Planarization With Fixed Subgraph Embedding

Christoph Buchheim[1], Michael Jünger[1], Maria Kandyba[2],
Merijam Percan[1], and Michael Schulz[1]

[1] Department of Computer Science, University of Cologne
{buchheim,juenger,percan,schulz}@informatik.uni-koeln.de
[2] Chair for Algorithm Engineering, Department of Computer Science, University Dortmund
maria.kandyba@cs.uni-dortmund.de

Abstract. The visualization of metabolic networks using techniques of graph drawing has recently become an important research area. In order to ease the analysis of these networks, readable layouts are required in which certain known network components are easily recognizable. In general, the topology of the drawings produced by traditional graph drawing algorithms does not reflect the biologists' expert knowledge on particular substructures of the underlying network.
To deal with this problem we present a constrained planarization method—an algorithm which computes a graph layout in the plane preserving the predefined shape for the specified substructures while minimizing the overall number of edge-crossings.

1 Introduction

This paper deals with the problem of visualizing of metabolic networks which, in general, contain a large amount of data and have a very complex structure. Representing this data as a drawing helps researchers to better understand and analyze the interplay of its components. Such components may be, e.g., the network motifs [1, 16, 17, 19] or well-analyzed substructures. Traditional graph drawing algorithms [3, 13, 14] often do not take into account the expert knowledge of biologists about the structure of such basic network components. Hence, we have the problem of creating a readable network layout in which certain subnetworks are recognizable, having a predefined form. Additionally, layouts with few edge-crossings are required to achieve the readability.

This problem can be formulated as follows: we are given a (non-planar) connected graph $G = (V, E)$, a connected subgraph $G' = (V', E')$ with $V' \subseteq V, E' \subseteq E$ and an embedding $\mathcal{P}'$ of G' with specified outer face f. Let $V'' \subseteq V'$ be the set of nodes on the border of f. The task is to construct a drawing Γ of G that satisfies the following constraints:

(a) The restriction of Γ to G' is a realization of $\mathcal{P}'$.
(b) The nodes and edges not in G' are drawn in the outer face f of $\mathcal{P}'$.

Even if the given graph G is planar, the constraints defined above may not allow an embedding without edge crossings. Hence, the major goal is to minimize the number of edge-crossings over all feasible drawings.

Without loss of generality, we can assume $\mathcal{P}'$ to be planar. If this is not the case, $\mathcal{P}'$ can be extended by replacing all edge crossings by *dummy nodes*. The constraint (b) implies that there are no edge crossings between the edges of $E \setminus E'$ and E'. Hence, we can assume that there is no edge (v, w) with $v \in V \setminus V'$ and $w \in V' \setminus V''$.

One of the popular graph drawing approaches is the *Topology-Shape-Metric* (TSM) approach, originally presented in [2]. The most prominent algorithms based on TSM are algorithms for orthogonal and quasi-orthogonal drawings [3, 15]. Thereby, an embedding of a given graph, i.e., the relative position of its nodes to each other is computed first. In the next step, the shape of the drawing, i.e, the course of the edges, i.p., the number and the corresponding angles of edge bends, is determined. Finally, the exact coordinates of the nodes and bends can be computed.

Since the drawings produced by a TSM algorithm preserve the embedding determined in its first step and since for our problem we only have constraints regarding admissible embeddings, this approach appears suitable in our case.

Common TSM algorithms first compute an embedding in which the number of crossings is as small as possible, by applying the well-known *planarization method*. Using the ideas of this method, we present an algorithm to find an embedding of a given graph that satisfies the constraints defined above. The resulting embedding can then be used for the construction of drawing Γ by applying already existing algorithms for the shape and metric steps, see, e.g., [13] for an overview.

Our paper is organized as follows: first, the planarization method as it is used by the common TSM algorithms is presented. In Section 3 we show how the planarization method can be used to find feasible embeddings for our problem. We then propose an algorithm that realizes this embedding in the plane. In Sections 4 and 5 we discuss algorithms for related problems. Finally, we present a drawing algorithm in Section 6.

2 The Planarization Method

The central idea of the planarization method is to compute first a maximum planar subgraph $H = (V, E(H))$ of G, i.e., a planar subgraph with the maximum number of edges. We then insert the remaining edges $E \setminus E(H)$ so that the number of resulting edge crossings is minimized. These steps imply solving two NP-hard problems:

2.1 Maximum Planar Subgraph Problem

For this problem both exact [12] and heuristic methods [7, 11, 18] exist. The exact method is a branch-and-cut algorithm based on an integer linear programming formulation that uses indicator variables $x_e \in \{0, 1\}$ for all $e \in E$ such that $x_e = 1$ if and only if $e \in E(H)$. One of the heuristic methods uses a greedy approach: we start with an empty graph $H' = (V, \emptyset)$. Then, edges $e \in E \setminus E(H')$ are iteratively added to H', testing the resulting graph for planarity in each such iteration. If the resulting $H' + e$ is not planar, e is not inserted into H'. The algorithm finally outputs the subgraph $H := H'$. Clearly, the quality of this subgraph highly depends on the order in which the edges are added to H'. Planarity testing can be done in linear time [5, 6].

2.2 Crossing-Optimal Edge Insertion

The edges $I = E \setminus E(H)$ now have to be added to H so that the total number of edge crossings in the final embedding of G is minimized. For a single edge (v, w) and a given

embedding $\mathcal{H}$ of the graph H this can be done in linear time by computing a shortest path in an auxiliary graph H_d, which is obtained from the planar dual of H by the addition of the nodes v and w and edges (v, v_f) and (w, w_f) for all nodes v_f and w_f that correspond to faces of $\mathcal{H}$ whose borders contain v and w, respectively. Thereby, any chosen initial embedding $\mathcal{H}$ of H remains unchanged in the final embedding. For the details of this method see, e.g., [13].

When this method is used, the number of the resulting crossings highly depends on the choice of $\mathcal{H}$. Gutwenger et al. [10] overcome this problem with an efficient algorithm that optimally inserts a single edge into a given graph, i.e., optimizes over all possible embeddings. The *block-tree* and *SPQR-tree* [4, 9] data structures are used in order to describe all possible planar embeddings of a biconnected graph. We will briefly outline the main ideas of this method:

An SPQR-tree represents a decomposition of a given biconnected graph $G = (V, E)$ in its 3-connected components. Each node in the node set V_T of an SPQR-tree corresponds to a biconnected graph, called *skeleton*. Each skeleton represents a simplified version of the original graph G. Each edge e' of a skeleton is either an original edge e of G or represents a biconnected subgraph of G, which is called *expansion graph* of e'. By choosing a unique embedding for each skeleton of an SPQR-tree, a unique embedding of the corresponding graph can be defined and vice versa.

There are three types of nodes $v \in V_T$:

S-node: the skeleton is a simple cycle containing at least three nodes.
P-node: the skeleton contains two nodes and at least 3 parallel edges.
R-node: the skeleton represents a 3-connected graph with at least four nodes.

It is shown in [10] that the edge insertion procedure can be reduced to finding insertion paths for the expansion graphs of some relevant R-nodes in the SPQR tree T_B for each block B of G. As the expansion graph of an R-node is 3-connected and therefore has a unique embedding, the edge-inserting algorithm presented above can then be applied.

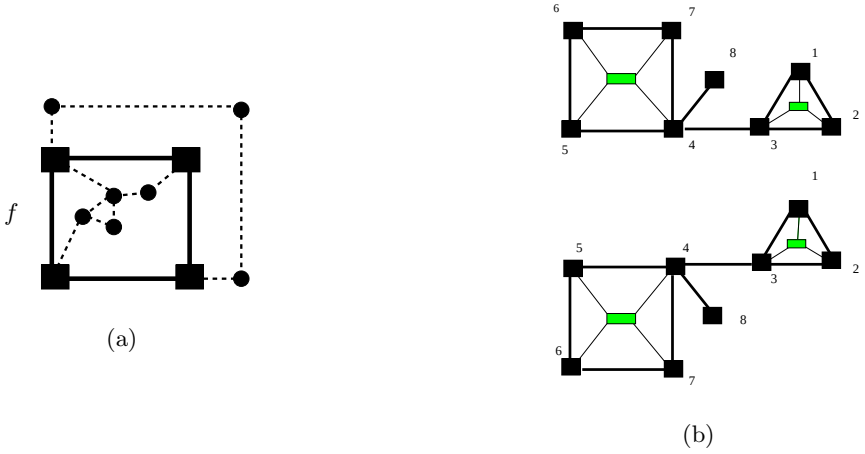
In the following, we call the above methods *fixed insertion* and *optimal insertion*, respectively.

3 The Modified Planarization Method

The direct application of the planarization method to our problem will, in general, violate the constraints (a) and (b). We need to perform certain modifications in order to make it applicable.

Analogously to the standard planarization method, a maximal planar subgraph H of G should be computed first. As it is forbidden to cross the edges $e \in E'$, H should already contain G' as a subgraph, i.e., $E' \subseteq E(H)$. This is always possible since we assume G' to be planar. Moreover, when constructing H we must guarantee the existence of at least one embedding of H that satisfies the following properties which result from the constraints (a) and (b):

(a′) The nodes of G' are placed according to the given embedding $\mathcal{P}'$.



(a) (b)

Fig. 1. Constraints for preprocessing.

(b′) All edges $E(H) \setminus E'$ lie in the outer face f of $\mathcal{P}'$.

Since we assume the remaining subgraph $G \setminus G'$ to be adjacent only to nodes $v \in V''$, we can temporarily remove from G all the nodes and edges of G' except the subgraph $G'' \subseteq G'$ that is induced by V''. The corresponding embedding $\mathcal{P}''$ can be easily extracted from $\mathcal{P}'$.

In order to satisfy the above conditions we preprocess the given instance by extending the given subgraph G'' with additional nodes and edges to fix the predefined embedding. This will be done in two steps: insertion of *wheel graphs* and application of the *algorithm FIXEMBED*. After these preprocessing steps we can compute a maximal planar subgraph H, which contains the extention of the subgraph G''.

3.1 Preprocessing

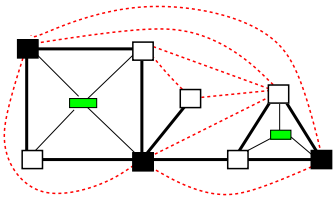
The planarity of H does not automatically guarantee a feasible planar embedding: e.g., Figure 1(a) shows a planar graph H where no embedding that satisfies (a′) and (b′) exists. The graph $H \setminus G''$ (circles and dashed lines) cannot be drawn in the outer face f of G'' (squares and bold lines) without producing crossings. To avoid such situations *wheel-graphs* are used: for each inner face g of G'' we insert a node v_g and connect it with all nodes v on the border of g by the corresponding edges (v, v_g). See, e.g., Figure 2 for such constructions. The insertion of the wheel-graphs results in the graph $\tilde{G}$. An embedding $\mathcal{P}''$ can then easily be extended to the corresponding $\tilde{\mathcal{P}}$.

However, this augmentation does not yet fix the predefined embedding, cf. Figure 1(b). The idea now is to augment $\tilde{G}$ with new edges such that there are only two possible embeddings of the resulting graph, which are mirror-images of each other. To realize this, we keep in mind that all 3-connected graphs satisfy this property.

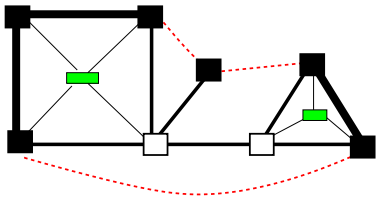
The graph $\tilde{G}$ has a special structure as each non-trivial 2-connected component is also 3-connected due to the presence of the wheel-graphs. Hence, if $\tilde{G}$ is 2-connected, no further extension will be necessary.

Let us assume that several 2-connected components exist. A straight-forward idea is to triangulate the outer face of $\tilde{P}$. However, this approach can produce many additional edges. The main disadvantage of this method is that it generates many needlessly *isolated* nodes. A node $v \in V''$ is isolated if adding any edge $(v, w) \in E \setminus E'$ to the triangulation of $\tilde{G}$ causes at least one edge-crossing (cf. white nodes in Figure 2(a)).

Hence, our goal is to extend $\tilde{G}$ with few edges and few isolated nodes. We suggest an algorithm called FIXEMBED that augments $\tilde{G}$ with $O(|V''|)$ edges in $O(|V''|)$ time so that only nodes $v \in V_S$ are isolated, where $V_S \subset V''$ is the set of cut vertices. We show that even though the resulting graph is not 3-connected, it has only two possible embeddings, which are mirror-images of each other.



(a) Wheel-graphs (rectangular nodes and thin edges), triangulation of the outer face (dashed edges) and isolated (white) nodes.



(b) Dashed edges are the FIXEMBED-edges, bold black and dashed lines build cycle K.

Fig. 2. Preprocessing variants.

Algorithm FIXEMBED:
Let $L(f) = (v_1, \ldots, v_k)$ be the list of nodes of the outer face f of $\mathcal{P}'$ in clock-wise order which are no cut nodes in $\tilde{G}$. This list can be determined in $O(|V''|)$ time. We consider nodes $x \in V''$ in the order of their appearance in $L(f)$, starting with $x = v_1$. Let v_l be the successor of x in $L(f)$. A new edge (x, v_l) is inserted in $\tilde{G}$ only if there is no such edge in $\tilde{G}$. We iterate by setting $x := v_l$. In the case $x = v_k$, an edge (v_k, v_1) is inserted if no such edge already exists and the algorithm terminates with the output graph $\hat{G}$.

It is clear that it is possible to construct an embedding of the resulting $\hat{G}$ in which the embedding $\mathcal{P}''$ is preserved. In such an embedding the outer face is bordered by a simple cycle K that consists of all newly inserted edges and also the edges (v, w) with $v, w \notin V_S$ and v, w belonging to the same biconnected component of $\tilde{G}$. Moreover, K contains only nodes $v \in V'' \setminus V_S$ (cf. Figure 2(b)).

Theorem 1. *The output graph $\hat{G}$ of the FIXEMBED algorithm is 2- but, in general, not 3-connected. However, the graph $\hat{G}$ has only two possible embeddings which are mirror-images of each other.*

Proof. The graph $\hat{G}$ is constructed by inserting additional edges to $\tilde{G}$. It is therefore clear that $\hat{G}$ is connected. Assume there is a node $v \in \hat{V} = \tilde{V}$ whose removal disconnects $\hat{G}$. Then v also disconnects $\tilde{G}$ and therefore $v \in V_S$. Consider the resulting connected components of $\hat{G} \setminus \{v\}$. Each such non-trivial component contains at least one node in $V'' \setminus \{V_S\}$. This is a contradiction, since all such nodes are connected by the simple cycle K. Hence, $\hat{G}$ is 2-connected.

In general, $\hat{G}$ may contain *lines*, which only have nodes $v \in V_S$ as inner nodes. A line is defined as a path in which only the start- and the end node have degrees different from two. The removal of the start- and the end node of such a path disconnects $\hat{G}$. Hence, $\hat{G}$ is, in general, not 3-connected. However, by replacing each such path in $\hat{G}$ by a single edge, we get a 3-connected graph Q. For each embedding $\mathcal{Q}$ of Q there is exactly one corresponding embedding of $\hat{G}$ and vice versa. Therefore, $\hat{G}$ has only two possible embeddings. □

3.2 Constrained Maximal Planar Subgraph

We have to modify both the greedy heuristic (cf. Section 2) and the ILP-formulation [12] in order to determine the required maximal or maximum planar subgraph H, respectively.

The only modification of the greedy heuristic is the use of the planar graph $\hat{G}$ as the initial graph.

The integer linear programming formulation of the Maximum Planar Subgraph problem can be modified by setting $x_e = 1$ for all edges $e \in \hat{E}$ and the branch-and-cut algorithm of [12] can be applied.

3.3 Constrained Fixed Insertion

The following algorithm is based on the fixed insertion algorithm presented in Section 2.2 that iteratively inserts edges of the set $I \subset E \setminus E(H)$ into the initially chosen embedding $\mathcal{H}$.

Initial embedding of H. As $\mathcal{H}$ remains unchanged, we must ensure the feasibility of $\mathcal{H}$ before starting the insertion procedure. Although we have introduced wheel-graphs in order to embed subgraphs of $H \setminus G''$ in the outer face of G'', not all embeddings of H satisfy this property (cf. Figure 3(a)). One way of obtaining a proper embedding is to use the c-planar embedding algorithm for c-planar and c-connected graphs described in [8]. The graph H can be viewed as a clustered graph with a single non-trivial cluster that contains all nodes of $\hat{G}$.

By using dummy nodes as shown in Figure 3(b) which do not belong to the cluster, we can avoid artificial edges to be interpreted as cluster edges. Due to such dummy nodes, the wheel-graphs and the planarity of G', the graph H is c-planar. It is also c-connected, as we assume G' to be connected. Hence, by using the algorithm of [8], we obtain a feasible embedding $\mathcal{H}$.

As soon as we have found this embedding, all FIXEMBED-edges, wheel-graphs and dummy nodes can be removed from H, since the embedding P'' of G'' will not be changed afterwards. Doing so reduces the complexity and tends to enhance the quality of the subsequent edge insertion steps.

(a) Flipping subgraphs.

(b) Dummy nodes (white) on FIXEMBED edges.

Fig. 3. Repairing the embedding.

Constrained Insertion. Let $\bar{H}$ and $\bar{\mathcal{H}}$ be the resulting graph and its embedding, respectively. An edge $(v, w) \in I$ can now be inserted by computing a shortest path in the auxiliary graph. As it is forbidden to have edge-crossings on edges of the given subgraph G' (cf. bold edges in Figure 4), we remove their corresponding dual edges. The existence of a shortest $(v \to w)$-

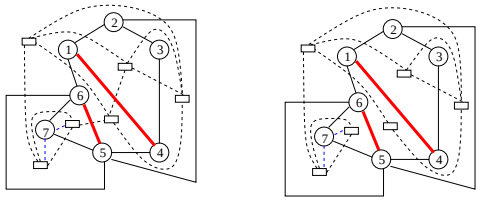


Fig. 4. The bold edges are not allowed to have crossings.

path is guaranteed by the structure of G'' and by the fact that no edge of $E \setminus E'$ is incident to any node of $V' \setminus V''$. The resulting embedding is feasible for the original problem: as there is no crossing on the edges of E'', no edge $e \in I$ can be placed in an inner face of $\mathcal{P}'$.

3.4 Constrained Optimal Insertion

We can also use the algorithm of Gutwenger et al. [10] to insert edges preserving the fixed embedding of G'' and not to allow the edges of E' to be crossed. However, as the algorithm optimizes over all possible embeddings of H, we are not allowed to remove the wheel-graphs and FIXEMBED-edges before applying it.

Constrained weighted optimal insertion. As described in Section 2.2 the algorithm determines optimal insertion paths for each biconnected component of the given graph. Our goal is to forbid edge-crossings on the edges of $\tilde{G}$. All these edges belong to the same block B of H, as $\tilde{G} \subseteq \hat{G}$ and $\hat{G}$ is 2-connected (cf. Theorem 1). Hence, we only need to adapt the algorithm for this single block B. For a given edge $(v, w) \in I$ the algorithm of Gutwenger et al. constructs a corresponding SPQR-Tree T_B, determines the relevant path $P_{(v,w)}$ in T_B and determines optimal insertion paths for each R-node on $P_{(v,w)}$ by using the dual graph of the corresponding expansion graph. In order to forbid edge-crossings on the edges $e \in \tilde{E}$, it is again sufficient to omit the corresponding dual edges.

The presence of the FIXEMBED-edges in H may cause unnecessary edge-crossings. It may occur that the insertion algorithm has two different possibilities to insert an edge: it either produces one edge-crossing with an original edge of G or several edge-crossings only with dummy edges. The insertion algorithm as described above would prefer the first case. In order to prevent this, a weighted version of this algorithm can be used: for each edge $e \in E(H)$ we assign weights $w(e) = 1$ for $e \in E(H) \cap E$ and $w(e) = 0$ for $E(H) \setminus E$.

Computing a feasible embedding. As soon as all edges I have been inserted to H, a corresponding feasible embedding must be determined. To achieve this, we apply the embedding algorithm for c-planar graphs [8] as in the previous section. Finally, the resulting embedding is obtained by deleting all artificial nodes and edges.

3.5 Embedding Completion

After the edge insertion step, we have an embedding $\mathcal{P}_{G''}$ of the graph $G \setminus (G' \setminus G'')$. In order to get the required embedding $\mathcal{P}$ of G, we have to insert the inner nodes and edges of G'' that were removed during the preprocessing step (cf. 3.1).

As the embedding $\mathcal{P}'$ of the given instance contains all adjacency lists of the inner nodes, we get the required embedding $\mathcal{P}$ of G by merging the lists of $\mathcal{P}'$ and $\mathcal{P}_{G''}$. Beforehand, we have to check whether the embedding of G'' in $\mathcal{P}_{G''}$ is exactly $\mathcal{P}''$ or its mirrored version. In the latter case the adjacency lists of the inner nodes in $\mathcal{P}'$ must be reversed.

4 Relaxation of the Embedding Constraints

The constraints defined in Section 3 impose many restrictions on the final drawing of the given graph. Compared to unrestricted embeddings of the same graph, the number of edge crossings in the constrained embedding may increase significantly. This, in turn, may impede the readability of a drawing and thus complicate the analysis. In such a case we suggest the following compromise: we relax the embedding-preservation constraint by requiring only each 2-connected component of the given subgraph to be embedded according to the embedding $\mathcal{P}'$. For each such component we also admit corresponding mirrored embeddings. E.g., both embeddings in Figure 1(b) become feasible. Such a relaxation of the constraints still allows the user to recognize G' and at the same time admit drawings with a lower number of edge-crossings. We can modify the algorithms presented in the previous section by omitting the application of the algorithm FIXEMBED. In the completion step, the adjacency lists of the inner nodes $V' \setminus V''$ in P' have to be adjusted according to the resulting embedding.

5 Multiple Subgraphs with Fixed Embedding

The algorithms presented above can easily be extended to the case in which two or more subgraphs with fixed embedding are given. We can have the following cases:

(1) The subgraphs $G_1, \ldots, G_k$ are node- and edge-disjoint.

(2) $G_1, \ldots, G_k$ are pairwise edge-disjoint and have no common inner nodes.
(3) The subgraphs have common edges or common inner nodes.

In case (1) the preprocessing step has to be performed for each subgraph and all resulting subgraphs $\hat{G}_1, \ldots, \hat{G}_k$ should be contained in the maximal planar subgraph H. This graph can then also be viewed as a cluster graph, containing k node-disjoint clusters at the same level. The insertion algorithms can also be modified in order to forbid the crossings on all edges $E''_1, \ldots E''_k$.

In case (2), we can treat connected $l \leq k$ subgraphs as a single subgraph, by taking the union of their nodes and edges. The embeddings $P_1, \ldots, P_l$ can be merged to a single embedding straightforwardly. The algorithms of Sections 3 and 4 can then be applied to this transformed instance.

The last case may have incompatible embeddings so that no feasible solution exists. If the given embeddings are compatible, this case can be reduced to the previous one.

6 Constructing a Quasi-Orthogonal Drawing

An attractive way to realize our constrained embeddings in the plane is the application of (quasi-)orthogonal drawing algorithms.

Using the algorithm for planar cluster drawing ([13], pages 161–163) we draw the given subgraph within a rectangular cluster region by using *cages* for the clusters as shown in Figure 5. The algorithm then constructs a drawing in which the edges of a cage always form a rectangle.

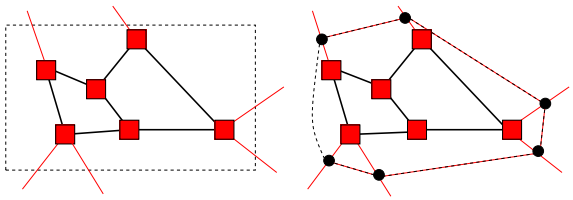


Fig. 5. Construction of a cage for a given subgraph.

However, in spite of the feasible relative positions of the nodes in V', it may occur that the resulting drawing significantly differs from the form expected by the user. Therefore, we finally outline an algorithm that preserves a given drawing of a subgraph G' and realizes the graph $G \setminus G'$ in a (quasi-)orthogonal manner. For the latter we compute a feasible embedding of $G \setminus (G' \setminus G'')$, i.e., we perform one of our algorithms on G without applying the completion step. Analogously to the planar cluster drawing algorithm, we insert cage edges into this embedding and remove G'' from it. A quasi-orthogonal drawing of the resulting embedding is then computed with the requirement that each cage must be rectangular with a certain minimum width and height so that the given drawing of G' can be placed inside.

References

1. A.-L. Barabasi and Z. N. Oltvai. Network biology: Understanding the cell's functional organization. *Nature Reviews*, 2004.
2. C Batini, E Nardelli, and R Tamassia. A layout algorithm for data flow diagrams. *IEEE Trans. Softw. Eng.*, 12(4):538–546, 1986.
3. G. Di Battista, P. Eades, R. Tamassia, and I. G. Tollis. *Graph Drawing*. Prentice Hall, 1999.
4. G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996.
5. K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335–379, 1976.
6. J. M. Boyer and W. J. Myrvold. On the cutting edge: Simplified o(n) planarity by edge addition. *J. Graph Algorithms Appl.*, 8(2):241–273, 2004.
7. H. N. Djidjev. A linear algorithm for finding a maximal planar subgraph.
8. Q. Feng, R. F. Cohen, and P. Eades. Planarity for clustered graphs. In *ESA '95*, volume 979 of *Lecture Notes in Computer Science*. Springer Verlag, 1995.
9. C. Gutwenger and P. Mutzel. A linear time implementation of spqr trees. In J. Marks, editor, *Graph Drawing*, volume 1984 of *Lecture Notes in Computer Science*, pages 77–90. Springer Verlag, 2001.
10. C. Gutwenger, P. Mutzel, and R. Weiskircher. Inserting an edge into a planar graph. In *SODA '01: Proceedings of the twelfth annual ACM-SIAM symposium on Discrete algorithms*, pages 246–255, Philadelphia, PA, USA, 2001. Society for Industrial and Applied Mathematics.
11. M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In G. Di Battista, editor, *Proceedings of fifth International Symposium on Graph Drawing '97*, volume 1353 of *Lecture Notes in Computer Science*, pages 193–204. Springer Verlag, 1997.
12. M. Jünger and P. Mutzel. Maximum planar subgraphs and nice embeddings: Practical layout tools. *Algorithmica*, 16:33–59, 1996.
13. M. Jünger and P. Mutzel. *Graph Drawing Software*. Springer Verlag, 2003.
14. M. Kaufmann and D. Wagner, editors. *Drawing Graphs, Methods and Models*, volume 2025 of *Lecture Notes in Computer Science*. Springer Verlag, 2001.
15. G. W. Klau. Quasi-orthogonales Zeichnen planarer Graphen mit wenigen Knicken. Master's thesis, Technische Fakultät der Universität des Saarlandes, 1997.
16. M. Koyutürk, A. Grama, and W. Szpankowski. An efficient algorithm for detecting frequent subgraphs in biological networks. In *Proceedings of the twelfth International Conference Intelligent Systems for Molecular Biology (ISMB'04)*, pages i200–i207, 2004.
17. R. Milo, S. Schen-Orr, S. Itzkovitz, N. Kashtan, D. Chklovskii, and U. Alon. Network motifs: Simple building blocks of complex networks. *Science*, 298:824–827, 2002.
18. J. A. La Poutré. Alpha-algorithms for incremental planarity testing (preliminary version). In *STOC '94: Proceedings of the twenty-sixth annual ACM symposium on Theory of computing*, pages 706–715, New York, NY, USA, 1994. ACM Press.
19. F. Schreiber and H. Schöbbermeyer. Towards motif detection in networks: Frequency concepts and flexible search. In E. Merelli, P. Gonzalez, and A. Omicini, editors, *Proceedings of the International Workshop on Network Tools and Applications in Biology (NETTAB'04)*, pages 91–102, 2004.