RECTANGULAR DUALIZATION OF BICONNECTED PLANAR GRAPHS IN LINEAR TIME AND RELATED APPLICATIONS

M. ANCONA, S. DRAGO and G. QUERCINI

Department of Computer Science (DISI), University of Genoa,
Via Dodecaneso 35, 16146 Genoa, Italy
E-mail: {ancona, drago, quercini}@disi.unige.it
http://www.disi.unige.it

A. BOGDANOVYCH

Faculty of Information Technology, University of Technology Sydney,
Sydney, NSW, Australia
E-mail: anton@it.uts.edu.au
http://www.it.uts.edu.au

Although rectangular dualization has been studied for several years in the context of floorplanning problems, its descriptive power has not been fully exploited for graph representation. The main obstacle is that the computation of a rectangular dual of any planar biconnected graph requires a sequence of non-trivial steps, some of which are still under investigation. In particular, the most tricky issue is the optimal *management* of separating triangles, for which no existing algorithm runs in linear time. In this paper we present our advances in rectangular dualization and we show two applications that, while very different, explain better than others its role.

Keywords: Rectangular Dual; Separating Triangles; Hierarchical Drawing; Structured Graphs; Electronic Institutions; Network Topologies

1. Introduction

Rectangular dualization is the computation of a rectangular dual of a planar graph. It was originally introduced to find rectangular topologies for floorplanning of integrated circuits:[1] by a floorplan, a rectangular chip area is partitioned into rectilinear polygons corresponding to the relative location of functional entities of the circuit. Subsequently, it found application in many other fields, in particular becoming effective in visualization problems. In this paper we briefly describe two possible applications. The first

one is visualization of network topologies, concerning the problem of engineering and optimizing large communication networks. This task is very challenging, as often real networks are huge, including hundreds and even thousands of nodes and links. In order to help human operators in maintaining and updating the description and documentation of its structure, a network is usually described in form of a hierarchy of subnetworks. Rectangular dualization is a very useful technique in graph drawing, especially when applied to the hierarchical drawing of a structured graph.
The second application domain which could benefit from rectangular dualization is the automatic design of Virtual Worlds whose development follows the 3D Electronic Institutions methodology,[2] which helps to separate the development into two independent phases: specification of the interaction rules and design of the 3D Interaction environment. During the specification phase not only the interactions rules are specified, but also the basic interaction components are determined. One part of the specification is the graph, which describes which scenes are required in the system (nodes of the graph), shows how the transitions between scenes are made (nodes) and which scenes can be reached from another scenes (arcs). By rectangular dualization of this graph, a two dimensional map of the Virtual Worlds is achieved and then scenes and transitions become 3-dimensional rooms, while the arcs of the graph determine which rooms have to be placed next to each other and have a shared door. The use of rectangular dualization is strongly limited by the fact that not all planar graphs admit a rectangular dual. However, it is possible to apply a minimal set of transformations to the original graph to obtain a graph admitting a rectangular dual representation. In the following sections we first describe an algorithm which computes the rectangular dual of any planar graph (section 2); then we outline the advantages of rectangular dualization in the two applications mentioned above (sections 3 and 4). In section 5 an overview of the paper and of future work is given.

2. OcORD: Optimal Constructor of a Rectangular Dual

A rectangular dual of a planar graph $G = (V, E)$ is a rectangle R partitioned into a set $\Gamma = R_1, ...R_n$ of non overlapping rectangles such that:

(1) no four rectangles meet at the same point;
(2) there is a one-to-one correspondence $f : V \to \Gamma$ such that two vertices u and v are adjacent in G if and only if their corresponding rectangles $f(u)$ and $f(v)$ share a common boundary.

Graphs not admitting a rectangular dual contain separating triangles, triangular regions of the graph that are not faces.[3–5] The idea behind OcORD, our tool aiming at constructing the rectangular dual of any planar graph in linear time, is to remove (we will use the term "break" from now on) all the separating triangles, if any, from the input graph, to obtain a graph admitting a rectangular dual. This can be accomplished by adding a *crossover* vertex on one edge of each separating triangle, as Lai and Leinwand proposed in.[6] However, their approach does not take care of adding a minimal set of crossover vertices, as they conjectured it was a NP-complete problem. In fact, it is possible to break two or more separating triangles by adding only one crossover vertex on an edge shared by all of them, instead of adding a vertex for each separating triangle. If a minimal set of crossover vertices is added, then we say that the breaking of separating triangles is optimal. In OCoRD this task is performed in five steps:

(1) four external vertices are added according to.[5]
(2) All the separating triangles are detected with the algorithm described in[7] and the geometrical dual of the resulting graph is computed.
(3) All the separating triangles are collapsed into macro-vertices, which creates a hierarchical structured graph (section 3.3).
(4) A crossover vertex is added on the duals of the edges belonging to a minimum covering. In[8] a formal proof that the number of crossover vertices added is minimum is given.
(5) The resulting graph is triangulated with the algorithm described in[9]

The graph obtained after these transformations admits a rectangular dual, that can be computed in linear time with several algorithms.[5,10] However, this procedure does not run in linear time and is fairly complicated, as it requires structuring the graph in step 3. Instead, we are investigating an algorithm that works on the plain graph and exploits matching in cubic graphs, that can be computed in linear time, as shown in.[11] Moreover, we have also developed a heuristic method that optimally breaks all the separating triangles in almost all the practical cases. We reserve to describe it in future work.

3. Visualization of Network Tolopogies

The plainest way to describe a communication network is to model the relationships among sites and links by means of a weighted undirected graph, but unless some more assumptions are taken, this approach can raise several problems. Problems are encountered when networks have a hierarchical topology, where nodes are classified in levels, usually two or three, according to their geographical position, number of users, and so on. Moreover, the optimization of large communication networks (for example the minimization of the number of message hops, i.e. the number of graph nodes traversed by a message) requires the use of special methods which need a most suitable graph representation. Many issues are encountered during network design phase, when repeated analysis and visualization of the network has to be performed: practical networks include hundreds or often thousands of nodes and links, so that even a simple description and documentation of the network structure is hard to maintain and update. In this case the network is usually described in form of a hierarchy of subnetworks that are represented by collapsing some subnetworks into single nodes or single links to be described in separated documents. Such a hierarchical approach to network (and graph) description can be formalized into a complex but flexible graph structure called *structured graph.* In the following sections we stress the importance of rectangular dualization in hierarchical graph drawing.

3.1. *Role of Rectangular Dualization in Network Drawing*

Using a rectangular dual in graph drawing offers several advantages:

(1) Its computational complexity is optimal: $O(n)$ time and $O(n^2)$ area, like other most efficient methods.
(2) It provides a symmetrical drawing with respect to x and y coordinates.
(3) It can be naturally extended to a hierarchical drawing of a structured graph G, by introducing the concept of *structured rectangular dual* (see section 3.3)
(4) The construction of a 2-visibility drawing from a rectangular dual is immediate (see Figure 1).
(5) A 2-bend[a] rectilinear drawing can be obtained from a rectangular dual in linear time.
(6) Also a 1-bend rectilinear drawing can be obtained from a rectangular dual in linear time.

The 2-visibility drawing method has been introduced by Kant.[12] In a 2-visibility drawing the vertices of a given graph are represented by rectangles (rectangular boxes) and the adjacency relationships are expressed by

[a]A bend is a point where the drawing of an edge changes its direction. A drawing is said to be k-bend if each edge has at most k bends

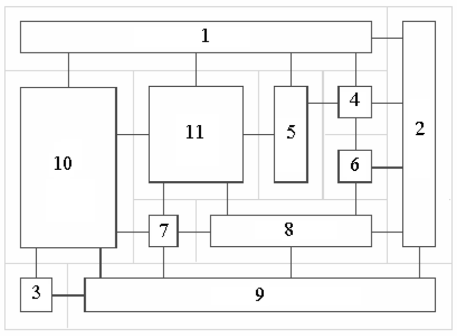


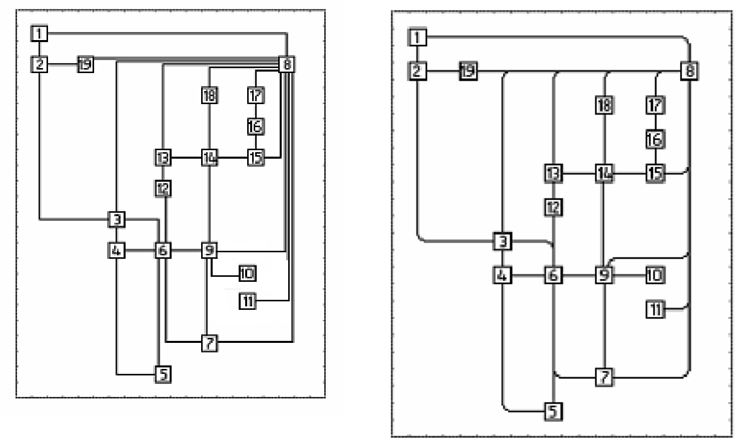
Fig. 1. A 2-visibility representation computed from a rectangular dual (depicted in the background)

horizontal and vertical lines drawn between boxes. The authors claim a high quality of the drawing. This kind of drawing can be trivially derived from the rectangular dual of a graph as follows. Two adjacent rectangles $R1$ and $R2$ of the dual graph share a portion of an edge that we call *window*. If inside $R1$ and $R2$ we draw two smaller rectangles large enough to be mutually visible through the window, they can be connected by a straight segment. The result of this procedure is shown in figure 1.

3.2. *Drawing Modes and Styles*

In this section we distinguish two concepts: *drawing mode* and *drawing style*. A drawing mode defines the form of edge and vertex representation adopted: in *straight-line* drawing mode edges are drawn as line segments connecting vertices; in *orthogonal drawing* the edges are drawn as sequences of segments parallel to x or y axes; in bus mode drawing (Figure 2) bundles of parallel edges are collected into a single path (or bus) until each edge emerges from the bus, and reaches its destination vertex (in some cases each bundle is identified by a label). Drawing modes for vertices include points, squares, rectangles or circles of fixed or variable size.

A drawing style is the combination of a drawing mode with a specific algorithm displaying data. In other words, a drawing style is concerned with relative positioning of vertices and edges in the plane, whereas a drawing mode with the form of representing edges and vertices. For example, in grid drawing style, vertices and edges are drawn only on a discrete grid in the plane; in the Kandinsky model (also known as *Podevs*), and in some of its variants, like the simple Kandinsky model (*Podevsnef*), vertices are represented by squares of equal size, placed on a coarse grid and edges are drawn, in orthogonal mode, on a finer grid.



(a) A classical drawing (b) A bus mode drawing

Fig. 2. Classical drawing and bus mode drawing of a given graph

The bus drawing mode is useful for undirected graphs, since the arrowheads would introduce ambiguities. In figure 2 Kandinsky drawing (figure 2(a)) and bus-mode drawing (figure 2(b)) of the same graph are compared. The latter, although it is 2-bend while the first is 1-bend, is more readable, as edges are not too close each other. To create a bus style drawing, we simply have to ignore the offsets of the edges on the finer grid, and transform each bend into a curved corner. Despite its evident utility (see schematics drawing tools like *Orcad*) this mode has not been considered by researchers in graph drawing. In this paper we use only bus-mode layouts because it is particularly useful for connecting vertices of high degree, a frequent case in clustered graphs. Bus-mode drawing is a powerful way for structuring and clustering edges of a graph. Buses represent a structured set of interconnections: while a sub-graph can be represented by a single macro-vertex, a set of edges contributing to the same logical function can be condensed into a single bus, that can be considered as a special form of hyper-edge. Our algorithm for constructing the dual graph assigns integer coordinates to each rectangle associated with a vertex. In this way, by scaling the rectangle sizes we can allocate space both for squares representing vertices and zones dedicated to interconnections. It is a trivial task to derive a bus-mode rectilinear drawing of a graph G from its rectangular dual. Such a drawing could be based on paths all composed by exactly two bends with a pleasant and clear visual effect. We call this kind of drawing the naive 2-bend bus-mode drawing. However, with a minimum effort, we can produce a 1-bend (maximum) drawing in linear time.

3.3. *Hierarchical Drawing of Structured Graphs*

Managing large networks requires the decomposition of the network in manageable units organized in a hierarchy (tree) describing how single parts contribute to form the original network. A *structured graph* (or *clustered graph*, henceforth referred to as "SG") is a form of abstraction applied to a large graph in order to make it modular and more manageable. The abstraction consists in collapsing a subgraph to a single vertex (called a *macrovertex*), or to a single link (called a *macrolink*), thus obtaining a simpler and hierarchically described graph. The structuring operation is usually iterated recursively until a large network is decomposed into relatively small and manageable components and subcomponents. Every subcluster is defined at several levels of nesting adopting a methodology that is usually applied to every large project (software and hardware design) involving hundreds or thousands of components: "modularity". Figure 3 shows an example of a structured graph. We will refer to the graph represented in figure as "gd94".

Formally, a SG is a pair $H = (G, T)$, where G is a connected simple (multi- or hyper-) graph and T is a tree describing the structure of H. The leaves of T are exactly the vertices of G and each node $t \in T$ represents a cluster $G_t = (V_t, E_t)$ of G such that $V_t \subseteq V$ is the subset of vertices that are leaves of the subtree of T rooted at t. G_t is the subgraph generated by V_t, while $H_t = (G_t, T_t)$ is the SG associated with t. Notice that the planarity of the underlying base graph does not imply the planarity of a structured graph if arbitrary subgraphs are collapsed. In order to preserve planarity, the collapsing operation should respect some conditions. Some researchers require H to be *c-planar*,[13] namely there exists a planar embedding of H in the plane, with a planar drawing such that no region R surrounding a macro-element is crossed by an edge having both vertices external to it. Instead, we give the following conditions:

(1) The original graph must be planar.
(2) Only complete and connected subgraphs are collapsed.
(3) Elements of the same kind (namely macro-vertices or macro-links) must be completely nested or independent, thus having an empty intersec-

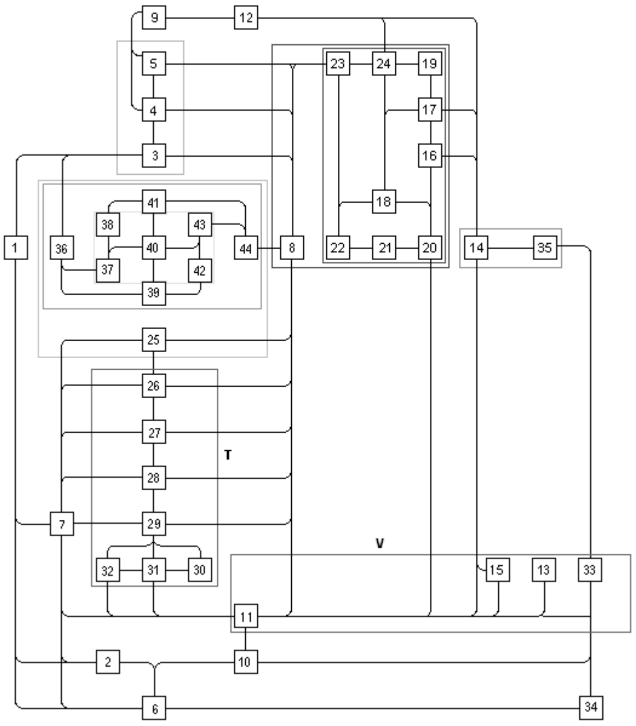


Fig. 3. Example of a structured graph (gd94), where each cluster is surrounded by a rectangle

tion.

(4) Different macro-elements (vertices and/or links) may share only vertices and not one or more links.

and we report the following well-known results.

Definition 3.1. A graph G' is said to be a subdivision of a graph G if G' is obtained from G by replacing some of its edges with paths having at most their endvertices in common.

Theorem 3.1 (Kuratowski, 1929). *A graph is planar if and only if it does not contain a subdivision of K_5 and $K_{3,3}$.*

Now we can prove the following:

Theorem 3.2. *In relation with the above definition of a structured graph,* *we have that if only connected subgraphs are collapsed, the resulting structured graphs are all planar.*

Proof. By absurd, suppose that there is a non planar clustered graph G' of a planar graph G. For Kuratowski's theorem G' contains a subdivision K of a K_5 or $K_{3,3}$. G' must contain at least a macro-vertex because G is planar. Let m be a macro-vertex of G' and Gm the corresponding clustered subgraph in G; by the connectivity of G_m follows that there is a path in G_m joining the two ports of m on which K is defined in G', that is, there exists a subdivision K'' of G, that is a contradiction. □

3.4. ***Hierarchically Structured Rectangular Duals***

Drawing a structured graph, or even drawing a medium-sized graph, requires to fit the available page size for a drawing, i.e., to decompose the representation into sheets and to refer them to the original graph. Either for structured graphs, often the size of a single component does not comply with visualization requirements. In this case, a visualization-oriented decomposition should be preferred. A hierarchical structure of a graph can be extracted from its rectangular dual. To this purpose we introduce the concept of *hierarchically structured rectangular dual* or, more simply, *hierarchical rectangular dual* (HRD) of a graph G. The HRD of a graph G is a tree of rectangles (R, T) such that the rectangles of the dual graph represent either single vertices, like in a standard rectangular dual, or connected subgraphs of G, abstracted to a single vertex (see, for example, rectangles violet (V) and turquoise (T) of figure 3) and called *macro-rectangles.*

In figure 3 clusters are obtained from a rectangular dual computed on "gd94" using OcORD. Rectangles in the figure are rectangles in the rectangular dual that have other rectangles inside, meaning that they do not represent vertices, but subgraphs. We are far from saying that rectangular dualization is a method for clustering graphs, as clustering is not only a topological issue, but has to take into account of many other factors. However, if we restrict our attention only to drawing, clusters suggested by a rectangular dual seem to be very efficient.

3.4.1. *Computing a HRD*

In the previous section we have described how a HRD suggests a clusterization of the primal graph. It is also possible to perform the inverse step, namely computing a HRD from a planar structured graph fulfilling conditions listed in section 3.3. The idea is the following: we build the rectangular dual R, where each cluster C_i is depicted as a rectangle R_i; for each cluster C_i we compute its rectangular dual and we draw it inside R_i. This procedures preserves the adjacencies between the macro-vertices, without caring about adjacencies between simple vertices lying in different clusters. Thus, the adjacency between two vertices does not necessarily imply the adjacency of the two corresponding rectangles in the rectangular dual. In order to fulfil this second constraint, each cluster must specify an interface to the other clusters. More in details, if A and B are two adjacent clusters, a vertex i (called *interface vertex*) is added so that each edge (a, b), $a \in A$ and $b \in B$, is split into two edges (b, i) and (a, i). Each cluster is then surrounded by the interface vertices. Thus, we compute the HRD of this graph and we draw the rectangular dual of each cluster inside the corresponding rectangles. However, this procedure raises several problems, in terms of efficiency of the final drawing, and this is the main reason why here we give only a brief overview. First of all, it is not clear how many interface vertices have to be added: an interface vertex for each adjacency of a cluster does not seem to be an optimal choice. Next, when computing the rectangular dual of clusters, we consider it having its interface vertices on the external face. Thus, an interface vertex between cluster A and cluster B appears on the external face (and in the rectangular dual) of both clusters, meaning that they are replicated. This topic will be object of further analysis in our research.

4. Representation of 3D Electronic Institutions

In this section we only introduce in short an application that, while being very different from network visualization, demonstrates the power and versatility of rectangular dualization. For more details the interested reader can refer to.[14]

3D Electronic Institutions are a new method of software design of open systems based on the metaphor of 3D Virtual Worlds. 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. Rectangular dualization plays its central role in automatically transforming the Performative Structure into a 3D Virtual World. In fact, the rectangular dual of the Performative Structure is a 2D map of the institution, which is further transformed into a 3D Virtual World. In such a process, rectangular dualization offers a space optimal solution, as it can be used for minimizing the distance between two agents (entities in the virtual world) that are expected to have frequent interactions. The automatic generation of a 3D Virtual Word consists of the following steps:

(1) The redundant information contained in the Performative Structure is filtered out. If some nodes of the graph are connected with more than one arc, only one randomly selected arc is left and all the others are deleted.
(2) The Performative Structure is transformed into a format compatible with OcORD input and a rectangular dual is computed. In the rectangular dual, scenes and transitions are transformed into rooms and connections are visualized as doors.
(3) A 3D Virtual World is generated from the 2D map created at the previous step. The generated 3D Virtual World is visualized and connected to the infrastructure, which controls the correct behaviour of the participants.

5. Conclusions and Future Work

In this paper we presented the main reasons behind our interest in rectangular dualization, describing some useful applications that benefit from this technique. Some work is still to be done in developing an efficient and linear-time algorithm for computing the rectangular dual of every planar graph. Moreover, the computation of a HRD of a structured graph is far from being efficient. Finally, rectangular dualization is only a particular case of a more general class of rectangular representations, called rectangular layouts. Most of them show the same problems arising in rectangular dualization and have a similar impact in practical applications (think about cartograms, for example). Our aim is also to extend our results to these layouts.

References

1. J. Grason, An Approach to Computerized Space Planning Using Graph Theory, in *Proceedings of the 8th Workshop on Design Automation (DAC '71)*, (ACM Press, New York, NY, USA, June 1971).
2. A. Bogdanovych, M. Esteva, S. Simoff, C. Sierra and B. Helmut, A Methodology for 3D Electronic Institutions, in *Proceedings of the Sixth International Joint Conference on Autonomous Agents and Multi-Agent Systems (AAMAS '07)*, (Springer-Verlag, Honolulu, USA, May 2007).
3. 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 (DAC '84)*, (IEEE Press, Piscataway, NJ, USA, June 1984).
4. S. M. Leinwand and Y. T. Lai, An algorithm for Building Rectangular Floor-Plans, in *Proceedings of the 21st Conference on Design Automation (DAC '84)*, (IEEE Press, Piscataway, NJ, USA, June 1984).
5. 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 (WG93)*, (Springer-Verlag, London, UK, June 1994).
6. Y. T. Lai and S. M. Leinwand, Algorithms for Floorplan Design via Rectangular Dualization, *IEEE Trans. on CAD of Integrated Circuits and Systems* **7**, 1278 (1988).
7. N. Chiba and T. Nishizeki, Arboricity and Subgraph Listing Algorithms, *SIAM Journal on Computing* **14**, 210 (1985).
8. 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**, 405 (2000).
9. T. Biedl, G. Kant and M. Kaufmann, On Triangulating Planar Graphs under the Four-Connectivity Constraint, *Algorithmica* **19**, 427 (1997).
10. J. Bhasker and S. Sahni, A Linear Algorithm to Find a Rectangular Dual of a Planar Triangulated Graph, in *Proceedings of the 23rd ACM/IEEE Conference on Design automation (DAC '86)*, (IEEE Press, Piscataway, NJ, USA, June 1986).
11. T. Biedl, B. Prosenjit, E. Demaine and A. Lubiw, Efficient Algorithms for Petersen's Matching Theorem, *Journal of Algorithms* **38**, 110 (2001).
12. U. Foßmeier, G. Kant and M. Kaufmann, 2-Visibility Drawings of Planar Graphs, in *Proceedings of the International Symposium on Graph Drawing (GD '96)*, (Springer-Verlag, London, UK, September 1997).
13. Q. Feng, Algorithms for Drawing Clustered Graphs, PhD Thesis, Department of Computer Science and Software Engineering, The University of Newcastle, Australia, (Newcastle, Australia, 1997).
14. A. Bogdanovych and S. Drago, Euclidean Representation of 3D Electronic Institutions: Automatic Generation, in *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '06)*, (ACM Press, New York, NY, USA, May 2006).