Planarization of Clustered Graphs
(Extended Abstract)*

Giuseppe Di Battista, Walter Didimo, and A. Marcandalli

Dipartimento di Informatica e Automazione, Università di Roma Tre, via della Vasca Navale 79, 00146 Roma, Italy. {gdb,didimo,marcanda}@dia.uniroma3.it

Abstract. We propose a planarization algorithm for clustered graphs and experimentally test its efficiency and effectiveness. Further, we integrate our planarization strategy into a complete topology-shape-metrics algorithm for drawing clustered graphs in the orthogonal drawing convention.

1 Introduction

Several application domains require to draw graphs in such a way that vertices are grouped together into *clusters*. For example, a large computer network is often partitioned into areas, and it is usual to represent the systems (routers, switches, etc.) belonging to the same area inside the same region (rectangle or convex polygon) of the drawing. Further, areas are recursively partitioned into sub-areas with a structure that can have many levels. Other examples come from the Computer Aided Software Engineering field, where in the diagrams representing the design process it is often required to highlight the "cohesion" among certain components.

Such application requirements motivated the study of algorithms for drawing graphs with recursive clustering structures over the vertices, such as *compound digraphs* and *clustered graphs*. Fig. 1 shows an example of clustered graph. Some of the papers on the subject are discussed below. In [23] Sugiyama and Misue proposed an algorithm for drawing compound digraphs within a drawing convention that is related to the one commonly adopted for hierarchical graphs [24]. Feng, Cohen, and Eades [14] studied the concept of planarity for clustered graphs (*c-planarity*) and provided the first c-planarity testing algorithm. The construction of orthogonal drawings of c-planar clustered graphs, based on visibility representations and bend-stretching transformations, is studied by Eades et al. [11]. Straight line and orthogonal drawing algorithms, three dimensional visualization techniques, and force directed methods are presented in [13,10,20,3], in [9], and in [17], respectively. The related problem of constructing balanced clusters on a given planar graph is studied by Duncan et al. [8]; the constructed clustered

* Research supported in part by the project "Algorithms for Large Data Sets: Science and Engineering" of the Italian Ministry of University and Scientific and Technological Research.

P. Mutzel, M. Jünger, and S. Leipert (Eds.): GD 2001, LNCS 2265, pp. 60–74, 2002.

graph satisfies the conditions for c-planarity. However, as far as we know, a complete algorithm, based on the topology-shape-metrics approach [5], for drawing clustered graphs in the orthogonal drawing convention is currently not available.

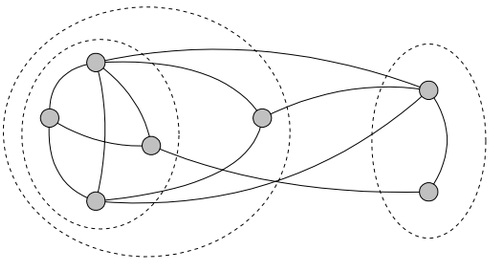


Fig. 1. A clustered graph: clusters are dashed.

The main results presented in this paper are the following:

- We propose a planarization algorithm for a clustered graph C with n vertices, m edges, and c clusters. It runs in $O(m\chi + m^2c + mnc)$ time, where χ is the number of crossings inserted in C by the algorithm. This is, as far as we know, the first complete planarization algorithm for clustered graphs.
- As a by-product of the planarization algorithm, we present an algorithm for constructing a "spanning tree" of C in $O(m)$ time. A definition of the term spanning tree for a clustered graph is given in Section 3.
- We present an implementation of the planarization algorithm with efficient data structures and show an experimental study that compares the effectiveness and the efficiency of the algorithm with those of simpler planarization techniques. The computational results put in evidence performance that are reasonable in many application domains.
- We describe the implementation of a complete drawing algorithm for clustered graphs, based on the topology-shape-metrics approach.

The rest of this paper is organized as follows. In Section 2 we introduce the basic terminology on clustered graphs. The planarization algorithm is presented in Sections 3 and 4. A simple planarization algorithm, whose behaviour is used as a reference point in our experiments, is described in Section 5. An experimental analysis of the effectiveness and of the efficiency of our planarization algorithm is presented in Section 6. The implementation of a complete drawing algorithm for clustered graphs is sketched in Section 7. Open problems are addressed in Section 8.

2 Preliminaries on Clustered Graphs and c-Planarity

We assume familiarity with connectivity and planarity of graphs [12,21,5]. Since we consider only planar graphs, we use the term *embedding* instead of *planar*

embedding. Also, we assume that an embedding determines a choice for the external face.

We import several definitions from the papers on c-planarity by Cohen, Eades, and Feng [14,13]. Given a graph, we call *cluster* a subset of its vertices. A cluster that is included into another cluster s is a *sub-cluster* of s. A *clustered graph* $C = (G, T)$ consists of an (undirected) graph $G = (V, E)$ and a rooted tree T such that the leaves of T are the vertices of G and each non-leaf node of T has at least two children. Each node ν of T corresponds to the cluster $V(\nu)$ of G whose vertices are the leaves of the subtree rooted at ν. The subgraph of G induced by $V(\nu)$ is denoted as $G(\nu)$. An edge e between a vertex of $V(\nu)$ and a vertex of $V - V(\nu)$ is said to be *incident* on ν. Clearly, the root of T does not have incident edges. In the paper we denote by n, m, and c the number of vertices of G, edges of G, and non-leaf nodes of T, respectively. Note that, since each non-leaf node of T has at least two children, we have $c < n$.

Graph G and tree T are called the *underlying graph* and the *inclusion tree* of C, respectively. Observe that, given two nodes μ and ν of T such that μ is an ancestor of ν, $V(\nu)$ is a sub-cluster of $V(\mu)$.

Clustered graph C is *connected* if for each node ν of T we have that $G(\nu)$ is connected. In a connected clustered graph $m \geq n - 1$.

Suppose that $C_1 = (G_1, T_1)$ and $C_2 = (G_2, T_2)$ are two clustered graphs such that T_1 is a subtree of T_2 and for each node ν of T_1, $G_1(\nu)$ is a subgraph of $G_2(\nu)$; then C_1 is a *sub-clustered-graph* of C_2.

In a *drawing* of a clustered graph $C = (G, T)$ each vertex of G is a point and each edge is a simple curve between its end-vertices. For each node ν of T, $G(\nu)$ is drawn inside a simple closed region $R(\nu)$ such that: (i) for each node μ of T that is neither an ancestor nor a descendant of ν, $R(\mu)$ is completely contained in the exterior of $R(\nu)$; (ii) an edge e incident on ν crosses the boundary of $R(\nu)$ exactly once. We say that edge e and region R have an *edge-region crossing* if both endpoints of e are outside R and e crosses the boundary of R more than once; we assume that a drawing of a clustered graph does not have edge-region crossings.

A drawing of a clustered graph is *c-planar* if it does not have edge crossings. A clustered graph is *c-planar* if it has a c-planar drawing.

Theorem 1. [14] *A connected clustered graph $C = (G, T)$ is c-planar if and only if G is planar and there exists a planar drawing of G such that for each node ν of T all the vertices and edges of $G - G(\nu)$ are in the external face of $G(\nu)$.*

Theorem 2. [14] *Let C be a connected clustered graph whose underlying graph has n vertices. There exists an $O(nc)$ time algorithm for testing the c-planarity of C.*

Given a graph $G = (V, E)$ that is (in general) not planar a *planarization* of G is an embedded planar graph $G' = (V', E')$ such that:

- $V' = V \cup D$; the vertices of D represent crossings between edges, and are called *dummy vertices*;

- dummy vertices have degree equal to four;
- each edge (u, v) of E is associated with a path $u, d_1, \ldots, d_k, v$ ($k \geq 0$) of G' such that $d_i \in D$ ($i = 1, \ldots, k$); we call such a path an *edge path* of G'; and
- each dummy vertex is incident on two distinct edge paths and the edges of the same path are not consecutive in the embedding around the dummy vertex.

Given a clustered graph $C = (G, T)$ that is not c-planar, a *planarization* of C is a c-planar clustered graph $C' = (G', T')$ such that:

- G' is a planarization of G;
- T' is a tree obtained from T by adding one leaf for each dummy vertex of G';
- let d be a dummy vertex of G' and let u and v be the end-vertices of any edge path containing d. Vertex d is a child of a node of T' (i.e. it belongs to a cluster of C') that lies on the path of T' between u and v.

3 Computing a Maximal c-Planar Sub-clustered-Graph

We now describe a planarization algorithm for clustered graphs, which we call `ClusteredGraphPlanarizer`. It follows the usual planarization strategy of the topology-shape-metrics approach. First, a maximal c-planar sub-clustered-graph of the given clustered graph is computed (Algorithm `MaximalcPlanar`). Second, the edges removed in the first step are reinserted (Algorithm `Reinsertion`) by suitably adding "dummy" vertices representing crossings. In this section we concentrate on Algorithm `MaximalcPlanar`, while Algorithm `Reinsertion` will be discussed in Section 4.

Although the problem of finding a maximal or a maximum planar subgraph of a given graph has been deeply studied (see, e.g., [7,19,18]), the problem of determining a maximal c-planar sub-clustered-graph of a clustered graph $C = (G, T)$ has not been investigated yet.

A possibility for solving the problem could be the one of inserting, starting from the empty graph, one-by-one the edges of G, repeating a c-planarity testing for each edge insertion and discarding the edges that cause crossings. This technique, although attractive for its simplicity, cannot be easily adopted. In fact, the only existing c-planarity testing algorithm [14] works only for connected clustered-graphs, while the intermediate clustered-graphs produced by the technique might be non-connected.

We adopt a different strategy. Instead of starting from the empty graph, we start from a connected c-planar sub-clustered-graph $C' = (G', T)$ that contains all the vertices of G. Such a sub-clustered-graph is computed so that its underlying graph contains a spanning tree of G. The computation of C' consists of two steps:

1. We compute a sub-clustered-graph C'' of C so that its underlying graph is a spanning tree of G. We call `SpanningTree` this step of the algorithm.

2. We compute C' from C'' by inserting edges that do not violate the c-planarity. We call `SimpleReinsertion` this step of the algorithm.

Intuitively, Algorithm `SpanningTree` constructs C'' by associating a spanning tree with each node of T (cluster of C) and by merging all such spanning trees. More formally, for each edge (u, v) of G we define the *allocation node* of (u, v) as $lca(u, v)$ in T, where $lca(u, v)$ denotes the *lowest common ancestor* of u and v. Note that $\nu = lca(u, v)$ is the deepest node of T such that cluster $V(\nu)$ contains both u and v. See Fig. 2 to have an example.

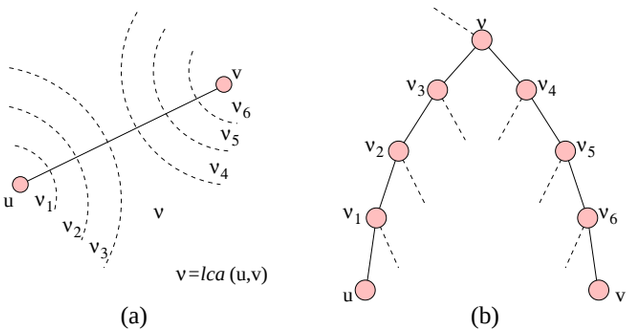


Fig. 2. (a) A fragment of a clustered graph. (b) A fragment of the inclusion tree, with vertices u and v and their lowest common ancestor (allocation node of (u, v)).

Let ν be a non-leaf node of T with children $\nu_1, \ldots, \nu_k$. Graph $F(\nu)$ is defined as follows (see for example Fig. 3):

- The vertices of $F(\nu)$ are $\nu_1, \ldots, \nu_k$.
- For each edge (u, v) such that $u \in V(\nu_i)$ and $v \in V(\nu_j)$ $(i \neq j)$ there is an edge (ν_i, ν_j) in $F(\nu)$. Edge (ν_i, ν_j) is the *representative* of (u, v) in $F(\nu)$.

Property 1. For each edge (u, v) of G there exists exactly one node ν of T such that $F(\nu)$ contains a representative of (u, v); ν is the allocation node of (u, v).

Property 2. The total number of vertices of graphs $F(\nu)$ is $n + c - 1$, and the total number of edges of graphs $F(\nu)$ is m.

We denote by $S(\nu)$ a spanning tree of $F(\nu)$. We construct a subgraph ST of G by selecting in G only the edges (u, v) such that, if ν is the allocation node of (u, v), then (u, v) has its representative in $S(\nu)$.

Property 3. Graph ST is a spanning tree of G.

We call $ST(\nu)$ the graph obtained from the intersection of ST and $G(\nu)$.

Property 4. Graph $ST(\nu)$ is a spanning tree of $G(\nu)$, for each ν in T.

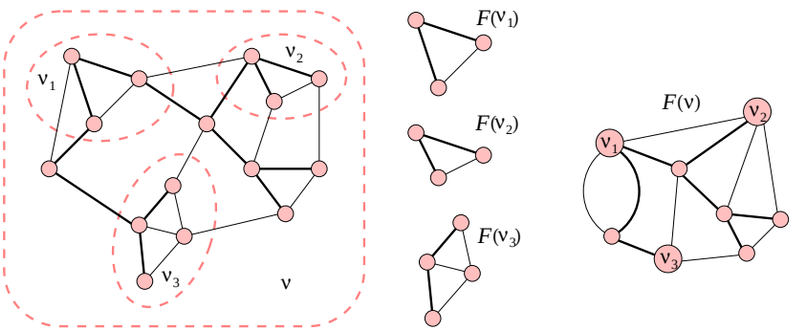


Fig. 3. A clustered graph and graphs $F(\nu)$, $F(\nu_1)$, $F(\nu_2)$, and $F(\nu_3)$. The edges in bold are those of the spanning trees.

Lemma 1. *A connected clustered graph whose underlying graph is a tree is c-planar.*

Proof. Follows from Theorem 1 and from the fact that, for any drawing, all the vertices of a tree stay on the same face.

Lemma 2. *Clustered graph $C'' = (ST, T)$ is a c-planar connected sub-clustered-graph of C.*

Proof. From Property 4 it follows that each $ST(\nu)$ is connected. Hence C'' is connected and its underlying graph is a tree. From Lemma 1 it follows that C'' is also c-planar.

Theorem 3. *Let $C = (G, T)$ be a connected clustered graph and let m be the number of edges of G. Algorithm* `SpanningTree` *computes a connected sub-clustered-graph of C whose underlying graph is a spanning tree of G in $O(m)$ time.*

Proof. For each node ν of T the construction of $F(\nu)$ is performed by visiting the edges of G and by assigning each edge to a specific $F(\nu)$. This requires the computation of the allocation node $\nu = lca(u, v)$ of each edge (u, v) and the computation of the children of ν in the paths from ν to u and from ν to v. A trivial implementation of this step would require $O(mc)$ time, where c is the number of non-leaf nodes of T. However, we can do it in $O(m)$ time by using a variation of the Schieber and Vishkin data structure [22]. The original data structure allows us, after a linear time preprocessing, to perform lowest common ancestor queries on a tree in constant time. Unfortunately, it does not give primitives to determine the required children of the lowest common ancestor. It is possible to suitably enrich the information associated with the "inlabel paths" of the data structure to solve the problem.

The computation of the spanning trees can be done in linear time.

Algorithm `SimpleReinsertion` constructs C' by reinserting into $C'' = (ST, T)$ some edges that do not cause crossings. This is done with the purpose of reducing the number of times the c-planarity testing is executed. The reinsertion strategy is based on the following lemma.

Lemma 3. *A connected clustered graph whose underlying graph has n vertices and n edges is c-planar.*

Observe that, while any connected graph with n vertices and $n+2$ edges is planar, there exist connected clustered graphs with n vertices and $n+1$ edges that are not c-planar. See Fig. 4.

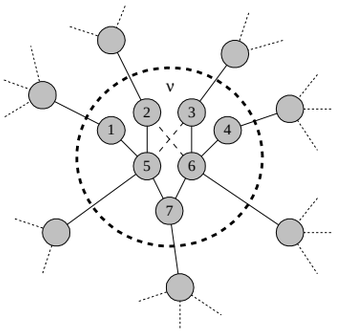


Fig. 4. Inserting edge $(2,6)$ and $(5,3)$ generates a crossing.

The following lemma generalizes Lemma 3.

Lemma 4. *Let $C = (G,T)$ be a connected clustered graph whose underlying graph G is a tree. Let $u_1,\ldots,u_k,v_1,\ldots,v_k$ be vertices of G such that edges $(u_1,v_1),(u_2,v_2),\ldots,(u_k,v_k)$ are not in G. If for each pair $(u_i,v_i),(u_j,v_j)$ $(i \neq j)$, $lca(u_i,v_j) \neq lca(lca(u_i,v_i),lca(u_j,v_j)) \neq lca(u_j,v_i)$, then C remains c-planar after adding $(u_1,v_1),(u_2,v_2),\ldots,(u_k,v_k)$.*

Because of Lemma 4, we can do the following. We visit T bottom-up. For each ν of T we check (i) if no edge of $G(\nu) - ST(\nu)$ has been already reinserted and (ii) if $G(\nu) - ST(\nu)$ is not empty. If both conditions hold, then we reinsert an edge of $G(\nu) - ST(\nu)$. We apply the same procedure to all the nodes of T.

Theorem 4. *Let C be a connected clustered graph whose underlying graph has m edges. Algorithm* `SimpleReinsertion` *computes a connected c-planar sub-clustered-graph of C in $O(m)$ time.*

After the construction of C' we reinsert a maximal number of edges of C by testing c-planarity for each edge insertion, using each time the c-planarity testing algorithm in [14]. After the last reinsertion we also compute a c-planar embedding [14].

The following theorem is proved by using Theorems 3, 4, and 2.

Theorem 5. *Let C be a connected clustered graph whose underlying graph has n vertices and m edges. Algorithm* `MaximalcPlanar` *computes a connected maximal c-planar embedded sub-clustered-graph of C in $O(mnc)$ time.*

4 Reinsertion of the Discarded Edges

We describe Algorithm `Reinsertion`. A technique similar to the one adopted by algorithm `Reinsertion` is sketched in [20]. Once a maximal connected c-planar embedded sub-clustered-graph $C_{mp} = (G_{mp}, T)$ of $C = (G, T)$ has been computed (where G_{mp} denotes a planar subgraph of G), the remaining edges are reinserted by using a variation of the "classical" technique [5] that is based on computing shortest paths on the dual graph of G_{mp}.

In fact, in the case of clustered graphs it is not possible to apply exactly the same technique that is adopted for graphs. Fig. 5 shows how reinserting an edge (u, v) by following a path on the dual graph of G_{mp} can cause: (i) an edge-region crossing (Fig. 5.a) and/or (ii) more than one crossing between a cluster and one of its incident edges. (Fig. 5.a).

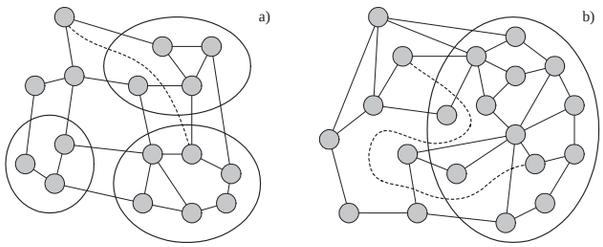


Fig. 5. Examples of wrong insertions: (a) edge-region crossing and (b) more than one crossing between a cluster and one of its incident edges.

In order to avoid the above problems, we compute the shortest paths on a planar embedded graph constructed as follows. Observe that the embedding of C_{mp} induces a planar embedding on G_{mp}.

- We "materialize" the boundary of each cluster. Namely, we augment G_{mp} as follows: (i) for each pair e, ν such that e is an edge of G_{mp} incident on cluster ν, we split e by inserting a *boundary vertex* $v_{e,\nu}$; (ii) for each face we traverse the border counterclockwise and construct a list of boundary vertices; (iii) for each pair of boundary vertices $v_{e_1,\nu}$, $v_{e_2,\nu}$ that belong to the same face f and that are consecutive in the list of f, we insert a *boundary edge* $(v_{e_1,\nu}, v_{e_2,\nu})$. See Fig. 6. Observe that each boundary edge corresponds to one specific cluster and that the boundary edges corresponding to the same cluster are arranged into simple cycles (*boundary cycles*). Boundary cycles and clusters (except for the root of T) are in one-to-one correspondence.
- We compute the planar embedded dual G'_{mp} of the planar graph described in the previous step. Note that each face f of G'_{mp} is inside a certain set of boundary cycles. Such cycles correspond to clusters that are on a rooted path p of T. If the set is empty we *associate* f with the root of T; otherwise, we *associate* f with the lowest node of p.

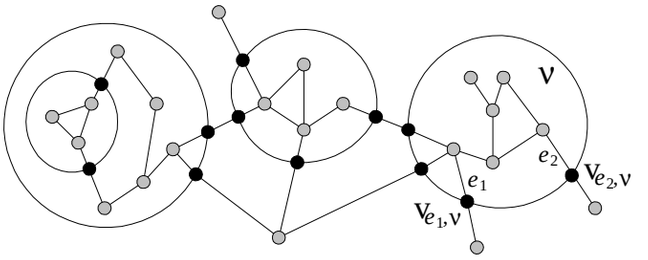


Fig. 6. Insertion of edges and vertices for representing the boundaries of the clusters.

At each edge reinsertion we perform the following algorithm. Let (u, v) be the edge to be reinserted and let $\nu_1, \ldots, \nu_k$ be the nodes of the path of T from u to v (the path contains the allocation node of (u, v)). We orient or temporarily remove each edge (f, g) of G'_{mp} as follows (See Fig. 7).

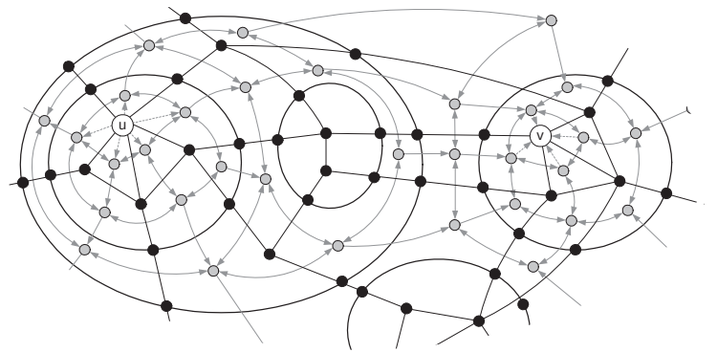


Fig. 7. Orientation and removal of edges of graph G'_{mp}. The edges of G'_{mp} are grey.

Two cases are possible:

1. f is associated with node ν_i and g is associated with node ν_j $(i, j \in \{1, \ldots, k\})$: if $i = j$, then we give a bidirectional orientation to (f, g), else (assume without loss of generality that $i < j$) we orient (f, g) from f to g.
2. either f or g is not associated with a node of $\nu_1, \ldots, \nu_k$: we temporarily remove (f, g).

After orienting the graph and temporarily removing some of its edges, we compute a directed shortest path between the set of faces incident on u and the set of faces incident on v. Edge (u, v) is reinserted into G_{mp} by following the computed shortest path. Dummy vertices representing edge crossings are inserted. The dual graph G'_{mp} is modified accordingly. The temporarily removed edges are restored and the orientation of the edges is removed.

Theorem 6. *Let $C_{mp} = (G_{mp}, T)$ be a maximal connected c-planar embedded sub-clustered-graph of a clustered graph $C = (G, T)$. Let m be the number of edges of G and let m' be the number of edges of $G - G_{mp}$. Starting from C_{mp}, Algorithm* `Reinsertion` *computes a planarization of C with χ dummy vertices (crossings) in $O(m'\chi + m'mc)$ time.*

Proof. Augmenting the embedding of C_{mp} inserting the border edges is done in $O(nc)$ time, where n is the number of vertices of G and c is the number of non-leaf nodes of T. The same amount of time is required to compute the dual graph G'_{mp}. At each edge reinsertion we spend $O(\chi + mc)$ time both for computing the orientation of the dual graph and for computing, with a breadth-first-search, the shortest path. We remark that each reinsertion can originate at most c crossings between the inserted edge and the border edges of the clusters.

In the implementation of Algorithm `Reinsertion` we adopted a slightly different strategy. Although such a strategy does not reduce the time complexity of the algorithm, it has shown positive effects on the execution time of the algorithm.

A heavy step of the algorithm is the one that requires the orientation and the removal of some edges of G'_{mp} at each edge reinsertion. To avoid that, we have used a modified breadth-first-search that is performed directly on G'_{mp}. Suppose that we are reinserting edge (u, v) and that we are visiting face f associated with cluster ν. Also, suppose we are going to traverse edge (f, g). Face g can be visited if and only if the cluster associated with g is either ν or the cluster that immediately follows ν in the path p of T from u to v. Of course, this requires the computation of p. This can be efficiently done by determining the allocation node of (u, v) in constant time exploiting the same variation of the Schieber and Vishkin data structure [22] mentioned in the proof of Theorem 3. A simpler but less efficient alternative would be the one of determining p with a dove-tail bottom-up visit of T starting from u and v.

The following theorem summarizes the time complexity of the whole planarization algorithm.

Theorem 7. *Let $C = (G, T)$ be a clustered-graph. Let n, m, and c be the number of vertices of G, edges of G, and non-leaf nodes of T, respectively. Algorithm* `ClusteredGraphPlanarizer` *computes a planarization of C with χ dummy vertices in $O(m\chi + m^2c + mnc)$ time.*

5 A Simple Planarization Algorithm

We now describe a planarization algorithm for a clustered graph $C = (G, T)$ simpler than the one described above. We use it as a reference point in experimenting the effectiveness of our algorithm. We call it `SimplePlanarizer`.

We visit T top-down. We start from the root ν of T and compute graph $F(\nu)$. Then, we apply any planarization algorithm (see, e.g. [5]), in order to compute a planarization of $F(\nu)$. Consider now a child ν_1 of ν. We would like to compute a planarization of $F(\nu_1)$ and to "glue" it into $F(\nu)$. It turns out that the ordering

of the edges incident on ν_1 in the planarization of $F(\nu)$ induces a constraint in the planarization of $F(\nu_1)$. Namely, the vertices of $F(\nu_1)$ that are incident on edges of G that are also incident on cluster ν must satisfy two constraints:

- they must stay on the external face of $F(\nu_1)$ and
- while visiting the external face of $F(\nu_1)$ they must appear in the same clockwise order of the edges incident on ν_1 in $F(\nu)$.

The above type of constraints on the topology can be imposed by using the facilities of existing graph drawing libraries (see, e.g. [16]). The same strategy is applied to all the descendants of ν, recursively.

We conclude this section by observing that Algorithm `SimplePlanarizer` works also for the case of non connected clustered graphs.

6 Experimental Study

The algorithms presented in Sections 3, 4, and 5 have been implemented and extensively tested. The implementation of all algorithms have been done by using the C++ language (Visual C++ compiler) and exploiting the basic graph drawing facilities of the `GDToolkit` library [16]. Also, for implementing Algorithm `MaximalcPlanar`, we used the implementation of the c-planarity testing algorithm [14] in the AGD library [1] .

All the experiments have run on a personal computer equipped with an AMD Athlon K7 (500 MHz) and 128 MB of RAM.

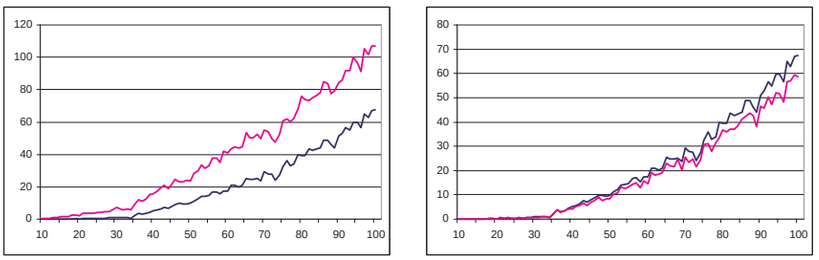
We have used two test suites. The first one (*Suite 1*) consists of more than $11,000$ graphs with number of vertices ranging from 10 to 100; it has been introduced in [6] and since then has become a widely used test-suite in experimental Graph Drawing. The average density of its graphs is about 1.3.

The second test suite (*Suite 2*) has been conceived to test the algorithms against graphs with higher density. It has been generated with a graph generator that works as follows. For generating a graph the user specifies two values: the number n of vertices and the desired density d. The graph is generated by randomly inserting nd edges on the set of n vertices. The result is kept if it is connected and does not contain multiple edges, otherwise it is discarded and the generation process is repeated. With such a graph generator we have generated 10 groups of graphs. Each group has a fixed density and contains 50 graphs with number of vertices in the range $10 - 100$. Densities range from 1.2 to 3.

The graphs of Suites 1 and 2 are just graphs and do not have a cluster structure. In order to perform our experiments, we have augmented them with clusters with the following algorithm. Let G be a graph of a suite. We randomly select a vertex v and compute a breadth-first-search spanning tree of G starting from v. At this point we randomly select a vertex u and perform a breadth-first-search on the spanning tree starting from u and visiting an edge with probability 0.7. The visited set of vertices of G is a cluster. The same algorithm is recursively repeated on the spanning tree using as boundary of the visits the clusters already computed. It is easy to see that the obtained clustered graphs are connected.

They have an average number of clusters that is about 1/4 of the number of vertices. The depth of the obtained inclusion trees is in the range $2 - 5$. The probability 0.7 of visiting an edge has been chosen after several experiments to keep high the number of generated clusters.

Fig. 8.a shows the average number of edge crossings obtained by Algorithm `SimplePlanarizer` and by Algorithm `ClusteredGraphPlanarizer` on the clustered graphs of Suite 1. Algorithm `ClusteredGraphPlanarizer` outperforms Algorithm `SimplePlanarizer` by about 40%. Another interesting reference point for evaluating the effectiveness of Algorithm `ClusteredGraphPlanarizer` is to compare its behavior against the one of a standard planarizer, used for planarizing the same graphs, considered without clusters. Fig. 8.b shows a comparison with the planarizer of `GDToolkit`. The results are quite encouraging and show how clusters have a limited impact on the number of crossings.

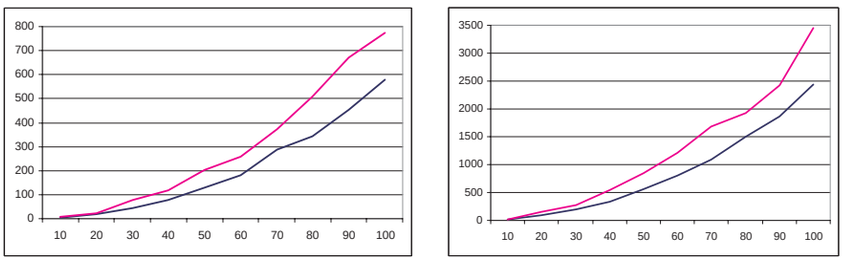


(a) (b)

Fig. 8. Suite 1. (a) Average number of edge crossings obtained by Algorithm `SimplePlanarizer` and by Algorithm `ClusteredGraphPlanarizer` (curve below). (b) Average number of edge crossings obtained by Algorithm `ClusteredGraphPlanarizer` and by the `GDToolkit` planarizer (curve below).

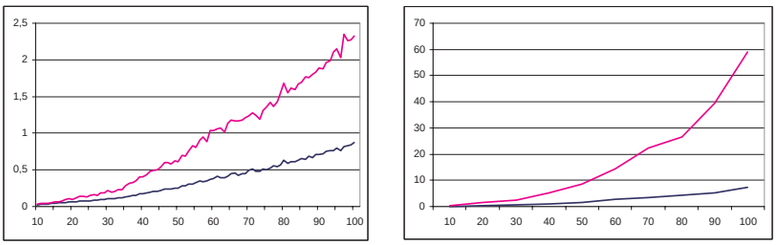
Fig. 9 confirms the good behavior of `ClusteredGraphPlanarizer` also with graphs of higher density. It has been obtained using the graphs with densities 2 and 3 of Suite 2. We omit the graphics obtained with other densities, since they confirm what appears in Fig. 9.

Fig. 10 shows how Algorithm `ClusteredGraphPlanarizer` have time performance that are reasonable for usual applications. Even for the largest graphs of Suite 1 (Suite 2) the time is less than 4 seconds (9 seconds). Algorithm `SimplePlanarizer` is much less efficient. However, the implementation of Algorithm `SimplePlanarizer` did not exploit sophisticated data structures at the level of Algorithm `ClusteredGraphPlanarizer`. We believe that a more careful implementation could reduce the gap between the time performance of the two algorithms.



(a) (b)

Fig. 9. Suite 2. Average number of edge crossings obtained by Algorithm `SimplePlanarizer` and by Algorithm `ClusteredGraphPlanarizer` (curves below). (a) Density 2. (b) Density 3.



(a) (b)

Fig. 10. (a) Suite 1. Average time (seconds) spent by Algorithm `SimplePlanarizer` and by Algorithm `ClusteredGraphPlanarizer` (curve below). (b) Suite 2, density 3. Average time (seconds) spent by by Algorithm `SimplePlanarizer` and by Algorithm `ClusteredGraphPlanarizer` (curve below).

7 A Complete Drawing Algorithm for Clustered Graphs

We have embedded Algorithm `ClusteredGraphPlanarizer` into a complete topology-shape-metrics algorithm for computing drawings of clustered graph within the orthogonal podevsnef [15] drawing convention.

After the planarization has been performed, for computing the shape of the drawing, representing clusters as boxes, we use the same technique adopted in [3, 20]. Namely, we use the variation described in [2] of the min-cost-flow based algorithm in [15]. The boundary cycles representing the borders of the clusters are constrained to have the shape of a rectangle. This can be easily done by using constraints on the flow traversing the edges of those cycles.

We compute the metric of the drawing with the compaction algorithm described in [4]. Fig. 11 shows a drawing computed by the algorithm.

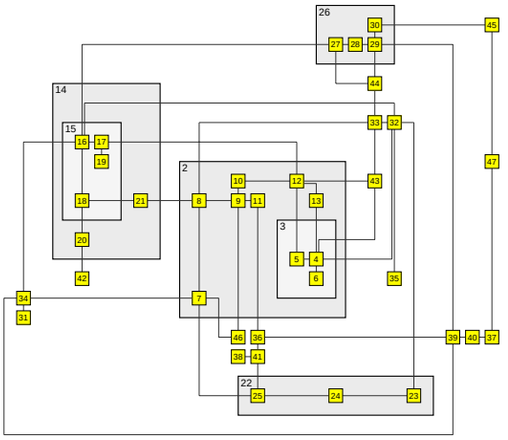


Fig. 11. A clustered graph with 40 vertices and 7 clusters.

8 Open Problems and Future Work

Several problems are still open in the field of clustered graphs. In particular, we are interested in finding planarization algorithms that are more efficient (theoretically and experimentally) than the one described in this paper. Further, the role of connectivity in the c-planarity of clustered graphs is still unclear. Namely, it is unknown whether the c-planarity testing problem for non connected clustered graphs is computationally hard or not. Studies on this matter could open new perspectives for the planarization problem.

References

1. AGD. A library of algorithms for graph drawing. Online. http://www.mpi-sb.mpg.de/AGD/.
2. P. Bertolazzi, G. Di Battista, and W. Didimo. Computing orthogonal drawings with the minimum numbr of bends. *IEEE Transactions on Computers*, 49(8), 2000.
3. U. Brandes, S. Cornelsen, and D. Wagner. How to draw the minimum cuts of a planar graph. In J. Marks, editor, *Graph Drawing (Proc. GD '00)*, volume 1984 of *Lecture Notes Comput. Sci.*, pages 103–114. Springer-Verlag, 2000.
4. G. Di Battista, W. Didimo, M. Patrignani, and M. Pizzonia. Orthogonal and quasi-upward drawings with vertices of arbitrary size. In *Proc. GD '99*, volume 1731 of *LNCS*, pages 297–310, 2000.
5. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, Upper Saddle River, NJ, 1999.
6. G. Di Battista, A. Garg, G. Liotta, R. Tamassia, E. Tassinari, and F. Vargiu. An experimental comparison of four graph drawing algorithms. *Comput. Geom. Theory Appl.*, 7:303–325, 1997.
7. H. N. Djidjev. A linear algorithm for the maximal planar subgraph problem. In *Proc. 4th Workshop Algorithms Data Struct.*, Lecture Notes Comput. Sci. Springer-Verlag, 1995.

8. C. A. Duncan, M. T. Goodrich, and S. G. Kobourov. Planarity-preserving clustering and embedding for large planar graphs. In J. Kratochvil, editor, *Graph Drawing (Proc. GD '99)*, volume 1731 of *Lecture Notes Comput. Sci.*, pages 186–196. Springer-Verlag, 1999.
9. P. Eades and Q. W. Feng. Multilevel visualization of clustered graphs. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 101–112. Springer-Verlag, 1996.
10. P. Eades, Q. W. Feng, and X. Lin. Straight line drawing algorithms for hierarchical graphs and clustered graphs. In S. North, editor, *Graph Drawing (Proc. GD '96)*, volume 1190 of *Lecture Notes Comput. Sci.*, pages 113–128. Springer-Verlag, 1996.
11. P. Eades, Q. W. Feng, and H. Nagamochi. Drawing clustered graphs on an orthogonal grid. *Journal of Graph Algorithms and Applications*, 3(4):3–29, 2000.
12. S. Even. *Graph Algorithms*. Computer Science Press, Potomac, Maryland, 1979.
13. Q. W. Feng, R. Cohen, and P. Eades. How to draw a planar clustered graph. In *Computing and Combinatorics (Cocoon '95)*, volume 959 of *Lecture Notes Comput. Sci.*, pages 21–30. Springer-Verlag, 1995.
14. Q. W. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In P. Spirakis, editor, *Symposium on Algorithms (Proc. ESA '95)*, volume 979 of *Lecture Notes Comput. Sci.*, pages 213–226. Springer-Verlag, 1995.
15. U. Fößmeier and M. Kaufmann. Drawing high degree graphs with low bend numbers. In F. J. Brandenburg, editor, *Graph Drawing (Proc. GD '95)*, volume 1027 of *Lecture Notes Comput. Sci.*, pages 254–266. Springer-Verlag, 1996.
16. GDToolkit. Graph drawing toolkit. Online. http://www.dia.uniroma3.it/∼gdt.
17. M. L. Huang and P. Eades. A fully animated interactive system for clustering and navigating huge graphs. In S. H. Whitesides, editor, *Graph Drawing (Proc. GD '98)*, volume 1547 of *Lecture Notes Comput. Sci.*, pages 374–383. Springer-Verlag, 1998.
18. M. Jünger, E. K. Lee, P. Mutzel, and T. Odenthal. A polyhedral approach to the multi-layer crossing number problem. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, number 1353 in Lecture Notes Comput. Sci., pages 13–24. Springer-Verlag, 1997.
19. M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16(1):33–59, 1996. (special issue on Graph Drawing, edited by G. Di Battista and R. Tamassia).
20. D. Lütke-Hüttmann. *Knickminimales Zeichnen 4-planarer Clustergraphen*. Master's thesis, Universität des Saarlandes, 1999.
21. T. Nishizeki and N. Chiba. Planar graphs: Theory and algorithms. *Ann. Discrete Math.*, 32, 1988.
22. B. Schieber and U. Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988.
23. K. Sugiyama and K. Misue. Visualization of structural information: Automatic drawing of compound digraphs. *IEEE Trans. Softw. Eng.*, 21(4):876–892, 1991.
24. K. Sugiyama, S. Tagawa, and M. Toda. Methods for visual understanding of hierarchical systems. *IEEE Trans. Syst. Man Cybern.*, SMC-11(2):109–125, 1981.