Communications to SIMAI Congress,
ISSN 1827-9015, Vol. 3 (2009) 301 (12pp)
DOI: 10.1685/CSC09301

A New Tool for Rectangular Dualization

Massimo Ancona, Gianluca Quercini, Paolo Pastorelli

Dipartimento di Informatica, Facoltà di Scienze MM.FF.NN.
Università degli Studi di Genova, Italy
ancona, quercini@unige.it
pastorelli.paolo@gmail.com

Abstract

OcORD is a software tool for rectangular dualization. Rectangular dualization is a dual representation of a plane graph introduced in the early seventies. It proved to be effective in applications such as architectural space planning and VLSI floorplanning. However, not all plane graphs admit a rectangular dual, which imposes severe limitations on its use in other applications. OcORD aims at freeing rectangular dualization from such restrictions and proving its effectiveness in graph visualization. This is achieved in two ways. Firstly, OcORD features a new linear-time algorithm creating a rectangular dual of any plane graph. Secondly, it shows how nice drawings of a graph can be easily obtained from its rectangular dual. Finally, the automatic generation of a Virtual World through rectangular dualization is described.

Keywords: `Rectangular Dualization, Orthogonal Graph Drawing, Bus-Mode Drawing, Clustered Graphs, Electronic Institutions.`

1. Introduction

A *rectangular dual* (RD) of a plane graph G is a dissection of a rectangle into as many non-overlapping subrectangles as the number of vertices of G, such that any two subrectangles are adjacent if and only if the corresponding vertices are adjacent in G. Its history dates back 40 years, when Grason described an automated approach to the space planning problem in architecture [9].

The most popular application of RD is *VLSI floorplanning*. The advent of the integrated circuits revolutionized the hardware industry more than any other technology. However, the design of a VLSI circuit becomes extremely challenging as its size increases. Special software, known as CAD (Computer-Aided Design), automate most of the design task and let users concentrate on high-level details. Usually, the design of a VLSI circuit is subdivided into different abstraction levels. In particular, at the *floorplanning* stage, the circuit is seen as a collection of interconnected rectangular

Received 15/02/2009, in final form 19/06/2009
Published 31/07/2009

blocks (or *macro-cells*), which have to be placed on a chip in such a way that some objective functions (area, latency, wire length) are minimized (or maximized). The interconnection between macro-cells is described with an *adjacency graph*; a rectangular dual of this graph is exactly a floorplan of the circuit.

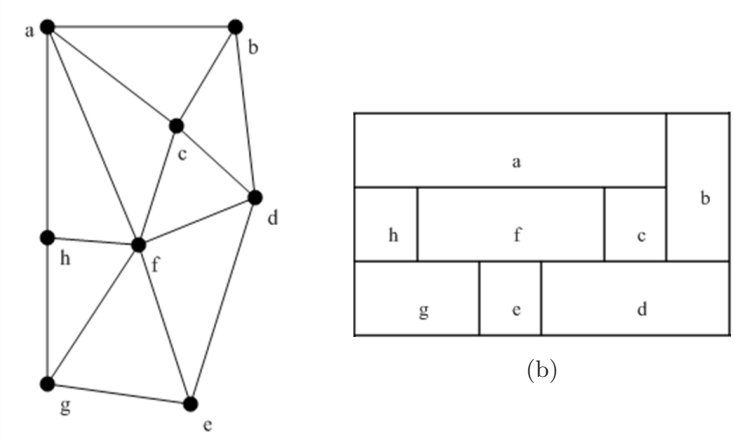
Lots of papers have been written on RD [9,11–13]. Despite of this, RD is still limited by the lack of efficient algorithms. In particular, at least in our knowledge, there is no algorithm which creates a rectangular dual of any plane graph in linear time. In this paper, we present our advances in RD, while describing the main features of OcORD, our software tool for rectangular dualization.

2. Preliminaries

This section introduces the key concepts used in this paper. The reader is referred to [8] for a quick introduction to graph theory.

A *rectangular dual* $\mathscr{R}(G)$ of a plane graph G is a pair (Γ, f) such that:

- $\Gamma = \{R_1, R_2, ..., R_n\}$ is a set of simple non-overlapping rectangles, which form a partition of a rectangle R (called *enclosure rectangle*);
- $f : V(G) \rightarrow \Gamma$ associates each vertex of G to a rectangle in Γ. f is a bijective function;
- Two vertices u and v are adjacent in G if and only if $f(u)$ and $f(v)$ are adjacent in $\mathscr{R}(G)$.



(a) (b)

Fig. 1. (a) A plane graph G. (b) A rectangular dual of G.

DOI: 10.1685/CSC09301

Not every plane graph has a rectangular dual. Necessary and sufficient admissibility conditions for rectangular dualization were proved independently by Lai-Leinwand [13] and Kozminski-Kinnen [12].

Theorem 2.1 (Lai-Leinwand, 1984). *A graph G admits a rectangular dual if and only if: (a) G is plane, (b) each inner face of G is bounded by a cycle of length 3 (hence, $3-$cycle) and (c) G contains no separating triangles.*

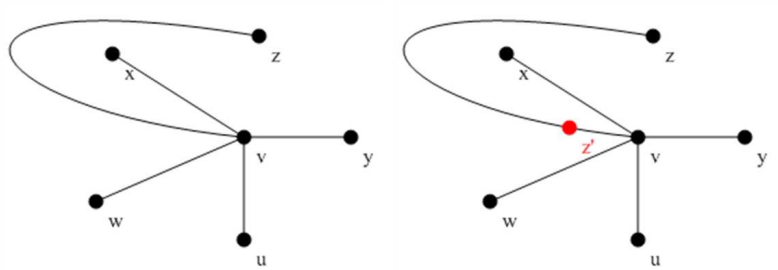
A *separating triangle* is a $3-$cycle which is the boundary of no face. A graph complying with the conditions stated by Theorem 2.1 is called *rectangular graph*.

3. OcORD Outlook

With respect to the previous version [2], OcORD was enriched with several interesting features. Firstly, a graph editor, which makes easier the creation of new graphs, has been developed (Sect. 4). At the current stage, the editor only features basic drawing tools and can not compete with already available tools, such as GraphViz. Secondly, OcORD uses a proprietary XML language to encode the graphs created with the editor (Sect. 5). Third, a full linear-time algorithm for rectangular dualization has been implemented (Sect. 6). Fourth, OcORD provides an effective tool for drawing a graph, given its rectangular dual (Sect. 7). Finally, it creates the basis for the visualization of Virtual Worlds based on the 3D Electronic Institution metaphor (Sect. 8).

4. The Graph Editor

The OcORD editor is composed of a drawing area and a toolbar providing basic facilities for drawing graphs. The editor provides for two working modes. The *drawing mode* allows nodes and edges to be drawn; in the *selection mode*, the appearance of nodes and edges can be modified as well as their position and shape. Edges are drawn as Bezier curves, which are widely used in computer graphics, as they appear reasonably smooth at all scales, as opposed to polygonal lines. At a first glance, the use of curvilinear edges may appear a kind of unneeded whim; after all, planar graphs admit a straight-line plane embedding. However, a straight-line drawing requires the nodes to be appropriately placed in the drawing area, in order to prevent edge crossings. Typically, one realizes that the position of a node is wrong only when two edges cross. Thus, if curvilinear edges were not allowed, edge crossings could be eliminated only by moving (potentially all) vertices.



(a) (b)

Fig. 2. (a) u precedes w in clockwise order around v and is on the left of the straight line directed from v to w. The same does not hold for z and x. (b) Edge (v, z) is approximated with a straight line (v, z').

4.1. *How Does OcORD Conceive a Drawing?*

Once OcORD has been given a drawing of a graph, a procedure is needed to "translate" the drawing into a set of adjacency lists. Each vertex v, indeed, is associated with an adjacency list $Adj(v)$, containing the neighbours of v sorted in clockwise order around v. In Fig. 2 (a) u precedes w in clockwise direction around v; this means that u in on the left of the straight line directed from v to w. This invariant does not hold for x and z; in fact, z precedes x, but lays on the right of the line (v, x). To fix this situation, the curvilinear edge (v, z) is approximated with a straight edge (v, z') (Fig. 2 (b)). Thus, $Adj(v)$ is obtained by radially sorting around v its neighbours; using Merge Sort, the creation of all adjacency lists takes $O(|N(v)| \log |N(v)|)$ time, where $N(v)$ denotes the set of the neighbours of v.

5. Input Format

Lots of words have been spent to (rightly) praise XML and its virtues. Jon Bosak, who led the creation of XML, claimed that "XML is the digital dial tone of the Web"; Peter Murray Rust (Unilever Centre for Molecular Sciences Informatics) added: "I assume that there are now (or soon will be) chips that are XML-aware. I love it.". These two colourful quotations define XML better than thousands words. Born to satisfy the demands for a more flexible mark-up language to complement HTML, it is now widely used in a number of different applications.

No standard XML language for graphs has been defined yet. Nevertheless, literature counts some remarkable works, including, in particular, GXL and GraphML [5,15]. Due to the lack of standards, we opted for creating a simple language, tailored on OcORD requirements. A graph in OcORD is

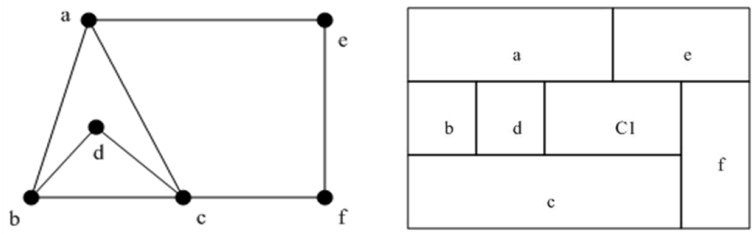
DOI: 10.1685/CSC09301

a collection of clusters; each is described using the XML element <graph>. Elements <node> and <edge> are used to describe nodes and edges respectively through a large set of attributes. Finally, element <intercluster> allows the definition of the edges whose endpoints belong to different clusters.

6. Basic Algorithm

The core of OcORD is an algorithm which creates a rectangular dual of any plane graph, after enforcing the conditions stated in Theorem 2.1. This algorithm improves the one described in [2] to run in linear-time. Given a plane graph G, the following steps are executed:

1. **Biconnectivity Test and Augmentation**. If G is not biconnected, then some edges are added to make it biconnected.
2. $4-$**completion**. Four vertices $North$, $West$, $South$ and $East$ are added to form a new exterior cycle of G. Edges are added to keep G biconnected.
3. **Separating triangle search and break**. Any separating triangle of G is eliminated ("broken", to follow the notation used in [13]).
4. **Triangulation**. The inner faces of G are triangulated in linear time [3]. After this step, G is a rectangular graph.
5. **Rectangular dualization**. A rectangular dual of G is created, using the linear-time algorithm described in [11]. This algorithm requires G to have an exterior $4-$cycle. This explains the $4-$completion (Step 2).



(a) (b)

Fig. 3. (a) A plane graph G with one separating triangle $\Delta = (a, b, c)$. (b) The rectangular dual of G. C_1 is a gate, corresponding to the crossover vertex added on edge (a, c) to break Δ. Rectangles a and c are adjacent through C_1. Rectangles a and d are adjacent, although their corresponding vertices are not.

The output of the algorithm $\mathscr{R}(G) = (\Gamma, f)$ is not properly the rectangular dual of G (Fig. 3). In fact, some rectangles may turn out to be adjacent, though the corresponding vertices are not. However, if two vertices

are adjacent in G, the corresponding rectangles are adjacent too, provided that G has no separating triangles. Indeed, breaking a separating triangle $\Delta = (u, v, w)$ is accomplished by removing one of its edges (say (u, v)), adding a new vertex c (called *crossover vertex*) and linking c to u and v. Thus, because of the crossover vertex, u and v are not adjacent anymore. Consequently, in $\mathscr{R}(G)$ the rectangles $f(u)$ and $f(v)$ will be adjacent only through rectangle $f(c)$ (called *gate*).

6.1. *Biconnectivity Augmentation*

Augmenting G to a biconnected graph with a minimum number of edges is a well-known NP-hard problem [10]. However, although our aim is to modify G as little as possible, minimizing the number of additional edges is not our primary concern. Any new edge,in fact, does not affect the area of the rectangular dual. For this reason, we chose the simple algorithm described in [14], with the improvements proposed in [10]. The idea is trivial: if u and w are consecutive in $N(v)$ and belong to different biconnected components, then edge (u, w) is added. This procedure is applied to all consecutive neighbours of all vertices of G. Kant remarked that in the worst case the degree of a single vertex may receive $O(n)$ new incident edges. Thus, he proposed to change the embedding of G, so that all neighbours of v, belonging to the same biconnected component, are consecutive in $N(v)$. In this way, the algorithm still runs in linear time and each vertex receives at most 2 extra edges.

6.2. *Finding Separating Triangles*

A separating triangle is a $3-$cycle which is not a face. In [6] a simple nice linear-time algorithm for finding all $3-$cycles is described. The nodes of G are sorted by decreasing degree, which can be accomplished using Bucket Sort, a well-known linear-time sorting algorithm. For each node v the following operations are performed: first, the neighbours of v are marked, then, for each marked node w, if w is adjacent to a marked node u, then (v, w, u) is a $3-$cycle. After visiting all its neighbours, v is removed from G, so that any separating triangle is found only once.

Each vertex of G can be assigned a natural number; consequently, we can assume that each triple (v, w, u) in L is such that $v < w < u$. These triples can be lexicographically sorted using Radix Sort, another linear-time sorting algorithm. Finally, a list F of $3-$faces is created and lexicographically sorted in the same way. We remark that finding all faces of a plane graph can be trivially accomplished in linear time. Since L and F are sorted, they can be searched in parallel to remove from L any item of F.

DOI: 10.1685/CSC09301

6.3. *Breaking Separating Triangles*

Minimizing the crossover vertices is important, as they increase the area of the rectangular dual of G. The first version of OcORD implemented a $O(n^3)$ time algorithm [1]. However, since we use rectangular dualization also to create graph drawings and most of the drawing algorithms are linear-time, including this algorithm in OcORD is not the best choice.

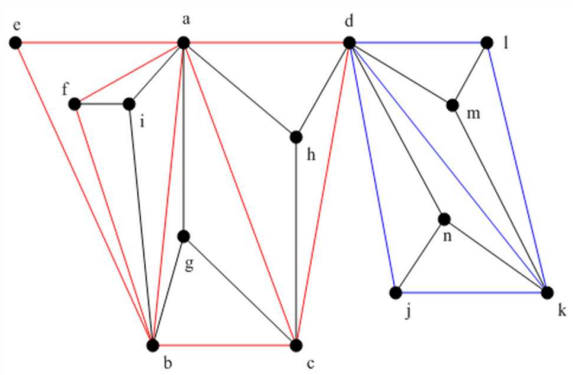


Fig. 4. A graph with two islands (highlighted with two colours)

We observe that separating triangles do not occur frequently in plane graphs. It was estimated in [16] that in randomly generated plane inner triangulated graphs (PTG)[a], the expected number of separating triangles is approximately 16% of the number of nodes. We confirmed this figure using a simple random graph generator included in OcORD. Since PTGs are the most dense planar graphs, we can conclude that the number of separating triangles in a planar graph is at most 16% of the number of nodes. Moreover, if all separating triangles of G are *independent* (no two of them share an edge), the solution to the problem is trivial, as a crossover vertex for each triangle must be added. Thus, the complexity of the problem does not strictly depend on the overall number of separating triangles, but on the size of the biggest *island*. An *island* (of separating triangles) is a maximal group of triangles which are pairwise adjacent (Fig. 4). Using the OcORD graph generator, we created 10000 PTGs, with order ranging from 100 to 1000 nodes, and verified that the island size is on average 2 and in the worst case is 21. Due to all these properties, we decided to tackle the problem implementing a set of linear-time heuristic algorithms. We are currently evaluating their performances on randomly generated graphs.

[a] A PTG is a graph whose inner faces are bounded by 3−cycles

7. Drawing Graphs with Rectangular Dualization

Graph Drawing has emerged as an important part of graph theory since the 80s. A number of different applications, ranging from circuit design to software engineering, take advantage from graph visualization. Graph Drawing aims at developing efficient algorithms for creating "nice" drawings of a graph. "Nice" refers, but is not limited to, aesthetic; a drawing pleasant to see, in fact, improves the understanding of the information that the graph wants to express.

In our knowledge, rectangular dualization has never been considered as a graph drawing tool. This is partly due to the lack of efficient (linear time) algorithms as well as to the admissibility conditions, which restrict the scope of rectangular dualization to a small class of graphs. In OcORD, rectangular dualization is used to create orthogonal drawings and bus-mode drawings of plane graphs.

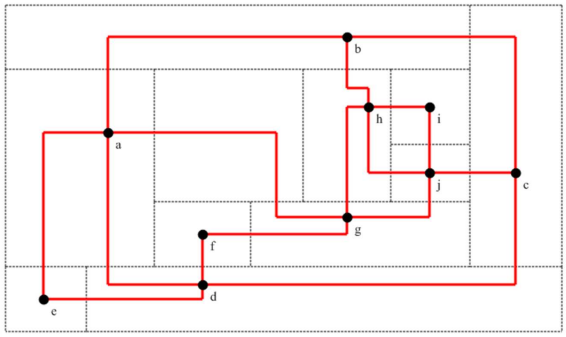


Fig. 5. An orthogonal drawing of a graph. The rectangle having no vertex inside is a gate through which vertices a and g are adjacent.

In an orthogonal drawing, each edge is drawn as a chain of horizontal and vertical segments (Fig. 5). This drawing style is particularly attractive, as it maximizes the *angular resolution*, that is the maximum angle between two adjacent edges. On the other hand, it can be used only with graphs with maximum degree 4. OcORD creates an orthogonal drawing computing an orthogonal transform for each rectangle in the rectangular dual. An *orthogonal transform* of a rectangle R is a pair $T_R = (p, \mathscr{H})$; p is a point enclosed in R representing the node v of G corresponding to R; $\mathscr{H}$ is a set of as many polygonal chains as the adjacencies of R. The endpoints of each polygonal chain in $\mathscr{H}$ are p and one point on the portion of edge (called *window*) R shares with an adjacent rectangle (Fig. 5).

A *bus-mode drawing* extends the orthogonal drawing to graphs with

DOI: 10.1685/CSC09301

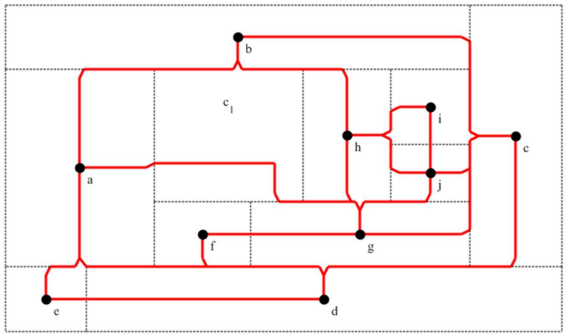


Fig. 6. An bus-mode drawing of a graph.

arbitrarily high degree [2]. The effectiveness of an orthogonal drawing rely on the fact that the angular resolution is $\frac{\pi}{2}$, which makes it readable and pleasant to see. A bus-mode drawing maintain this angular resolution by grouping the edges incident with a vertex into up to 4 *bundles* (Fig. 6). Since each bundle may contain several edges, it may not be clear whether two nodes are adjacent or not. To eliminate the ambiguity, the bundles are provided with curved corners which indicate the path of each edge. Each bundle is like a bus which conveys a group of wires between two electronic devices; hence, the name of this drawing style. To create a bus-mode representation from $\mathscr{R}(G)$, OcORD places a node v at the center of each rectangle $f(v)$. If two points have the same $x-$ or $y-$coordinate up to a predefined constant $\epsilon > 0$, the position of one is modified so that they can be joined with a straight edge. For each point p_v, inside rectangle $f(v)$, at most four bundles, leaving p_v and running parallel to the $x-$ and $y-$axis, are created, based on the following rules:

- An upward bundle, if and only if at least one neighbour of $f(v)$ is above $f(v)$.
- A leftward bundle, if and only if at least one neighbour of $f(v)$ is to the left of $f(v)$.
- A downward bundle, if and only if at least one neighbour of $f(v)$ is below $f(v)$.
- A rightward bundle, if and only if at least one neighbour of $f(v)$ is to the right of $f(v)$.

As said before, each bundle is terminated by at most two curved corners, which indicate the direction that the bundle follows when leaving $f(v)$. If the rectangles $f(u)$, such that $u \in N(v)$, can be reached following only

one direction, the bundle has only one curved corner. After leaving $f(v)$, a bundle runs along the sides of the rectangles and enters a rectangle (through a curved corner) only to reach an endpoint.

8. Visualization of Electronic Institutions

3D Electronic Institutions are a new method of software design of open systems based on the metaphor of 3D Virtual Worlds [7]. One of the drawbacks of the Virtual Worlds technology is that its design and development has emerged as a phenomenon shaped by a home computer user, rather than by the research and development in universities or companies. Thus, Virtual Words do not have the means to enforce technological norms and rules of their inhabitants. The enforcement of organizational conventions in 3D Electronic Institutions methodology is achieved by separating different patterns of conversational activities into separate methodological entities (scenes), assigning different roles to different types of participants, specifying the rules (protocols) for inter-participant interactions and defining the role flow of participants between different scenes. The specification of scenes and the role flow are done in a form of a directed graph, where nodes represent scenes and arcs and their labels define the role flow. This graph, called *Performative Structure*, forms a basis for the visualization of the system. The institution, in fact, is visualized as a 3D virtual building and the rectangular dual of the Performative Structure is the floorplan of it.

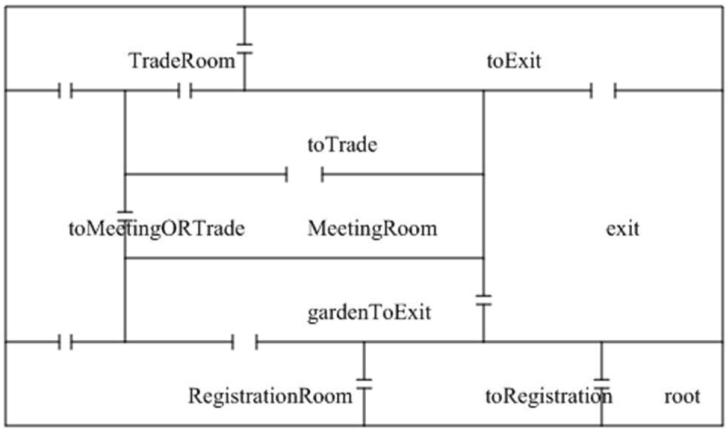


Fig. 7. Floorplan of an Electronic Institution. Any two adjacent rooms are linked through *doors*.

An Electronic Institution is specified using a software called ISLANDER

DOI: 10.1685/CSC09301

[7]. The output of ISLANDER is an XML file, which defines the roles allowed within the institution, the language that agents use to interact and the performative structure, that is the graph representing the institution. OcORD obtains from this file the embedding of the Performative Structure and imports it into its graph editor. Such an embedding may not be plane. Generally, a Performative Structure may even not be planar. Thus, OcORD first checks whether the given embedding is plane; if it is, the drawing is immediately displayed. If some edges cross, the Boyer-Myrvold algorithm [4] is used to determine whether the graph is planar and, if so, to compute a plane embedding. Finally, a drawing of the new embedding is created using rectangular dualization. Fig. 7 shows a floorplan of an Electronic Institution.

9. Conclusions and Future Work

In this paper, we shortly described our advances in rectangular dualization. With respect to the previous versions, OcORD has been enriched with interesting features. The algorithm for rectangular dualization is almost complete; we are currently evaluating a set of heuristics for breaking separating triangles in linear time. Moreover, we are investigating other algorithms for drawing graphs from their rectangular dual. In particular, we will extend rectangular dualization to hierarchically clustered graphs.

REFERENCES

1. A. Accornero, M. Ancona, and S. Varini, All Separating Triangles in a Plane Graph Can Be Optimally "Broken" in Polynomial Time, *International Journal of Foundations of Computer Science*, **11/3** (2000), pp. 405-421.
2. M. Ancona, S. Drago, G. Quercini, and A. Bogdanovych, Rectangular dualization of biconnected plane graphs in linear time and related applications, *Applied and Industrial Mathematics in Italy II* (2007), pp. 37-48.
3. T. Biedl, G. Kant, and M. Kaufmann, On triangulating planar graphs under the four-connectivity constraint, *Lecture Notes in Computer Science*, **824** (1994), pp. 83–94.
4. J. Boyer, and W. Myrvold. Stop minding your P's and Q's: a simplified O(n) planar embedding algorithm . In *Proceedings of the tenth annual ACM-SIAM symposium on Discrete algorithms*, pp. 140–146. SIAM, Philadelphia, 1999.
5. U. Brandes, M. Eiglsperger, I. Herman, M. Himsolt, and M. S. Marshall,

GraphML Progress Report. Structural Layer Proposal, *Lecture Notes in Computer Science*, **2265** (2002), pp. 109–112.

6. N. Chiba and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.*, **14/1** (1985), pp. 210–223.
7. S. Drago, A. Bogdanovych, M. Ancona, S. Simoff, and C. Sierra. From Graphs to Euclidean Virtual Worlds: Visualization of 3D Electronic Institutions. In *Thirtieth Australasian Computer Science Conference*. ACS, Ballarat, Australia, 2007.
8. R. Diestel, *Graph Theory*, 3rd ed., Springer-Verlag, Heidelberg, 2005.
9. J. Grason. An approach to computerized space planning using graph theory. In *Proceedings of the 8th workshop on Design automation*, pp. 170–178. ACM, New York, 1971.
10. G. Kant. *Algorithms for drawing planar graphs*. PhD thesis, Department of Computer Science, University of Utrecht, Netherlands, 1993.
11. G. Kant, and X. He. Two Algorithms for Finding Rectangular Duals of Planar Graphs. In *Proceedings of the 19th International Workshop on Graph-Theoretic Concepts in Computer Science*, pp. 396–410. Springer-Verlag, London, 1994.
12. K. Kozminski, and E. Kinnen. An algorithm for finding a rectangular dual of a planar graph for use in area planning for VLSI integrated circuits. In *Proceedings of the 21st conference on Design automation*, pp. 655–656. IEEE Press, Piscataway, 1984.
13. Y. T. Lai, and S. M. Leinwand, Algorithms for Floorplan design via rectangular dualization, *IEEE Transactions on Computer-Aided Design*, **7/12** (1988), pp. 1278–1289.
14. R.C. Read, A new method for drawing a graph given the cyclic order of the edges at each vertex, *Congr. Numer.*, **56** (1987), pp. 31–44.
15. A. Winter, Exchanging Graphs with GXL, *Lecture Notes in Computer Science*, **2265** (2002), pp. 54–58.
16. K. H. Yeap, and M. Sarrafzadeh. A Theorem of Sliceability. In *Proceedings of the Second Great Lakes Computer Science Conference*. Western Michigan University, Kalamazoo, 1991.