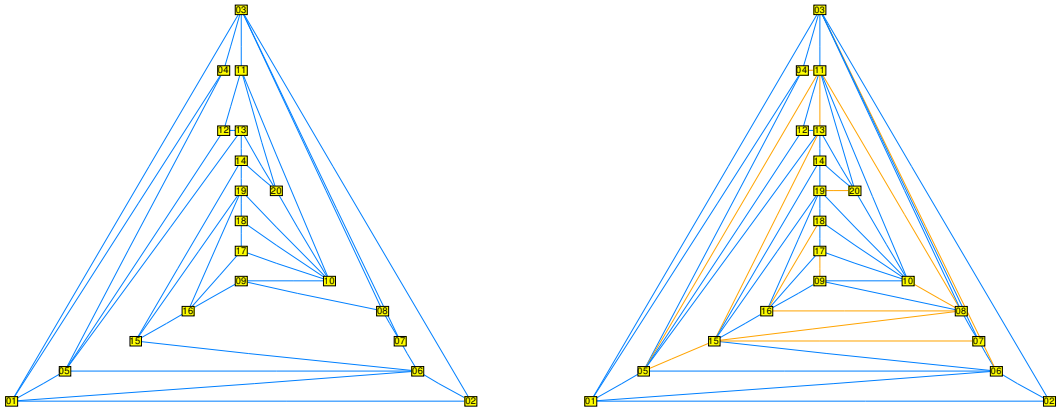


Figure 19.6 Augmentation of a 3-connected planar graph.

Constrained orientations have many applications [FOdM94a, FMOdMR95] (see also above). These orientations are a basic tool in solving combinatorial problems that preserve topological properties [FOdM01d]. Planar augmentations are a simple example of such problems.

Augmentation problems are concerned with the addition of dummy edges to a graph in order to obtain some connectivity or maximality properties. For instance, the problem of finding the minimum number of edges to augment a graph to a biconnected graph has been solved in [ET76]. If the original graph is planar and if it is required to preserve the planarity, the problem is NP-complete [Kan93]. Triangulating a biconnected graph while minimizing the maximum degree has also been proved to be an NP-complete problem.

`Pigale` offers several optimal augmentation algorithms, including a linear-time algorithm for augmenting a 3-connected planar graph to a maximal planar graph (see Figure 19.6) that increases the degree of any vertex of the graph by no more than 6 (which is optimal) [FOdM95, FOdM94b].

19.3.5 Graph Symmetry and Clustering

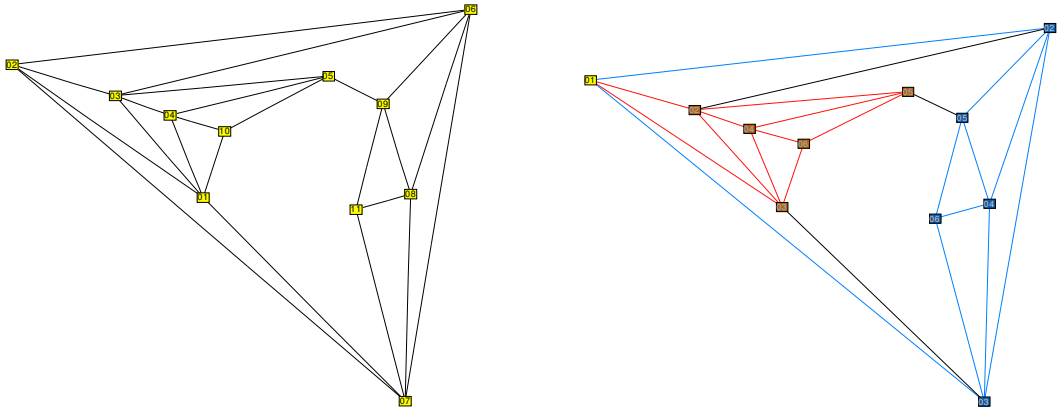


Figure 19.7 Finding a symmetry of a graph.

Based on spectral analysis [FK92], `Pigale` offers a heuristic to find symmetries in a general simple graph (planar or not) [Fra99, FOdM06] (see Figure 19.7). These symmetries

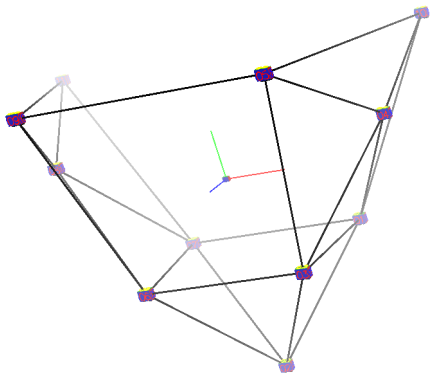


Figure 19.8 3D view of the graph displaying the symmetry.

may actually be viewed in the 3D drawing built from the spectral analysis of the graph (see Figure 19.8).

Using spectral analysis, Babai proved in 1978 that the abstract automorphism group of any multigraph G having s distinct eigenvalues with respective multiplicities $m_1, m_2, \ldots, m_s$ is a subgroup of $\omega(m_1) \oplus \omega(m_2) \oplus \ldots \oplus \omega(m_s)$, where $\omega(m)$ denotes the real orthogonal group of dimension m [Bab78]. As a consequence, if all the eigenvalues of G are simple, the only automorphisms of G are involutions.

Some years before, Mani proved that every triconnected planar graph G can be realized as the 1-skeleton of a convex polytope P in $\mathbb{R}^3$ such that all automorphisms of G are induced by isometries of P [Man71]. One non trivial consequence of this result is that the automorphism group $\mathrm{Aut}(G)$ of a triconnected planar graph G has a chain of normal subgroups $\mathrm{Aut}(G) = G_0 \triangleright G_1 \triangleright \ldots \triangleright G_m = 1$, where each quotient G_i/G_{i-1} is either cyclic, or isomorphic to a symmetric group or A_5.

The result of Mani may be expressed in a weaker form: any triconnected planar graph has an embedding f into $\mathbb{R}^3$, such that $\mathrm{Aut}(G)$ is the group of isometries of $\mathbb{R}^3$ globally preserving the point set $P = f(V(G))$, that we shall denote by $\omega(3, P)$.

These two results are generalized in [FOdM06], where it is proved that every twin-free loopless multigraph G has some *regular embedding*, that is, some embedding $f : V(G) \to \mathbb{R}^k$ such that $\mathrm{Aut}(G)$ is isomorphic to the group $\omega(k, f(V(G)))$ of isometries of $\mathbb{R}^k$ globally preserving $f(V(G))$, and that this group might be expressed as a subgroup of a group sum relying on spectral considerations. This result is proved using techniques similar to those used in the symmetry detection heuristic presented in [Fra99]. The problem of finding regular embeddings is reduced to the one of finding metrics on the vertex set of the multigraph that define *Euclidean*, *reconstructing*, and *commuting* distance matrices, which may be built from particular symmetric real matrices with 0 on the diagonal (the commuting reconstructing predistances).

Several such distances have been implemented in `Pigale` (see Table 19.2 and Figure 19.9).

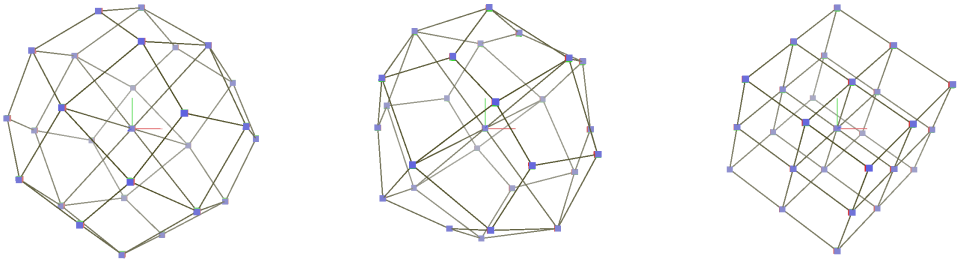


Figure 19.9 Embedding a cube in $\mathbb{R}^{n-1}$ using different distances (from left to right): Czekanovski-Dice, translated adjacency, and Laplacian.

Czekanovski-Dice distance	$\text{dist}^2(i,j) = 1 - \frac{\lvert N(i) \cap N(j) \rvert}{\lvert N(i) \rvert + \lvert N(j) \rvert}$
Oriented distance	$\text{dist}^2(i,j) = 1 - \frac{\lvert N^-(i) \cap N^-(j) \rvert}{\lvert N^-(i) \rvert + \lvert N^-(j) \rvert} - \frac{\lvert N^+(i) \cap N^+(j) \rvert}{\lvert N^+(i) \rvert + \lvert N^+(j) \rvert}$
Adjacency distance (not Euclidean)	$\text{dist}^2(i,j) = \begin{cases} 0, & \text{if } i = j \text{ or } i \text{ and } j \text{ are adjacent} \\ 1, & \text{otherwise} \end{cases}$
Translated adjacency distance	$\text{dist}^2(i,j) = \begin{cases} 0, & \text{if } i = j \\ 1 - \frac{2}{n}, & \text{if } i \text{ and } j \text{ are adjacent} \\ 1, & \text{otherwise} \end{cases}$
Bisection distance	$\text{dist}^2(i,j) = \begin{cases} 0, & \text{if } i = j \\ 1 - \frac{2}{d(i)+d(j)+2}, & \text{if } i \text{ and } j \text{ are adjacent} \\ 1, & \text{otherwise} \end{cases}$
$\mathbb{R}^2$ distance	$\text{dist}^2(i,j) = (x(i) - x(j))^2 + (y(i) - y(j))^2$
Laplacian distance	$\text{dist}^2(i,j) = \begin{cases} 0, & \text{if } i = j \\ 2n - d(i) - d(j), & \text{if } i \text{ and } j \text{ are adjacent} \\ 2n - d(i) - d(j) + 2, & \text{otherwise} \end{cases}$
Q distance	$\text{dist}^2(i,j) = \begin{cases} 0, & \text{if } i = j \\ 1, & \text{if } i \text{ and } j \text{ are non adjacent} \\ 1 - \frac{1}{\sqrt{d(i)d(j)}}, & \text{otherwise} \end{cases}$

Table 19.2 Choice of distances for the spectral analysis/embedding in `Pigale`; $N(i)$ (resp. $N^-(i), N^+(i)$) denotes the set of the neighbors (resp. in-neighbors, out-neighbors) of vertex i and $d(i) = |N(i)|$ denotes the degree of vertex i.

19.4 Random Map Generators

Several polynomial-time random planar map uniform generators have been implemented by Gilles Schaeffer in `Pigale` [Sch99]:

- planar maps (connected, 2-connected, or 3-connected),
- planar cubic maps (2-connected, 2-connected bipartite, 3-connected, 3-connected bipartite, or dual-4-connected),
- planar 4-regular maps (2-connected, 3-connected, or bipartite),
- planar bipartite maps.

Also, linear-time uniform generators of outerplanar maps have been implemented by Nicolas Bonichon [BGH03].

The implementation of these algorithms in `Pigale` uses the uniform pseudo-random number generator of Matsumoto and Nishimura [MN98]. This pseudo-random number generator is also used to generate random graphs where edges are independently included with fixed probabilities (Erdős-Rényi model).

19.5 Graph Drawing Algorithms

This section is devoted to the graph drawing algorithms provided by `Pigale`.

19.5.1 Planar Straight-Line Grid Drawings

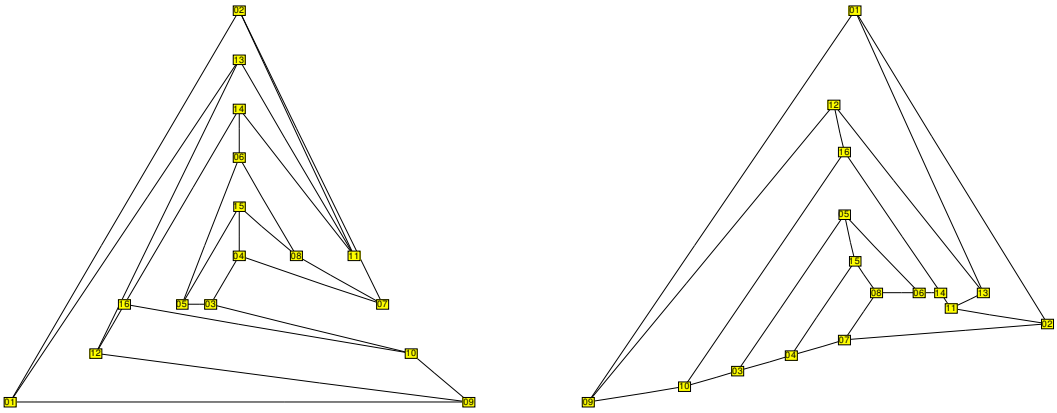


Figure 19.10 Fraysseix Pach Pollack (with edge augmentation) and Schnyder (using vertex augmentation).

`Pigale` includes several linear-time planar straight-line drawing algorithms for simple planar graphs, including the Fraysseix-Pach-Pollack algorithm [FPP88, FPP90] and Schnyder's algorithm [Sch89, Sch90] (see Figure 19.10). Some bounds and conjectures on the size of straight-line drawings may be found in [FOdM01c].

A linear-time compact convex drawing algorithm for 3-connected planar graphs [BFM04], as well as a compact polyline drawing for simple planar graphs [BLSM02], have been added by Nicolas Bonichon (see Figure 19.11).

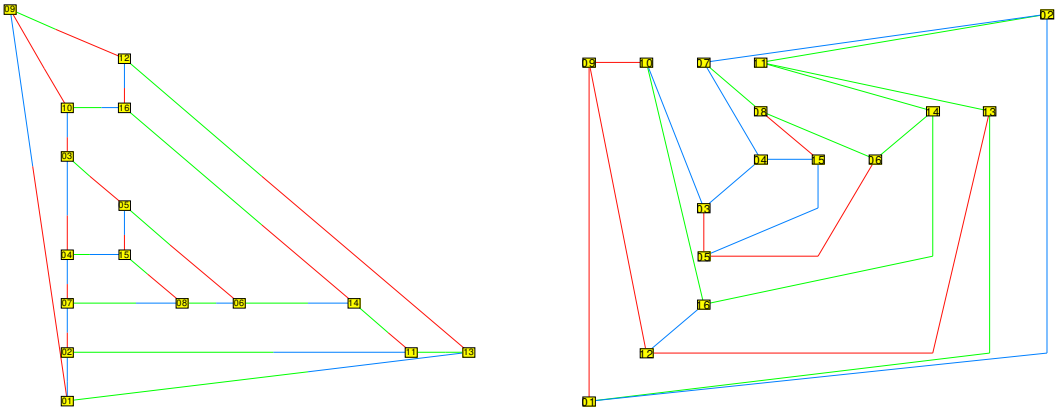


Figure 19.11 Compact convex drawing and compact polyline drawing.

19.5.2 Spring Embedders

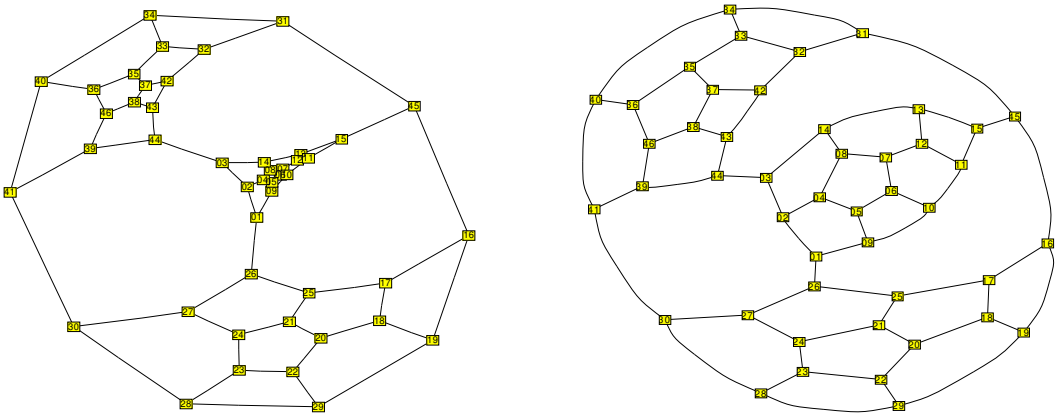


Figure 19.12 Tutte drawing and a drawing with curved edges based on a spring embedder initialized with Tutte drawing.

The Tutte drawing of a 3-connected planar graph [Tut60, Tut63] is implemented in `Pigale` and usually represents a good starting drawing for a spring embedder drawing algorithm (see Figure 19.12). Our spring embedder has the particularity to preserve an initial geometric map (relative positions and crossings) of a (nonplanar) graph.

19.5.3 Visibility Drawing and Variants

Visibility and rectilinear drawings [RT86, TT86] have received much attention because of their good readability (see Figure 19.13). All the algorithms mentioned in this section are linear-time algorithms. With the exception of the Polrec algorithm, all the representations described in this section concern simple planar graphs. The area of the drawing may be further reduced by allowing horizontal and vertical visibility, as in an algorithm proposed by de Fraysseix, Pach, and Pollack (see Figure 19.14).

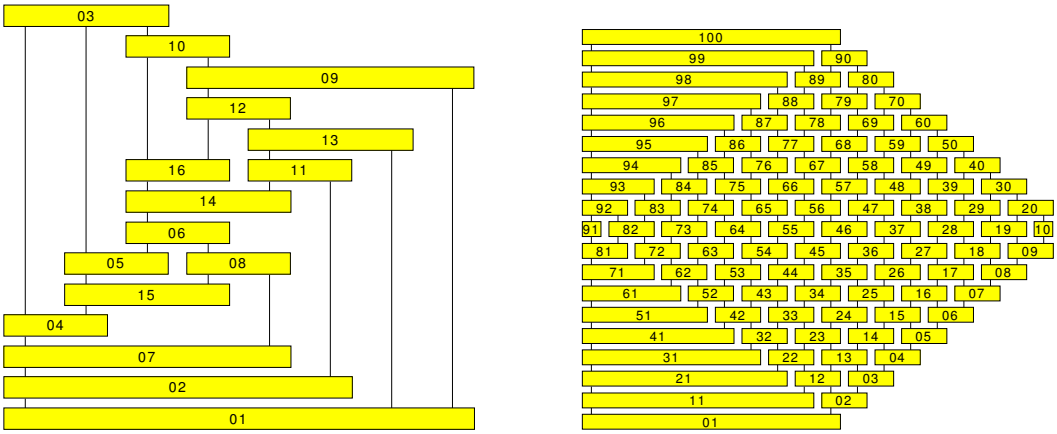


Figure 19.13 Visibility drawings. The drawing on the right is within a 10×10 grid.

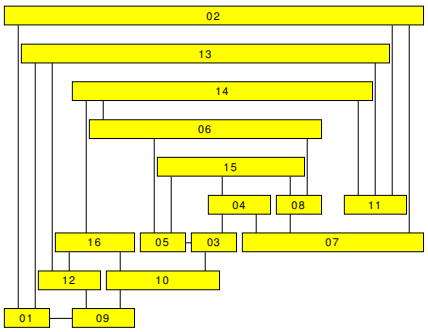


Figure 19.14 Rectilinear drawing constructed by an algorithm by de Fraysseix, Pach, and Pollack.

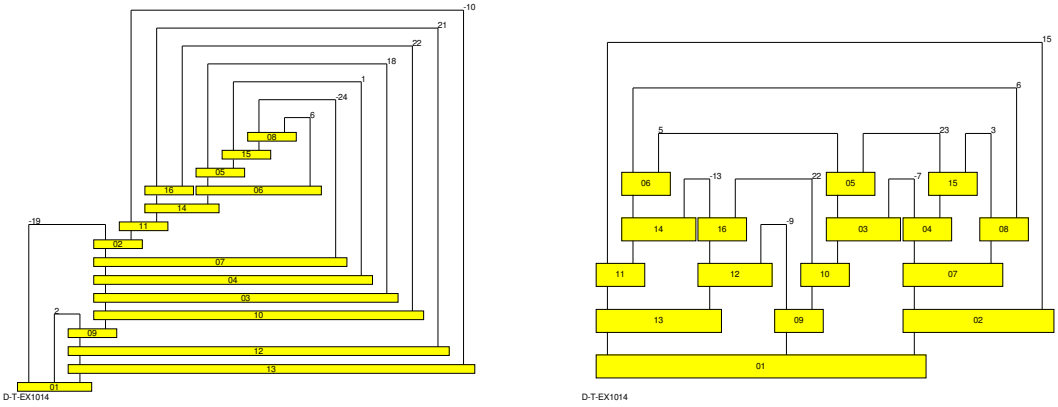


Figure 19.15 Polrec drawings based on a DFS-tree and a BFS-tree, respectively.

The Polrec algorithm produces a drawing where vertices are represented by boxes, a tree is represented using straight-line vertical segments and cotree-edges are represented by U-shaped polylines (see Figure 19.15). Such a representation can be used for non-simple nonplanar graphs with loops (see Figure 19.16).

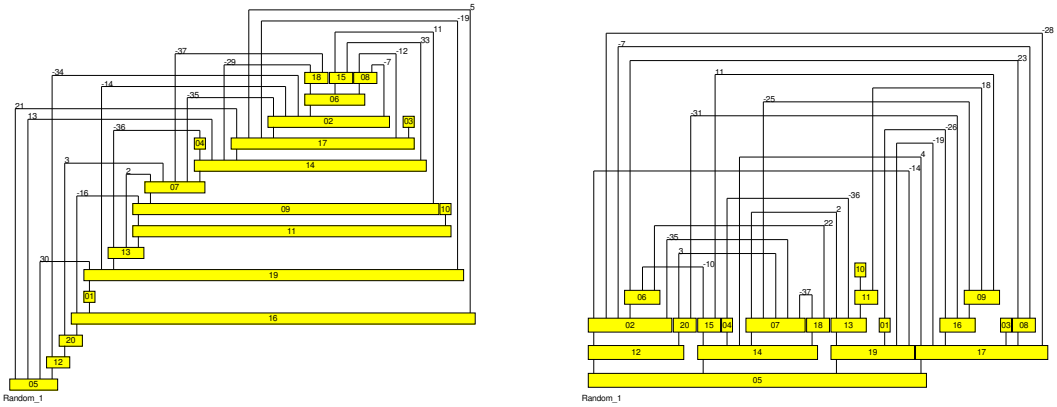


Figure 19.16 DFS-based and BFS-based Polrec representations of a nonplanar graph.

19.5.4 Contact Drawings

An emerging representation of graphs concerns contact and intersection representations. All the algorithms mentioned in this section are linear-time algorithms and concern simple planar graphs.

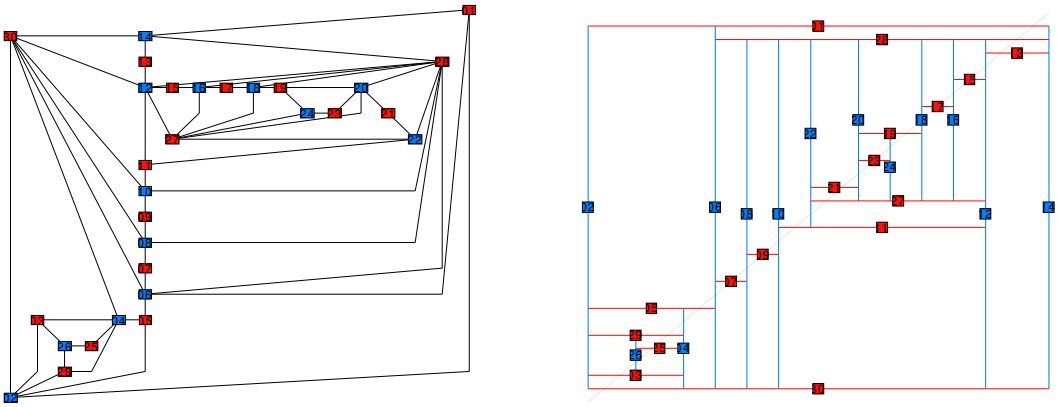


Figure 19.17 Representation of a bipartite planar graph by contact of segments.

`Pigale` offers a representation of bipartite planar graphs as contact graphs of horizontal and vertical straight line segments [FOdMP91, FOdMP95] (see Figure 19.17), as well as a representation of planar graphs by contacts of T-shaped vertices or by contacts of triangles [FOdMR94, FOdMR97] (see Figure 19.18). The generalization of the representation of planar graphs by contact of triangles to linear hypergraphs [FOdMR08] has not been implemented yet.

19.5.5 Spectral Drawings in $\mathbb{R}^n$

As mentioned in Section 19.3.5, spectral analysis may be used to generate 3D visualizations of (nonplanar) graphs in polynomial time (see Figure 19.19). The time complexity of the algorithm derives from the complexity of the computation of the eigenvalues of an $n \times n$ matrix, where n is the number of vertices of the represented graph.

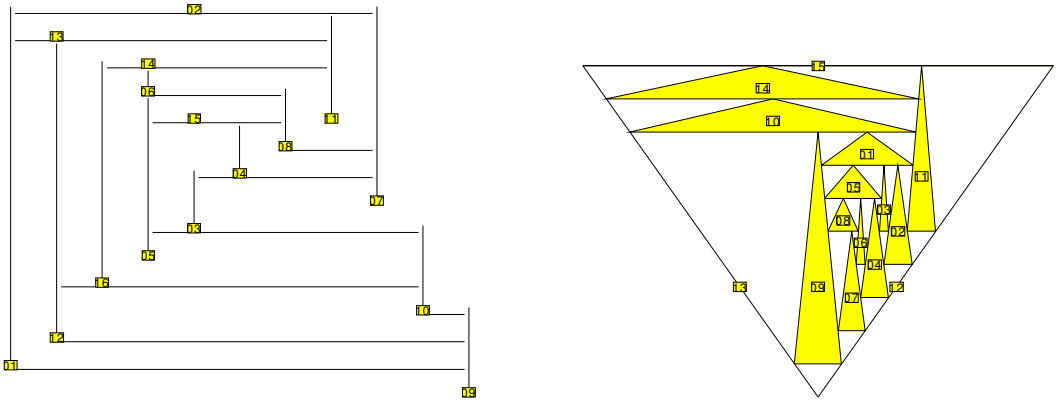


Figure 19.18 Contacts of Ts and contacts of triangles.

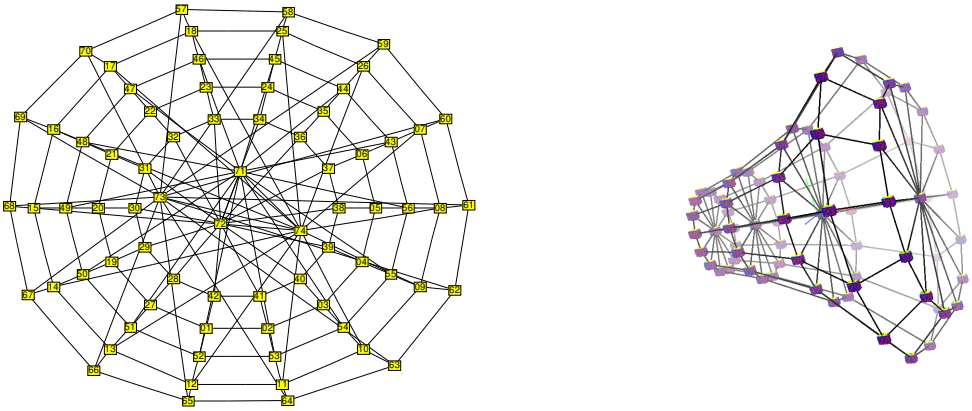


Figure 19.19 3D embedding of a nonplanar graph with 3D symmetries.

19.6 Implementation

19.6.1 User Interface

`Pigale` provides a graph editor that allows the user to load, save or generate graphs, to edit them, to check the properties of the graph (automatically displayed by the program), to perform several transformations (augmentations, orientations, computation of duals, etc.), and to compute representations of the graph.

While mouse-editing a graph, a user can add, delete, contract, bisect, orient, reorient, unorient, and color edges, and can set their width; the user can also add, move, delete, and color vertices, and can put numerical labels on them.

19.6.2 File Storage

We use a general proprietary format, called TGF. A TGF file contains records, here corresponding to graphs. Each record consisting of a variable number of fields. One of its main advantage is that we can write and read any complex data structure. But it is dependent of the processor type (e.g., big-endian or little-endian).

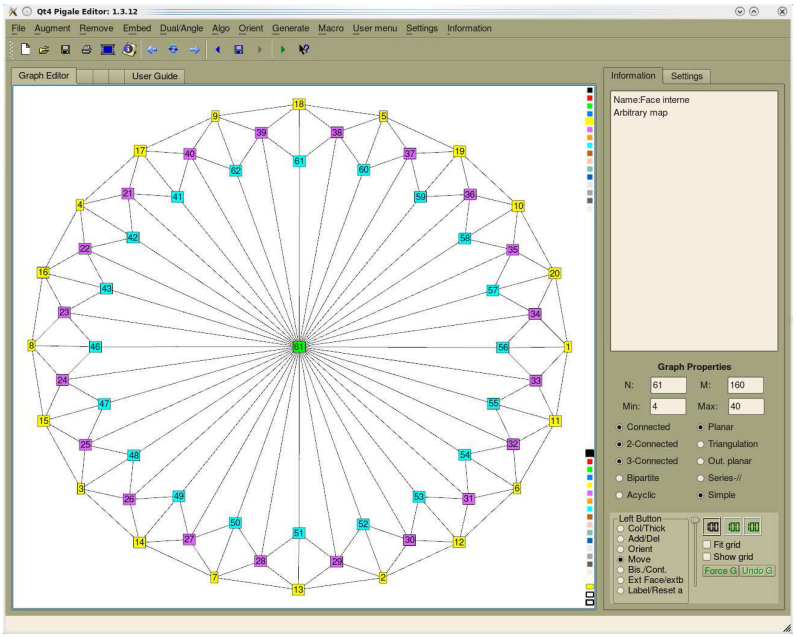


Figure 19.20 `Pigale` editing window.

We have partially implemented the GRAPHML file format (cf. [BEH+02] and the web site `http://graphml.graphdrawing.org/`), which is now the only way to add text labels to the vertices.

We use a very simple ASCII file structure to store graphs. For example, the following file defines a graph, called Triangle, with three vertices, labeled 10, 20, 30, and three edges. The first 0 on the last line indicates the end of the list of edges. The second zero indicates the end of the graph data.

```
PIG:0 Triangle
10 20
20 30
30 10
0 0
```

19.6.3 Macro Recording

One can record any number of functions from the menus into macros, which can be saved as text files.

A macro can be repeated any number of times (possibly until the user will press the ESC key). If the first record of the macro is not a call to a graph generator, the macro will start loading the next graph of the current file.

Macros can be used to develop and benchmark algorithms and to test conjectures.

19.6.4 Multi-Threaded Server

The `Pigale` editor may be put in server mode, which allows the editor to be controlled by a client application. A simple program `client` is provided as an example of how to communicate with the server. The client reads its instructions from `stdin` so that it should

not be difficult for applications to communicate with the server. However, it is not difficult to write, for instance, a web server that acts as a front end to `Pigale`.

19.7 Interfacing with PIGALE

As mentioned in its name, `Pigale` is not only an editor, but also a library. Nearly all the algorithms in `Pigale` may be run in a non-graphic context through a library call.

An example of a simple C++ program using `Pigale` library is given below.

```
#include <Pigale.h>

int main ()
{
  GraphContainer GC; // defined in TAXI/graph.h
  // GC is the object that will contain all the information of a graph.
  int n = 4; // n =  number of vertices [1,n]
  int m = 5; // m = number of edges     [1,m]

  GC.setsize(n,m);  // defines the size of the container
  /*
   - a tvertex v is an integer v(): 1 <= v() <= n = GC.nv()
   - a tedge   e is an integer e(): 1 <= e() <= m = GC.ne()
   - a tedge   e is composed of 2 tbrin b0,b1 equal to e() and -e()
   tvertex, tedge, tbrin behave like integers in many respects
  */

  Prop<tvertex> vin(GC.Set(tbrin()),PROP_VIN);
  // vin is an array  of tbrin whose values are tvertex.

  // Create the edges: each edge (tedge) is incident to 2 vertices (tvertex)
  vin[1] = 1; vin[-1] = 2;  // edge 1 is incident to vertices 1 and 2
  vin[2] = 1; vin[-2] = 3;
  vin[3] = 2; vin[-3] = 3;
  vin[4] = 3; vin[-4] = 4;
  vin[5] = 2; vin[-5] = 4;

  // create a topological graph access
  TopologicalGraph G(GC); // defined in TAXI/graphs.h

  // print the number of vertices and edges
  cout << "Nodes: " << G.nv() << "\tEdges: " << G.ne()<< endl;

  // print the edges (if e is a tedge, e() is the int that represents it)
  cout << "Edges:" << endl;
  for(tedge e = 1; e <= G.ne();e++)
    cout << e() << " = [" << G.vin[e] << "," << G.vin[-e] << "]" <<endl;

  // For planarity test, graphs should be LOOPLESS. You can remove loops:
  // int nloops = RemoveLoops();

  // Compute a planar embedding or return -1
  if(G.Planarity() == 0)
    {cout << "not planar" << endl; return -1;}
```

```
// At each vertex v  there is a tbrin G.pbrin[v] incident to it:
//          G.vin[G.pbrin[v]] = v;
// So we can print the planar map, that is the cirular order of
// half edges around each  vertex.
cout << "Map (half edges):"<<endl;
for(tvertex v = 1; v <= G.nv() ; v++)
  {cout << v() <<"  -> ";
    tbrin first = G.pbrin[v];
    tbrin b = first;
    do
      {cout << b() << " ";
      }
    while((b = G.cir[b]) != first);
    cout << endl;
  }
// Or you could print the circular order of vertices aroud each vertex
cout << "Map (vertices):"<<endl;
for(tvertex v = 1; v <= G.nv() ; v++)
  {cout << v() <<"  -> ";
    tbrin first = G.pbrin[v];
    tbrin b = first;
    do
      {cout << G.vin[-b]() << " ";
      }
    while((b = G.cir[b]) != first);
    cout << endl;
  }
return 0;
}
```

References

[Bab78] L. Babai. Automorphism group and category of cospectral graphs. *Acta Math. Acad. Sci. Hung.*, 31:295–306, 1978.

[BCPD04] J. M. Boyer, P. F. Cortese, M. Patrignani, and G. Di Battista. Stop minding your P's and Q's: implementing fast and simple DFS-based planarity and embedding algorithm. In *Graph Drawing*, volume 2912 of *Lecture Notes in Computer Science*, pages 25–36. Springer, 2004.

[BEH+02] U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall. GraphML progress report: Structural layer proposal. In Springer-Verlag, editor, *Proc. 9th Intl. Symp. Graph Drawing (GD '01)*, volume 2265, pages 501–512, 2002.

[BFM04] N. Bonichon, S. Felsner, and M. Mosbah. Convex drawings of 3-connected planar graphs (extended abstract). In J. Pach, editor, *Graph Drawing 2004*, volume 3383 of *Lecture Notes in Computer Science*, pages 60–70. Springer Verlag, 2004.

[BGH03] N. Bonichon, C. Gavoille, and N. Hanusse. Canonical decomposition of outerplanar maps and application to enumeration, coding and generation. In Springer-Verlag, editor, *29th International Workshop, Graph-Theoretic Concepts in Computer Science (WG)*, volume 2880 of *Lecture Notes in Computer Science*, pages 81–92, 2003.

[BLSM02] N. Bonichon, B. Le Saëc, and M. Mosbah. Optimal area algorithm for planar polyline drawings. In Springer-Verlag, editor, *28th International Workshop, Graph-Theoretic Concepts in Computer Science (WG)*, volume 2573 of *Lecture Notes in Computer Science*, pages 35–46, 2002.

[CE91] M. Chrobak and D. Eppstein. Planar orientations with low out-degree and compaction of adjacency matrices. *Theoret. Comput. Sci.*, 86:243–266, 1991.

[ET76] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM J. Comput.*, 5:653–665, 1976.

[FK92] H. de Fraysseix and P. Kuntz. Pagination of large scale networks. *Algorithms review*, 2(3):105–112, 1992.

[FMOdMR95] H. de Fraysseix, T. Matsumoto, P. Ossona de Mendez, and P. Rosenstiehl. Regular Orientations and Graph Drawing. In *Third Slovenian International Conference in Graph Theory*, pages 12–13, 1995. abstract.

[FOdM94a] H. de Fraysseix and P. Ossona de Mendez. On regular orientations. In *Prague Midsummer Combinatorial Workshop*, pages 9–13, 1994. Abstract.

[FOdM94b] H. de Fraysseix and P. Ossona de Mendez. Some augmentation problems. In *Effiziente Algorithmen*, volume 34/1994, page 11, 1994. Abstract.

[FOdM95] H. de Fraysseix and P. Ossona de Mendez. Regular orientations, arboricity and augmentation. In *DIMACS International Workshop, Graph Drawing 94*, volume 894 of *Lecture Notes in Computer Science*, pages 111–118, 1995.

[FOdM01a] H. de Fraysseix and P. Ossona de Mendez. An algorithm to find a Kuratowski subdivision in DFS cotree critical graphs. In Edy Try Baskoro,

editor, *Proceedings of the Twelfth Australasian Workshop on Combinatorial Algorithms (AWOCA 2000)*, pages 98–105, Indonesia, 2001. Institut Teknologi Bandung.

[FOdM01b] H. de Fraysseix and P. Ossona de Mendez. Connectivity of planar graphs. *Journal of Graph Algorithms and Applications*, 5(5):93–105, 2001.

[FOdM01c] H. de Fraysseix and P. Ossona de Mendez. Lower bounds on sets supporting Fáry drawings. In O. Pangrac, editor, *Graph Theory Day V*, volume 2001-539 of *KAM Series*, pages 35–37, 2001.

[FOdM01d] H. de Fraysseix and P. Ossona de Mendez. On topological aspects of orientations. *Discrete Mathematics*, 229(1-3):57–72, 2001.

[FOdM02] H. de Fraysseix and P. Ossona de Mendez. A characterization of DFS cotree critical graphs. In *Graph Drawing*, volume 2265 of *Lecture notes in Computer Science*, pages 84–95, 2002.

[FOdM03] H. de Fraysseix and P. Ossona de Mendez. On cotree-critical and DFS cotree-critical graphs. *Journal of Graph Algorithms and Applications*, 7(4):411–427, 2003.

[FOdM04] H. de Fraysseix and P. Ossona de Mendez. Connectivity of planar graphs. In *Graphs Algorithms and Applications 2*. World Scientific, 2004.

[FOdM06] H. de Fraysseix and P. Ossona de Mendez. Regular embeddings of multigraphs. In M. Klazar, J. Kratochvil, M. Loebl, J. Matousek, R. Thomas, and P. Valtr, editors, *Topics in Discrete Mathematics*, volume 26 of *Algorithms and Combinatorics*, pages 553–563. Springer-Verlag, 2006. Dedicated to Jarik Nešetřil on the occasion of his 60th birthday.

[FOdM12] H. de Fraysseix and P. Ossona de Mendez. Planarity and Trémaux trees. *European Journal of Combinatorics*, 33(3):279–293, 2012.

[FOdMP91] H. de Fraysseix, P. Ossona de Mendez, and J. Pach. Representation of planar graphs by segments. *Intuitive Geometry*, 63:109–117, 1991.

[FOdMP95] H. de Fraysseix, P. Ossona de Mendez, and J. Pach. A left-first search algorithm for planar graphs. *Discrete Computational Geometry*, 13:459–468, 1995.

[FOdMR94] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. On triangle contact graphs. *Combinatorics, Probability and Computing*, 3:233–246, 1994.

[FOdMR95] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Bipolar orientations revisited. *Discrete Applied Mathematics*, 56:157–179, 1995.

[FOdMR97] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. On triangle contact graphs. In *Combinatorics, Geometry and Probability: A Tribute to Paul Erdős*, pages 165–178. Cambridge University Press, 1997.

[FOdMR06] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Depth-first search and planarity. *International Journal of Foundations of Computer Science*, 17(5):1017–1029, 2006. Special Issue on Graph Drawing.

[FOdMR08] H. de Fraysseix, P. Ossona de Mendez, and P. Rosenstiehl. Representation of planar hypergraphs by contacts of triangles. In *Proceedings of Graph Drawing 2007*, volume 4875/2008 of *Lecture Notes in Computer Science*, pages 125–136. Springer, 2008.

[FPP88] H. de Fraysseix, J. Pach, and R. Pollack. Small sets supporting Fary embeddings of planar graphs. In *Twentieth Annual ACM Symposium on Theory of Computing*, pages 426–433, 1988.

[FPP90] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1990.

[FR82] H. de Fraysseix and P. Rosenstiehl. A depth-first search characterization of planarity. *Annals of Discrete Mathematics*, 13:75–80, 1982.

[FR83a] H. de Fraysseix and P. Rosenstiehl. A discriminatory theorem of Kuratowski subgraphs. In J. W. Kennedy, M. Borowiecki, and M. M. Sysło, editors, *Graph Theory, Łagów 1981*, volume 1018 of *Lecture Notes in Mathematics*, pages 214–222. Springer-Verlag, 1983. Conference dedicated to the memory of Kazimierz Kuratowski.

[FR83b] H. de Fraysseix and P. Rosenstiehl. Système de référence de Trémaux d'une représentation plane d'un graphe planaire. *Annals of Discrete Mathematics*, 17:293–302, 1983.

[FR85] H. de Fraysseix and P. Rosenstiehl. A characterization of planar graphs by Trémaux orders. *Combinatorica*, 5(2):127–135, 1985.

[Fra99] H. de Fraysseix. An heuristic for graph symmetry detection. In J. Kratochvíl, editor, *Graph Drawing*, volume 1731 of *Lecture Notes in Computer Science*, pages 276–285. Springer, 1999.

[Fra08] H. de Fraysseix. Trémaux trees and planarity. In P. Ossona de Mendez, M. Pocchiola, D. Poulalhon, J.L. Ramírez Alfonsín, and G. Schaeffer, editors, *The International Conference on Topological and Geometric Graph Theory*, volume 31 of *Electronic Notes in Discrete Mathematics*, pages 169–180. Elsevier, 2008.

[HT73] J. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM J. Comput.*, 2(3):135–158, 1973.

[Kan93] G. Kant. *Algorithms for Drawing Planar Graphs*. PhD thesis, Dept. Comput. Sci., Univ. Utrecht, Utrecht, Netherlands, 1993.

[Man71] P. Mani. Automorphismen von polyedrischen Graphen. *Mathematische Annalen*, 192:279–303, 1971.

[MN98] M. Matsumoto and T. Nishimura. Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Transactions on Modeling and Computer Simulation*, 8(1):3–30, 1998.

[OdM94] P. Ossona de Mendez. *Orientations bipolaires*. PhD thesis, Ecole des Hautes Etudes en Sciences Sociales, Paris, 1994.

[RT86] P. Rosenstiehl and R. E. Tarjan. Rectilinear planar layout and bipolar orientation of planar graphs. *Discrete and Computational Geometry*, 1:343–353, 1986.

[Sch89] W. Schnyder. Planar graphs and poset dimension. *Order*, 5:323–343, 1989.

[Sch90] W. Schnyder. Embedding planar graphs in the grid. In *First ACM-SIAM Symposium on Discrete Algorithms*, pages 138–147, 1990.

[Sch99] G. Schaeffer. Random sampling of large planar maps and convex polyhedra. In ACM, editor, *Annual ACM Symposium on Theory of Computing (Atlanta, GA, 1999)*, pages 760–769 (electronic), New-York, 1999.

[Tar74] R. E. Tarjan. Testing graph connectivity. In *Conference Record of Sixth Annual ACM Symposium on Theory of Computing (Seattle, Washington)*, pages 185–193, 1974.

[TT86] R. Tamassia and I. G. Tollis. A unified approach to visibility representations of planar graphs. *Discrete Comput. Geom.*, 1(4):321–341, 1986.

[Tut60] W. T. Tutte. Convex representations of graphs. In *Proc. London Math. Society*, volume 10, pages 304–320, 1960.

[Tut63] W. T. Tutte. How to draw a graph. In *Proc. London Math. Society*, volume 13, pages 743–768, 1963.