I/O-Optimal Planar Embedding Using Graph Separators*

Norbert Zeh

School of Computer Science
Carleton University
Ottawa, ON K1S 5B6
Canada
nzeh@scs.carleton.ca

Abstract

We present a new algorithm to test whether a given graph G is planar and to compute a planar embedding $\hat{G}$ of G if such an embedding exists. Our algorithm utilizes a fundamentally new approach based on graph separators to obtain such an embedding. The I/O-complexity of our algorithm is $O(\text{sort}(N))$. A simple simulation technique reduces the I/O-complexity of our algorithm to $O(\text{perm}(N))$. We prove a matching lower bound of $\Omega(\text{perm}(N))$ I/Os for computing a planar embedding of a given planar graph.

1 Introduction

I/O-efficient graph algorithms have received considerable attention because massive graphs arise naturally in many applications. Recent web crawls, for example, produce graphs of on the order of 200 million nodes and 2 billion edges. Recent work in web modeling uses depth-first search, breadth-first search, shortest paths, and connected components as primitive operations for investigating the structure of the web [7]. Massive graphs are also often manipulated in Geographic Information Systems (GIS), where many fundamental problems can be formulated as basic graph problems. The graphs arising in GIS applications are often planar. Yet another example of massive graphs is AT&T's 20TB phone call graph [8]. When working with such large data sets, the transfer of data between internal and external memory, and not the internal memory computation, is often the bottleneck. Thus, I/O-efficient algorithms can lead to considerable run-time improvements.

Many graph algorithms designed in the RAM model of computation use breadth-first search (BFS) or depth-first search (DFS) to explore the given graph, as these two strategies can easily be realized in linear time; yet they provide valuable information about the structure of the graph. Unfortunately, no I/O-efficient algorithms for BFS and DFS in arbitrary sparse graphs are known, while existing algorithms perform reasonably well on dense graphs. This forces algorithm designers to develop algorithms that avoid using BFS or DFS as primitive operations, if they are to be I/O-efficient.

Recently, a number of papers [4, 5, 17, 33] have focused on developing I/O-efficient algorithms for fundamental graph problems on embedded planar graphs. These algorithms solve BFS, DFS, single source shortest paths (SSSP), and the problem of computing weighted separators of an embedded planar graph in $O(\text{sort}(N))$ I/Os using linear space. From a practical point of view the fact that these algorithms need a planar embedding to be given as part of the input is not a serious constraint, as in most large scale applications it

*Research supported in part by NSERC and NCE GEOIDE.

is known that a given graph is planar only because a planar embedding of the graph is given. From a theoretical point of view, however, it is desirable to design I/O-efficient algorithms for planarity testing and planar embedding, in order to obtain I/O-efficient algorithms for a number of fundamental problems on planar graphs which do not require any additional information about the given graph.

1.1 Model of Computation

The algorithm in this paper is designed and analyzed in the Parallel Disk Model (PDM) [32]. In this model, D identical disks of unlimited size are attached to a machine with an internal memory capable of holding M data items. These disks constitute the external memory of the machine. Initially, all data is stored on disk. Each disk is partitioned into blocks of B data items each. An I/O-operation is the transfer of up to D blocks, at most one per disk, to or from internal memory from or to external memory. The complexity of an algorithm in the PDM is the number of I/O-operations it performs.

Sorting, permuting, and scanning an array of N consecutive data items are primitive operations often used in external memory algorithms. Their I/O-complexities are $\text{sort}(N) = \Theta((N/(DB))\log_{M/B}(N/B))$, $\text{perm}(N) = \Theta(\min(N, \text{sort}(N)))$, and $\text{scan}(N) = \Theta(N/(DB))$, respectively [32].

1.2 Previous Results

I/O-efficient graph algorithms have been studied in a number of papers [1, 2, 4, 5, 9, 11, 17, 20, 22, 23, 25, 27, 31]. We only discuss results on undirected BFS, DFS, single source shortest paths, planar embedding, and graph connectivity here. The best SSSP algorithm for arbitrary undirected graphs takes $O(|V| + (|E|/B)\log_2|E|)$ I/Os [20]. The best BFS algorithm for arbitrary undirected graphs takes $O(|V| + \text{sort}(|E|))$ I/Os [27]. Recently a BFS algorithm for graphs of bounded degree has been presented in [25]. If d is the maximum vertex degree in the graph, the algorithm takes $O(|V|/(\gamma\log_d B) + \text{sort}(B^\gamma|V|))$ I/Os using $O(|V|/B^{1-\gamma})$ blocks of external memory, for $0 < \gamma \leq \frac{1}{2}$.

In [17], an $O(\text{sort}(N))$ I/O algorithm for computing a 2/3-separator of size $O(\sqrt{N})$ for an embedded planar graph G is given, provided that a BFS-tree of G is part of the input. In [4], this idea has been extended to obtain an $O(\text{sort}(N))$ I/O algorithm to compute an ε-separator of size $O(\text{sort}(N) + \sqrt{N/\varepsilon})$ for an embedded planar graph, provided that a BFS-tree of the graph is given. Using the computed separator, the SSSP problem can then be solved in $O(\text{sort}(N))$ I/Os for the given graph [4]. In a recent paper [5], two DFS algorithms for embedded planar graphs are given. The first one takes $O(\text{sort}(N)\log N)$ I/Os. The second one takes $O(I(N))$ I/Os, where $I(N)$ is the number of I/Os required to compute a BFS-tree of an embedded planar graph. Another recent result [33] shows how to compute an ε-separator of size $O(\sqrt{N/\varepsilon})$ for an unweighted planar graph in $O(\text{sort}(N))$ I/Os, provided that $\varepsilon N\log^2(DB) \leq M$. This algorithm does not require a BFS-tree or embedding of the given graph as part of the input. Together with the results of [4] and [5] this implies that BFS, DFS, and SSSP can be solved in $O(\text{sort}(N))$ I/Os on embedded planar graphs, provided that $M \geq (DB)^2\log^2(DB)$. Also, once a BFS-tree and an embedding of G is given, the separator algorithm of [3] can be realized using external memory techniques to compute in $O(\text{sort}(N))$ I/Os an ε-separator of size $O(\sqrt{N/\varepsilon})$ for a weighted planar graph G. We are not aware of any results on computing planar embeddings I/O-efficiently.

In [22], it is shown how to test a given graph G for outerplanarity and compute an outerplanar embedding of G. Given the embedding, it is shown how to solve BFS, DFS, and how to compute a 2/3-separator of size 2 for G. All algorithms in [22] take $O(\text{sort}(N))$ I/Os. In [23] it is shown how to solve the SSSP problem in $O(\text{sort}(N))$ I/Os on graphs of bounded treewidth. It is shown in [22] that BFS, DFS, and SSSP require at least $\Omega(\text{perm}(N))$ I/Os, even on outerplanar graphs. It is also shown that outerplanar embedding takes at least $\Omega(\text{perm}(N))$ I/Os.

In [9], I/O-efficient algorithms for computing the connected and biconnected components of an undirected graph are presented. These algorithms take $O(\text{sort}(N))$ I/Os on graphs which are sparse under edge contraction. This includes planar graphs. These algorithms are the result of applying a general simulation technique for PRAM algorithms in external memory to the connectivity algorithm of [10] and the biconnectivity algorithm of [30]. There are no direct results on computing triconnected components I/O-efficiently, although one may apply the PRAM simulation of [9] to the triconnectivity algorithm of [13].

A number of PRAM algorithms for planarity testing and planar embedding have been proposed [18, 19, 26, 28]. In [19], the first such algorithm using a linear number of processors was presented; the algorithm runs in $O(\log_2^2 N)$ time. The algorithm of [28] runs in $O(\log_2 N)$ time using $O(C(N))$ processors, where $C(N)$ is the number of processors required to compute the connected components of a graph in $O(\log_2 N)$ time. Using the PRAM simulation technique of [9], one can obtain I/O-efficient, but suboptimal, embedding algorithms from the algorithms of [19, 28]. A more direct implementation of the algorithm of [28] using external memory techniques produces a planar embedding algorithm that takes $O(\text{sort}(N))$ I/Os. However, it is not clear whether the algorithm of [28] can be used to test whether a given graph is planar.

In internal memory, planarity testing and the problem of computing a planar embedding of a given graph G are well-studied. The first paper to present a linear time algorithm for planarity testing and planar embedding is [16]. A previous algorithm of [21] was later made to run in linear time using results of [6, 12]. Important implementation details of the algorithm of [16] are provided in [24]. Any graph traversal can be used to identify the connected components of a graph in linear time. In [29], a linear time algorithm for finding the biconnected components of a graph is presented. In [15], the idea of [29] was extended to identify the triconnected components of a biconnected graph.

1.3 Our Result

In this paper, we present a new algorithm to test whether a given graph is planar and to compute a planar embedding of the graph if the answer is affirmative. Our algorithm takes $O(\text{sort}(N))$ I/Os using linear space, provided that $M \geq (DB)^2 \log^2(DB)$. Intuitively, our approach can be described as follows: First use the algorithm of [33] to compute a subset S of $O(N/(DB))$ vertices of G whose removal partitions G into $O(N/(DB)^2)$ subgraphs $G_1, \ldots, G_k$ of size at most $(DB)^2$ each. Each graph G_i is adjacent to at most DB separator vertices, which we denote by ∂G_i. Let $\tilde{G}_i$ be the subgraph of G induced by the vertices in $V(G_i) \cup \partial G_i$, for $1 \leq i \leq k$. Let $G'_1, \ldots, G'_l$ be the connected components of graphs $\tilde{G}_1, \ldots, \tilde{G}_k$. Denote the set of separator vertices in G'_j, $1 \leq j \leq l$, by S_j.

For each graph G'_j, compute a constraint graph C_j of size $O(|S_j|)$ which captures the constraints imposed on the embedding of G by G'_j. Informally, these constraints are of the form: Can two separator vertices be on the same face of an embedding $\hat{G}'_j$ of G'_j? In which order do they have to appear along the boundary of such a face? Etc. These constraints are derived from a decomposition of G'_j into its biconnected and triconnected components. Joining graphs $C_1, \ldots, C_l$ at their separator vertices, we obtain an approximate graph A of size $O(N/(DB))$. We show that G is planar if and only if $\tilde{G}_1, \ldots, \tilde{G}_k$ are planar and A is planar. Also, a planar embedding of G can be obtained from a planar embedding of A by locally replacing the embeddings $\hat{C}_1, \ldots, \hat{C}_l$ of graphs $C_1, \ldots, C_l$ in $\hat{A}$ with consistent embeddings $\hat{G}'_1, \ldots, \hat{G}'_l$ of graphs $G'_1, \ldots, G'_l$.

Our algorithm spends $O(\text{sort}(N))$ I/Os to compute the separator S, using the result of [33]. Computing graphs $C_1, \ldots, C_l$ takes $O(\text{scan}(N))$ I/Os, as each graph G'_j has size at most $(DB)^2 + (DB) \leq M$ and, hence, the construction of C_j from G'_j can be carried out in internal memory. Computing a planar embedding of A takes $O(N/(DB))$ I/Os using the algorithm of [16], as $|A| = O(N/(DB))$. Finally, the embedding of G is constructed in $O(\text{sort}(N))$ I/Os, as the replacement of embeddings $\hat{C}_1, \ldots, \hat{C}_l$ by embeddings $\hat{G}'_1, \ldots, \hat{G}'_l$ can be done locally and thus in internal memory; the necessary coordination between these local replacement steps is achieved using time-forward processing [9].

In order to prove that our algorithm is optimal up to a constant factor, we show that it takes $\Omega(\text{perm}(N))$ I/Os to compute a planar embedding of a given planar graph. A simple simulation technique together with any linear time embedding algorithm reduces the I/O-complexity of our embedding algorithm to $O(\text{perm}(N))$ I/Os, thereby matching the lower bound.

1.4 Preliminaries

An *undirected multigraph* $G = (V, E)$ is an ordered pair of a set V and a multiset E. The elements of V are the *vertices* of G; the elements of E are the *edges* of G and are unordered pairs $\{v, w\}$, $v, w \in V$. We sometimes represent the elements in E as triples (v, w, i) to distinguish the different copies of edge $\{v, w\}$ in E. Triples (v, w, i) and (w, v, i) are considered to represent the same edge. For an edge $\{v, w\} \in E$, vertices v and w are the *endpoints* of edge $\{v, w\}$. Vertices v and w are said to be *adjacent*. Edge $\{v, w\}$ is *incident* to vertices v and w. Graph G is *simple* if every edge appears at most once in E.

A multigraph $H = (W, F)$ is a *subgraph* of a multigraph G if $W \subseteq V$ and $F \subseteq E$. A *path* in G is a subgraph $P = (W, F)$ of G such that $W = \{v_0, \ldots, v_p\}$ and $F = \{\{v_{i-1}, v_i\} : 1 \leq i \leq p\}$. In this case, we write $P = (v_0, \ldots, v_p)$. We call v_0 and v_p the *endpoints* of P. A path $P = (v_0, \ldots, v_p)$ is *simple* if vertices $v_0, \ldots, v_{p-1}$ are pairwise distinct and vertices $v_1, \ldots, v_p$ are pairwise distinct. We call P a *cycle* if $v_0 = v_p$. In this case, we write $P = (v_0, \ldots, v_{p-1})$.

For a set $W \subseteq V$ of vertices, let $G[W]$ be the subgraph of G *induced* by W. Graph $G[W]$ is defined as $G[W] = (W, \{\{v, w\} \in E : v, w \in W\})$. Similarly, for a set $F \subseteq E$ of edges, graph $G[F]$ is defined as $G[F] = (\bigcup_{\{v,w\} \in F} \{v, w\}, F)$. For a set of vertices $W \subseteq V$, let $G - W = G[V \setminus W]$; for a vertex $v \in V$, graph $G - v$ is the same as graph $G - \{v\}$. For a set of edges $F \subseteq E$, let $G - F = G[E \setminus F]$. For a subgraph $H = (W, F)$ of G, let $G - H = G[E(G) \setminus E(H)]$. Let $F \subseteq E$ and $H = G[F]$. Then $H = G[E \setminus F]$. For a graph $G = G_1 \cup G_2$ such that $V(G_1) \cap V(G_2) = W$ and a graph G_1' with $W \subseteq V(G_1')$, let $G[G_1/G_1']$ be the graph $G_1' \cup G_2$. Intuitively, $G[G_1/G_1']$ is the graph obtained from G by replacing subgraph G_1 with graph G_1'.

A multigraph G is *connected* if there is a path with endpoints v and w, for all $v, w \in G$. The *connected components* of G are the maximal connected subgraphs of G. A *cutpoint* of a connected multigraph G is a vertex v such that $G - v$ is disconnected. Multigraph G is *biconnected* if it does not have any cutpoints. The *biconnected components* or *bicomps* of a connected multigraph G are the maximal biconnected subgraphs of G. The bicomps of an arbitrary multigraph are the bicomps of its connected components.

Given a multigraph $G = (V, E)$ and a subgraph $H = (W, F)$ of G, the bridges of H are defined as follows: Consider the connected components of $G - V(H)$. Let K be such a component. Then K defines a *non-trivial bridge* of H which is the subgraph of G induced by all edges incident to vertices in K. A *trivial bridge* is an edge in $G - H$ with both endpoints in H. The trivial and non-trivial bridges are the *bridges* of H in G.

A pair $\{v, w\}$ of vertices of a biconnected graph G is a *separation pair* if graph $H = (\{v, w\}, \emptyset)$ has at least two non-trivial bridges in G or at least three bridges, one of which is non-trivial. In the former case, we call $\{v, w\}$ a *non-trivial separation pair*. If G is a simple graph, then all separation pairs are non-trivial. Graph G is *triconnected* if it does not have a separation pair.

Given a separation pair $\{v, w\}$ with bridges $B_1, \ldots, B_q$, the *split* $s(v, w, i)$ chooses two graphs B' and B'' such that $B' = B_1 \cup \cdots \cup B_{q'}$, $B'' = B_{q'+1} \cup \cdots \cup B_q$, $E(B') \geq 2$ and $E(B'') \geq 2$, and partitions G into two subgraphs $G_1 = (V(B'), E(B') \cup \{(v, w, i)\})$ and $G_2 = (V(B''), E(B'') \cup \{(v, w, i)\})$. Edge (v, w, i) is called the *virtual edge* corresponding to split $s(v, w, i)$. The *split components* of G are defined as the graphs obtained by recursively splitting G_1 and G_2 until there are no more separation pairs. The split components of G are not necessarily unique. There are three types of split components: (1) triconnected simple graphs, (2) triple bonds (two vertices with three edges between them), and (3) triangles.

The *merge* $m(v, w, i)$ of two graphs G_1 and G_2 sharing a virtual edge (v, w, i) constructs a graph $G = (V(G_1) \cup V(G_2), (E(G_1) \setminus \{(v, w, i)\}) \cup (E(G_2) \setminus \{(v, w, i)\}))$ from G_1 and G_2. A graph G can be recon-

structed from its split components by recursive application of merge operations. To construct the triconnected components of a biconnected graph G, merge bonds sharing virtual edges until no two bonds share a virtual edge, and merge simple cycles sharing virtual edges until no two simple cycles share a virtual edge. The resulting graphs are the *Tutte components*, *triconnected components*, or *tricomps* of G. If G is not biconnected, the tricomps of G are the triconnected components of its bicomps. The triconnected components of G are unique and of three types: (1) triconnected simple graphs, (2) bonds, and (3) simple cycles. The separation pairs corresponding to the remaining virtual edges are the *Tutte pairs* of G. Let $H = (W, F)$ be a graph obtained by merging a number of tricomps, and let F' be the set of virtual edges in H. Then the *kernel* H° of H is the graph $H^\circ = (W, F \setminus F')$.

A graph G is *planar* if it can be drawn in the plane so that the edges of G do not intersect, except at their endpoints. Such a drawing of G is called a *topological embedding* of G and denoted by $\mathcal{E}(G)$. Every topological embedding of G defines an order of the edges incident to each vertex $v \in G$, clockwise around v. A representation of these orders for all vertices $v \in G$ is called a *combinatorial embedding* of G and denoted by $\hat{G}$. For most graph algorithms, the latter is sufficient, so that the planarity problem is the problem of computing a combinatorial embedding of a given graph. Given a topological embedding $\mathcal{E}(G)$ consistent with a combinatorial embedding $\hat{G}$ of G, we call the connected regions of $\mathbb{R}^2 \setminus \mathcal{E}(G)$ the *faces* of $\hat{G}$. Let F denote the set of faces of $\hat{G}$. By Euler's formula $|V| + |F| - |E| = 2$. In particular, $|E| \leq 3|V| - 6$, for every simple planar graph G. We define the *size* $|G|$ of a planar graph G as the number $|V|$ of vertices in G. The planar embedding of a simple triconnected planar graph G is unique in the following sense [14]: Let $\hat{G}_1$ and $\hat{G}_2$ be two planar embeddings of G. Then $\hat{G}_1$ and $\hat{G}_2$ have the same number of faces, and there exists a bijection $\sigma : [1,k] \to [1,k]$ such that if $f_1, \ldots, f_k$ are the faces of $\hat{G}_1$ and $f'_1, \ldots, f'_k$ are the faces of $\hat{G}_2$, then faces f_i and $f'_{\sigma(i)}$ have the same vertices on their boundaries; moreover, either the order of the vertices on the boundary of face f_i is the same as that on the boundary of $f'_{\sigma(i)}$, for all $1 \leq i \leq k$, or the order is reversed, for all $1 \leq i \leq k$.

Given a set $S \subseteq V$ of vertices of G and a subgraph $H \subseteq G - S$, ∂H is the set of vertices in S adjacent to vertices in H. We call ∂H the *boundary* of H.

A graph $G = (V, E)$ is *bipartite* if the vertex set V can be partitioned into two sets V_1 and V_2 such that $v \in V_1$ and $w \in V_2$, for every edge $\{v, w\} \in E$. In this case we write $G = (V_1, V_2, E)$. We need the following technical results.

Lemma 1.1 [33] *Let $G = (V_1, V_2, E)$ be a simple connected bipartite planar graph such that the vertices in V_2 have degree at least three each. Then $|V_2| \leq 2|V_1|$.*

Lemma 1.2 *Given an embedding $\hat{G}$ of a simple biconnected planar graph G, graph G is triconnected if and only if there are no two faces f_1 and f_2 of $\hat{G}$ and two vertices v and w such that v and w are on the boundary of both f_1 and f_2, but edge $\{v, w\}$ does not exist or is on the boundary of at most one of f_1 and f_2.*

Proof. First assume that there are two faces f_1 and f_2 of $\hat{G}$ and two vertices v and w such that v and w are on the boundaries of both f_1 and f_2, but edge $\{v, w\}$ does not exist or is on the boundary of at most one of f_1 and f_2. Assume w.l.o.g. that edge $\{v, w\}$ is on the boundary of face f_2 if this edge exists (Figure 1.1a). Let P_1 be the path from v to w counterclockwise around f_1, and P_2 be the path from v to w clockwise around f_1. As edge $\{v, w\}$ is not on the boundary of face f_1, path P_1 contains an internal vertex x, and path P_2 contains an internal vertex y. We can connect vertices v and w by two curves α and β that are completely contained in the interiors of faces f_1 and f_2, respectively. The union of curves α and β is a closed Jordan curve γ which does not intersect any edges of G and contains only vertices v and w. Vertex x is inside γ, vertex y is outside. Thus, any path from x to y must contain either v or w, so that $\{v, w\}$ is a separation pair. Hence, graph G cannot be triconnected.

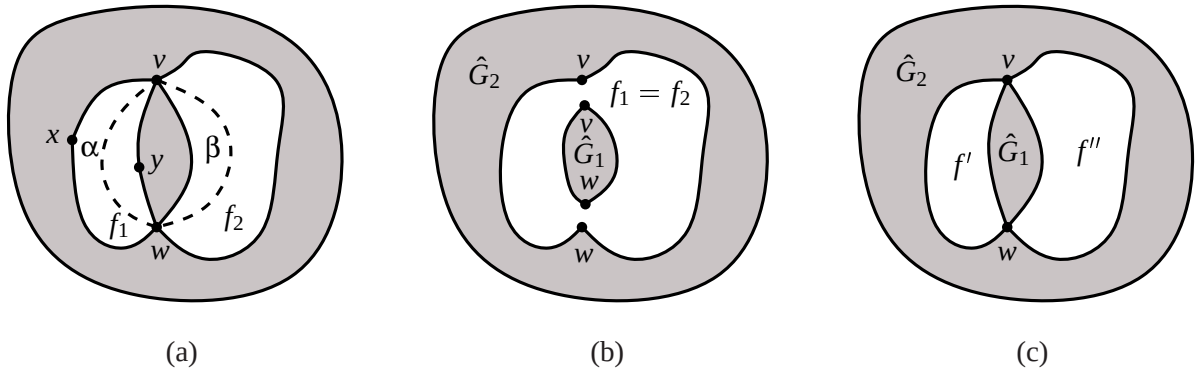


Figure 1.1
Proof of Lemma 1.2.

Now assume that G is not triconnected. Then G must contain a separation pair $\{v,w\}$. The vertex set $V \setminus \{v,w\}$ can be partitioned into two non-empty sets V_1 and V_2 such that any path between two vertices $x \in V_1$ and $y \in V_2$ must contain at least one of v and w. Let $G_1 = G[V_1 \cup \{v,w\}]$ and $G_2 = G[V_2 \cup \{v,w\}]$. If edge $\{v,w\}$ is in G, remove this edge from G_2. Graphs G_1 and G_2 are biconnected and planar. In particular, let $\hat{G}_1$ and $\hat{G}_2$ be the restrictions of embedding $\hat{G}$ of G to G_1 and G_2, respectively. Then $\hat{G}_2$ is contained in a face f_1 of $\hat{G}_1$, and $\hat{G}_1$ is contained in a face f_2 of $\hat{G}_2$. Both f_1 and f_2 contain v and w (Figure 1.1b).

By joining the two copies of v and w in G_1 and G_2, face $f_1 = f_2$ in Figure 1.1b is split into two faces f' and f'' both of which have v and w on their boundaries (Figure 1.1c). However, neither G_1 nor G_2 consist of a single edge, so that f' and f'' cannot share edge $\{v,w\}$. Faces f' and f'' are faces of $\hat{G}$, so that we have shown that there are two faces f' and f'' of $\hat{G}$ and two vertices v and w so that v and w are shared by faces f' and f'', but edge $\{v,w\}$ is not. □

2 Overview of Our Algorithm

In this section, we present the framework of our algorithm. In subsequent sections we fill in the necessary details. Algorithm 2.1 gives an outline of our algorithm.

Theorem 2.1 *Algorithm 2.1 correctly tests whether a simple graph G is planar and computes a planar embedding $\hat{G}$ of G if G is planar. The algorithm takes $O(\mathrm{sort}(N))$ I/Os using $O(N/B)$ blocks of external memory, provided that $M \geq (DB)^2 \log^2(DB)$.*

Proof. We first prove the correctness of Algorithm 2.1. Assume that our algorithm reports that graph G is not planar. This can happen in lines 2, 6, 12, or 18. If $|E| > 3|V| - 6$, graph G is not planar by Euler's formula. The separator algorithm of [33] can fail for two reasons: Either G is not sparse under edge contraction, or the algorithm identifies a non-planar subgraph of G. In both cases, G cannot be planar. If one of the graphs $G'_1, \ldots, G'_l$ is non-planar, G cannot be planar, as these are subgraphs of G. Finally, if A is non-planar, G cannot be planar by Lemma 7.1. On the other hand, if our algorithm does not report that G is non-planar, the output of the algorithm is a planar embedding $\hat{G}$ of G, by Lemma 8.1.

Next we analyze the I/O-complexity of Algorithm 2.1. Lines 1–3 take $O(\mathrm{scan}(N))$ I/Os, as they only require counting the vertices and edges in G. As shown in [33], Lines 4–7 take $O(\mathrm{sort}(N))$ I/Os, provided that $M \geq (DB)^2 \log^2(DB)$. Lines 8–9 take $O(\mathrm{scan}(N))$ I/Os, as each subgraph $\tilde{G}_i$ has size at most $(DB)^2 + (DB)$; thus, each subgraph fits into internal memory, and we can compute graphs $G'_1, \ldots, G'_l$ by loading

Input: A simple graph $G = (V, E)$.
Output: A planar embedding $\hat{G}$ of G or the answer that G is not planar.

1: **if** $|E| > 3|V| - 6$ **then**
2: Report that G is not planar and exit.
3: **end if**
4: Compute a set S of $O(N/(DB))$ vertices of G whose removal partitions G into $O(N/(DB)^2)$ subgraphs $G_1, \ldots, G_k$ such that $|G_i| \leq (DB)^2$ and $|\partial G_i| \leq DB$, for $1 \leq i \leq k$.
5: **if** Step 4 reports an error **then**
6: Report that G is not planar and exit.
7: **end if**
8: Let $\tilde{G}_1, \ldots, \tilde{G}_k$ be the graphs defined as $\tilde{G}_i = (V(G_i) \cup \partial G_i, \{\{v, w\} \in E : v \in V(G_i) \wedge w \in V(G_i) \cup \partial G_i\})$.
9: Let $G'_1, \ldots, G'_l$ be the connected components of graphs $\tilde{G}_1, \ldots, \tilde{G}_k$.
10: **for** $j = 1, \ldots, l$ **do**
11: **if** G'_j is not planar **then**
12: Report that G is not planar and exit.
13: **end if**
14: Compute the constraint graph C_j of G'_j.
15: **end for**
16: Let $A = G[S] \cup C_1 \cup \cdots \cup C_l$
17: **if** A is not planar **then**
18: Report that G is not planar and exit.
19: **end if**
20: Compute a planar embedding $\hat{A}$ of A.
21: **for** $i = 1, \ldots, j$ **do**
22: Let $\hat{C}_j$ be the restriction of embedding $\hat{A}$ to C_j.
23: Replace $\hat{C}_j$ by an embedding $\hat{G}'_j$ of G'_j.
24: **end for**
25: Let $\hat{G}$ be the resulting embedding of G.

Algorithm 2.1
Planarity algorithm.

graphs $\tilde{G}_1, \ldots, \tilde{G}_k$ into internal memory, one at a time, and partitioning each of them into its connected components. Similarly, Lines 10–15 take $O(\text{scan}(N))$ I/Os, as each subgraph G'_j is small enough to fit into internal memory. Line 16 requires sorting and scanning the vertex and edge sets of graphs $G[S], C_1, \ldots, C_l$, in order to eliminate multiple vertices and edges with the same name in different subgraphs. Thus, this step takes $O(\text{sort}(N))$ I/Os. By Lemma 7.2, graph A has size $O(N/(DB))$, so that Lines 17–20 take $O(N/(DB))$ I/Os. Lines 21–25 take $O(\text{sort}(N))$ I/Os, by Lemma 8.1. □

In Sections 3 through 6, we describe the construction of constraint graphs $C_1, \ldots, C_l$ from graphs $G'_1, \ldots, G'_l$, show that each such graph C_j has size linear in the number of separator vertices in G'_j, and that the graph $G[G'_j/C_j]$ is planar if and only if G is planar. An inductive application of this argument proves that A is planar if and only if G is planar. In Section 7, we prove that graph A has size $O(N/(DB))$. In Section 8, we show that a planar embedding $\hat{G}$ of G can be derived from a planar embedding $\hat{A}$ of A by locally replacing the restrictions $\hat{C}_1, \ldots, \hat{C}_l$ of $\hat{A}$ to subgraphs $C_1, \ldots, C_l$ with consistent embeddings $\hat{G}'_1, \ldots, \hat{G}'_l$ of graphs $G'_1, \ldots, G'_l$. In fact, this follows immediately from the properties of graphs $C_1, \ldots, C_l$ shown in Sections 3 through 6; but Section 8 provides important technical details of the replacement procedure.

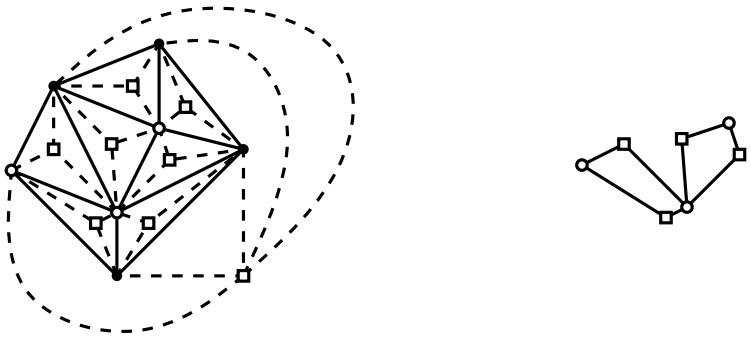
3 Computing the Constraint Graphs

The core of our algorithm is the construction of the constraint graphs $C_1, \ldots, C_l$ from graphs $G'_1, \ldots, G'_l$. Our construction ensures that graph $G[G'_j/C_j]$ is planar if and only if G is planar; a planar embedding $\hat{G}$ of G can be obtained from a planar embedding $\hat{G}[G'_j/C_j]$ of $G[G'_j/C_j]$ by locally replacing the embedding of C_j induced by $\hat{G}[G'_j/C_j]$ with a consistent embedding of G'_j.

We assume for the rest of the paper that graphs $G'_1, \ldots, G'_l$ are planar because otherwise Algorithm 2.1 correctly reports that G is non-planar, regardless of the correctness of the rest of the algorithm. The construction of graphs $C_1, \ldots, C_l$ partitions each graph G'_j into its bicomps $\mathcal{B}_{j,1}, \ldots, \mathcal{B}_{j,q_j}$, and each bicomp $\mathcal{B}_{j,k}$ into its tricomps $\mathcal{T}_{j,k,1}, \ldots, \mathcal{T}_{j,k,r_{j,k}}$. The constraint graph C_j of G'_j is now constructed in a bottom-up fashion from the constraint graphs $C_{\mathcal{T}_{j,k,l}}$ of tricomps $\mathcal{T}_{j,k,l}$. In particular, the constraint graph $C_{\mathcal{B}_{j,k}}$ of a bicomp $\mathcal{B}_{j,k}$ is computed by classifying the tricomps of $\mathcal{B}_{j,k}$ into two classes; "essential" tricomps are replaced by their constraint graphs; "inessential" tricomps are either completely removed, or groups of them are replaced by constraint graphs of constant size. The construction of C_j from constraint graphs $C_{\mathcal{B}_{j,1}}, \ldots, C_{\mathcal{B}_{j,q_j}}$ follows the same pattern. "Essential" bicomps are replaced by their constraint graphs; "inessential" bicomps are either removed, or groups of them are replaced by constraint graphs of constant size.

The classification of subgraphs as essential or inessential is closely tied to the concept of required vertices in such a subgraph. For any subgraph H of G, the *required* vertices of H are the vertices shared by H and $\bar{H}$. That is, for a graph G'_j, all separator vertices in G'_j are required; for a bicomp $\mathcal{B}$ all separator vertices and cutpoints in $\mathcal{B}$ are required; finally, for a tricomp $\mathcal{T}$ all separator vertices, cutpoints, and members of separation pairs are required. For any graph H, the vertex set of its constraint graph C_H has to contain at least the required vertices of H. To see why this is necessary, assume that there exists a path in H connecting two of its required vertices. If this path is part of a subgraph of G homeomorphic to K_5 or $K_{3,3}$, and one of the two required vertices is not present in C_H, $G[H/C_H]$ may be planar even though G is not.

The following sections follow the bottom-up construction of graphs $C_1, \ldots, C_l$. Section 4 describes the construction of the constraint graph $C_{\mathcal{T}}$ of a tricomp $\mathcal{T}$. Section 5 shows how to construct the constraint graph $C_{\mathcal{B}}$ of a bicomp $\mathcal{B}$, using the constraint graphs of the tricomps of $\mathcal{B}$ as building blocks. Section 6 describes how to assemble the constraint graph C_j of graph G'_j from the constraint graphs of the bicomps of G'_j.



(a) (b)

Figure 4.1
(a) A tricomp $\mathcal{T}$ with its face-on-vertex graph G_F. Required vertices are white discs; vertices that are not required are black discs; face vertices are squares. Edges of $\mathcal{T}$ are solid; edges of G_F are dashed. (b) The compressed face-on-vertex graph G'_F of $\mathcal{T}$.

4 The Constraint Graph of a Tricomp

Let $\mathcal{T}$ be a tricomp, and let $R(\mathcal{T})$ denote its set of required vertices. Our goal is to construct a constraint graph $C_\mathcal{T}$ for $\mathcal{T}$ whose vertex set has size $O(|R(\mathcal{T})|)$, which contains all virtual edges of $\mathcal{T}$, and such that G is planar if and only if $G[\mathcal{T}^\circ/C_\mathcal{T}^\circ]$ is planar. If $\mathcal{T}$ is a bond, the constraint graph $C_\mathcal{T}$ of $\mathcal{T}$ is $\mathcal{T}$ itself. If $\mathcal{T}$ is a cycle $\mathcal{T} = (v_1, \ldots, v_k)$, let $v'_1, \ldots, v'_l$ be the required vertices in $\mathcal{T}$, appearing in this order along $\mathcal{T}$. Then $C_\mathcal{T}$ is the cycle $C_\mathcal{T} = (v'_1, \ldots, v'_l)$. The rest of this section deals with the construction of $C_\mathcal{T}$ in the case when $\mathcal{T}$ is a triconnected simple graph.

Let $\hat{\mathcal{T}}$ be the unique planar embedding of $\mathcal{T}$. The *face-on-vertex graph* G_F of $\hat{\mathcal{T}}$ is defined as follows (Figure 4.1a): G_F contains all vertices of $\mathcal{T}$ as well as one vertex v_f per face f in $\hat{\mathcal{T}}$. There is an edge between a face vertex v_f and a vertex $w \in \mathcal{T}$ if and only if w appears on the boundary of face f. Graph G_F is planar. Ordering edges $\{v_f, w_1\}, \ldots, \{v_f, w_k\}$ around v_f in the same order as vertices $w_1 \ldots, w_k$ along the boundary of face f in $\hat{\mathcal{T}}$, we obtain a planar embedding $\hat{G}_F$ of G_F.

Our goal is to construct $C_\mathcal{T}$ so that the order of required vertices around the faces of $\hat{\mathcal{T}}$ is preserved in any embedding of $C_\mathcal{T}$. We remove all vertices in $V(\mathcal{T}) \setminus R(\mathcal{T})$ from G_F. Next we remove face vertices adjacent to at most one required vertex from G_F; but we ensure that the degree of any required vertex in G_F remains at least two. We call the resulting graph G'_F the *compressed face-on-vertex graph* of $\hat{\mathcal{T}}$ (Figure 4.1b).

Lemma 4.1 *The compressed face-on-vertex graph G'_F of a tricomp $\mathcal{T}$ is planar and has at most $10|R(\mathcal{T})|$ vertices.*

Proof. As G'_F is a subgraph of G_F, the planarity of G'_F is obvious. In fact, we will use the embedding $\hat{G}'_F$ of G'_F induced by the embedding $\hat{G}_F$ of G_F in the remainder of this section. In order to count the number of vertices in G'_F, we partition the vertices in $R(\mathcal{T})$ into two groups. The vertices in the first group, R_1, are adjacent only to face vertices of degree one. The vertices in the second group, R_2, have at least one neighbor of degree at least two.

The total number of face vertices adjacent to vertices in R_1 is $2|R_1|$. Every vertex in R_2 has at most one adjacent face vertex of degree one. In order to count the face vertices of degree two in G'_F, consider the subgraph H of G'_F induced by all vertices in R_2 and all face vertices of degree two in G'_F. We construct a

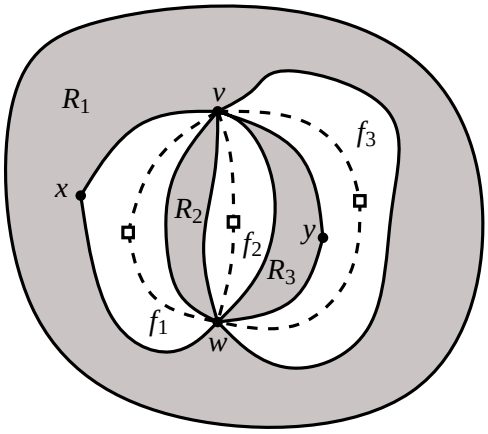


Figure 4.2
Proof of Lemma 4.1.

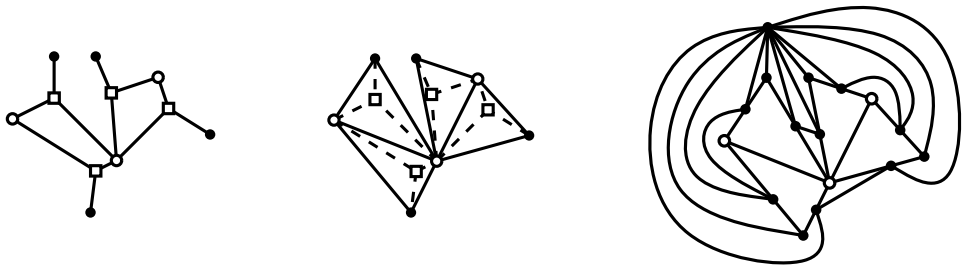
graph H' containing all vertices in R_2. There is an edge $\{v,w\} \in H'$ if there exists a face vertex in H which is adjacent to v and w. As H is a subgraph of G'_F, H is planar. A planar embedding of H' can easily be derived from a planar embedding of H. Hence, H' has at most $3|R_2|$ edges. We associate a face vertex x of degree two with edge $\{v,w\} \in H'$ if x is adjacent to v and w in H. We show that there are at most two face vertices associated with each edge in H', so that there are at most $6|R_2|$ face vertices of degree two in G'_F.

Let v and w be two vertices so that edge $\{v,w\}$ has three face vertices v_{f_1}, v_{f_2}, and v_{f_3} associated with it. See Figure 4.2. Then $\mathbb{R}^2 \setminus (v \cup w \cup f_1 \cup f_2 \cup f_3)$ consists of three disjoint regions R_1, R_2, and R_3. Only one of these regions can be degenerate, i.e., consist only of the embedding of edge $\{v,w\}$. W.l.o.g., assume that R_2 is this region. Then R_1 contains some vertex x, and R_3 contains some vertex y. Any path from x to y must pass through either v or w, as there are no edges crossing faces f_1, f_2, and f_3. Thus, $\{v,w\}$ is a separation pair, contradicting the triconnectivity of $\mathcal{T}$.

The subgraph H'' of G'_F induced by all vertices in R_2 and all face vertices of degree at least 3 is planar and bipartite, where $V_1 = R_2$ and V_2 contains all face vertices of degree at least 3. Hence, by Lemma 1.1, $|V_2| \leq 2|V_1| = 2|R_2|$. Thus, G'_F has at most $(|R_1| + 2|R_1|) + (|R_2| + |R_2| + 6|R_2| + 2|R_2|) \leq 10|R(\mathcal{T})|$ vertices. □

Next we use G'_F to construct the constraint graph $C_\mathcal{T}$ of $\mathcal{T}$. In order to construct $C_\mathcal{T}$, we augment G'_F so that we can construct a graph H whose face-on-vertex graph has G'_F as a subgraph. In particular, every face-vertex in G'_F must have degree at least three in order to represent a valid face. Thus, for each face vertex v_f in G'_F of degree at most two, we add one or two dummy vertices and make them adjacent to v_f. If v_f has degree two, let x and y be the two required vertices adjacent to v_f in G'_F. If there is an edge between x and y in $\mathcal{T}$, assume that x, edge $\{x,y\}$, and y appear in this order clockwise along f's boundary. Then we add the dummy vertex z incident to v_f clockwise between y and x in the embedding $\hat{G}'_F$. This construction is illustrated in Figure 4.3a for the tricomp $\mathcal{T}$ shown in Figure 4.1a.

We construct a graph H whose face-on-vertex graph has G'_F as a subgraph. Graph H has the required and dummy vertices of G'_F as vertices. There is an edge $\{x,y\}$ in H if there exists a face-vertex v_f in G'_F such that edges $\{v_f,x\}$ and $\{v_f,y\}$ appear consecutively in the clockwise order around v_f. This construction is shown in Figure 4.3b. The embedding $\hat{H}$ of graph H has a number of faces which are not represented by face vertices in G'_F. (In Figure 4.3b the only such face is the outer face.) In order to construct $C_\mathcal{T}$, we augment H so that the resulting graph $C_\mathcal{T}$ is triconnected and has the property that every face in the unique planar embedding of $C_\mathcal{T}$ which does not correspond to a face vertex in G'_F has at most one required vertex



(a) (b) (c)

Figure 4.3
(a) The compressed face-on-vertex graph G'_F of tricomp $\mathcal{T}$ shown in Figure 4.1a, augmented with dummy vertices. (b) The graph H whose face-on-vertex graph has the augmented face-on-vertex graph G'_F as a subgraph. (c) The constraint graph $C_{\mathcal{T}}$ of $\mathcal{T}$.

on its boundary. As a result, there exists a natural bijection between the faces of $\hat{\mathcal{T}}$ and $\hat{C}_{\mathcal{T}}$ with at least two required vertices on their boundaries.

The augmentation of H proceeds in three phases. First we iterate over all faces of $\hat{H}$ which do not correspond to face vertices in G'_F. For every boundary cycle of such a face f, let $v_1, \ldots, v_k$ be the required vertices in that cycle. (Note that this cycle is not necessarily simple, and some required vertices may appear more than once along the cycle.) For each vertex v_i, we split the two edges preceding and succeeding v_i in the cycle by adding two dummy vertices u_i and w_i on these two edges; we connect vertices u_i and w_i by an edge $\{u_i, w_i\}$. As a result face f is partitioned into a number of triangles $\{u_i, v_i, w_i\}$ and a face f' which does not have any required vertex on its boundary. If H is disconnected, the graph obtained after this augmentation is also disconnected. This is equivalent to some face f' having more than one boundary cycle. As long as there is such a face f', we choose two of its boundary cycles and two pairs of consecutive vertices $\{x, y\}$ and $\{u, v\}$ on these two cycles. Let x, y and u, v appear in this order clockwise along these two cycles. Then we add edges $\{x, v\}, \{x, u\}, \{y, u\}$ to H, thereby concatenating the two boundary cycles. Once this procedure is finished, every face of H has a single boundary cycle. We triangulate each face f' not corresponding to a vertex in G'_F by adding a dummy vertex in the center of f' and connecting this vertex to every vertex on the boundary of f'. The resulting graph is the constraint graph $C_{\mathcal{T}}$ of $\mathcal{T}$.

Lemma 4.2 *The constraint graph $C_{\mathcal{T}}$ of a tricomp $\mathcal{T}$ is a planar graph with $O(|R(\mathcal{T})|)$ vertices.*

Proof. For bonds and cycles the claim trivially holds, as all vertices in $C_{\mathcal{T}}$ are required. If $\mathcal{T}$ is a triconnected simple graph, we show the lemma as follows. By Lemma 4.1, graph G'_F contains at most $2|R(\mathcal{T})|$ face vertices of degree one and at most $6|R(\mathcal{T})|$ face vertices of degree two. Thus, we add at most $10|R(\mathcal{T})|$ dummy vertices to G'_F, in order to increase the degree of each face vertex to at least three. Hence, graph H is a planar graph with at most $11|R(\mathcal{T})|$ vertices. The construction of $C_{\mathcal{T}}$ from H adds at most one vertex per edge of H and at most one vertex per face of H, $55|R(\mathcal{T})|$ vertices in total. Thus, $C_{\mathcal{T}}$ has at most $66|R(\mathcal{T})|$ vertices. The planarity of $C_{\mathcal{T}}$ is explicitly guaranteed by the above construction. □

Lemma 4.3 *The constraint graph $C_{\mathcal{T}}$ of a triconnected simple graph $\mathcal{T}$ is a triconnected simple graph.*

Proof. To show that $C_{\mathcal{T}}$ does not contain multiple edges, observe that graph H constructed from G'_F does not have multiple edges: As we add dummy vertices separately for each face vertex of G'_F, the only edges that may be duplicated in H are those both of whose endpoints are required vertices. If there are two faces

f_1 and f_2 sharing vertices v and w in H, the two corresponding faces in $\mathcal{T}$ share v and w as well. Hence, they share edge $\{v,w\}$, by the triconnectivity of $\mathcal{T}$ and Lemma 1.2. In this case, our construction guarantees that no dummy vertices are inserted between v and w in the embedding. Thus, edge $\{v,w\}$ is shared by f_1 and f_2. All edges that are subsequently added to H during the construction of $C_{\mathcal{T}}$ are added between vertices that are not in H; it is easy to verify that we add each such edge only once. Hence, $C_{\mathcal{T}}$ is a simple graph.

To show that $C_{\mathcal{T}}$ is triconnected, let $\hat{C}_{\mathcal{T}}$ be the planar embedding of $C_{\mathcal{T}}$ derived from the planar embedding $\hat{\mathcal{T}}$ of $\mathcal{T}$ using the above construction. By Lemma 1.2, it is sufficient to show that for all faces f_1 and f_2 sharing two vertices v and w, edge $\{v,w\}$ is on the boundary of both f_1 and f_2.

To prove this, we partition the faces of $\hat{C}_{\mathcal{T}}$ into two categories: *Required* faces are those that correspond to face vertices in G'_F; all other faces are *auxiliary* faces.

If f_1 and f_2 are both required faces, then v and w are required vertices. This is true because dummy vertices are created separately for each face vertex of G'_F, so that no two required faces share a dummy vertex. Faces f_1 and f_2 correspond to two faces f'_1 and f'_2 of $\mathcal{T}$ sharing vertices v and w. Thus, by Lemma 1.2, edge $\{v,w\}$ must be on the boundary of both f'_1 and f'_2. As shown above, edges between required vertices are preserved in $C_{\mathcal{T}}$.

It is easy to verify that all auxiliary faces of $\hat{C}_{\mathcal{T}}$ are triangles. Thus, if f_1 and f_2 are both auxiliary faces, edge $\{v,w\}$ is on the boundary of both f_1 and f_2.

If f_1 and f_2 are of different types, assume w.l.o.g. that f_1 is required and f_2 is an auxiliary face. As f_2 is a triangle, edge $\{v,w\}$ is on the boundary of face f_2. Thus, it remains to show that edge $\{v,w\}$ is on the boundary of face f_1.

Let f be the face of H containing face f_2. Face f is split into two types of faces: triangles produced by bridging required vertices on the boundary of f and triangles produced by triangulating the resulting face f'. If f_2 is of the former type, f_1 and f_2 can only share vertices u_i and v_i or vertices v_i and w_i because vertices u_i and w_i are on the boundary of different required faces. Assume w.l.o.g. that f_1 and f_2 share vertices u_i and v_i. Vertex u_i has been inserted on an edge $\{x,v_i\}$ on the boundary of a required face f_0, so that u_i, v_i, and edge $\{u_i,v_i\}$ are on the boundary of face f_0. As required faces are not partitioned by our algorithm, and f_0 is the only required face having vertex u_i on its boundary, we have $f_1 = f_0$. Hence, edge $\{u_i,v_i\}$ is on the boundary of face f_1.

If f_2 is of the latter type, the edge shared by f_1 and f_2 is on the boundary of face f'. By the above argument, no face in f' can share two vertices u_i and w_i with a required face. Also, no face can share the central vertex x of f' with a required face. All other pairs of vertices on the boundary of triangles in f' correspond to edges on the boundaries of required faces. □

Lemma 4.4 *Let $\hat{\mathcal{T}}$ be a planar embedding of $\mathcal{T}$, and $\hat{C}_{\mathcal{T}}$ be a planar embedding of $C_{\mathcal{T}}$. Let $f_1,\ldots,f_k$ be the faces of $\hat{\mathcal{T}}$ with at least two required vertices on their boundaries, and $f'_1,\ldots,f'_l$ be the faces of $\hat{C}_{\mathcal{T}}$ with at least two required vertices on their boundaries. Then $k=l$, and there exists a bijection $\sigma:[1,k]\to[1,k]$ such that faces f_i and $f'_{\sigma(i)}$ have the same required vertices on their boundaries, in the same order.*

Proof. The lemma holds trivially for bonds and simple cycles.

If $\mathcal{T}$ is a triconnected simple graph, $C_{\mathcal{T}}$ is triconnected by Lemma 4.3. Hence, embeddings $\hat{\mathcal{T}}$ and $\hat{C}_{\mathcal{T}}$ are unique. In particular, $\hat{C}_{\mathcal{T}}$ is the embedding of $C_{\mathcal{T}}$ derived from $\hat{\mathcal{T}}$ by our construction above. Faces $f_1,\ldots,f_k$ in $\hat{\mathcal{T}}$ correspond to face vertices $w_1,\ldots,w_k$ in G'_F, which in turn correspond to faces $f'_1,\ldots,f'_k$ in $\hat{C}_{\mathcal{T}}$. Moreover, it is easily checked that our construction preserves the order of required vertices around these faces.

Thus, in order to prove the lemma, we have to show that each auxiliary face has at most one required vertex on its boundary. Every auxiliary face is the result of partitioning a face f of graph H in the above construction which does not correspond to a vertex in G'_F. Face f is partitioned into triangles $\{u_i,v_i,w_i\}$, for each required vertex v_i on the boundary of f, and a face f' which does not have any required vertices

on its boundary. Vertices u_i and w_i on the boundary of a triangle $\{u_i, v_i, w_i\}$ are dummy vertices. Face f' is partitioned into triangles none of which has a required vertex on its boundary. Thus, no auxiliary face has more than one required vertex on its boundary. □

Let $\mathcal{T}_1, \ldots, \mathcal{T}_q$ be the tricomps of graphs $G'_1, \ldots, G'_l$. Then we define a sequence of graphs $G_0^{(1)}, \ldots, G_q^{(1)}$ such that $G_0^{(1)} = G$ and $G_i^{(1)} = G_{i-1}^{(1)}[\mathcal{T}_i^\circ / C_{\mathcal{T}_i}^\circ]$, for $1 \leq i \leq q$. Graph $G^{(1)}$ is defined as $G^{(1)} = G_q^{(1)}$.

Lemma 4.5 *Graph $G_i^{(1)}$ is planar if and only if $G_{i-1}^{(1)}$ is planar, for $1 \leq i \leq q$. A planar embedding of $G_{i-1}^{(1)}$ can be obtained by locally replacing the embedding of $C_{\mathcal{T}_i}^\circ$ induced by a planar embedding $\hat{G}_i^{(1)}$ of $G_i^{(1)}$ with a consistent embedding of $\mathcal{T}_i^\circ$.*

Proof. Consider an embedding $\hat{G}_{i-1}^{(1)}$ of $G_{i-1}^{(1)}$. Let $\hat{\mathcal{T}}_i$ be the unique planar embedding of $\mathcal{T}_i$, and $\hat{\mathcal{T}}_i^\circ$ be the planar embedding of $\mathcal{T}_i^\circ$ obtained from $\hat{\mathcal{T}}_i$ by removing all virtual edges in $\mathcal{T}_i$. Let $\hat{C}_{\mathcal{T}_i}$ be the unique planar embedding of $C_{\mathcal{T}_i}$, and $\hat{C}_{\mathcal{T}_i}^\circ$ be the planar embedding of $C_{\mathcal{T}_i}^\circ$ obtained from $\hat{C}_{\mathcal{T}_i}$ by removing all virtual edges in $C_{\mathcal{T}_i}$.

We partition $\bar{\mathcal{T}}_i^\circ$ into maximal subgraphs each of which is embedded inside a face f of $\hat{\mathcal{T}}_i^\circ$. Each such graph K is incident only to required vertices on the boundary of f. If f is the result of merging a number of faces of $\hat{\mathcal{T}}_i$ by removing virtual edges, each constituent face of f in $\hat{\mathcal{T}}_i$ has at least two required vertices on its boundary. Thus, these constituent faces are preserved in $\hat{C}_{\mathcal{T}_i}$, by Lemma 4.4. Moreover, they share the same virtual edges in $\hat{C}_{\mathcal{T}_i}$ as in $\hat{\mathcal{T}}_i$. Thus, face f has a corresponding face f' in $\hat{C}_{\mathcal{T}_i}^\circ$ with the same required vertices on its boundary, in the same order. If f is a face of $\hat{\mathcal{T}}_i^\circ$ consisting of a single face of $\hat{\mathcal{T}}_i$ with at least two required vertices on its boundary, this face is preserved in $\hat{C}_{\mathcal{T}_i}$ and thus in $\hat{C}_{\mathcal{T}_i}^\circ$, by Lemma 4.4. Thus, in both cases, we can embed K inside f'. If face f has only one required vertex on its boundary, K can be embedded inside an arbitrary face of $\hat{C}_{\mathcal{T}_i}^\circ$ with this vertex on its boundary. Embedding all subgraphs K in this manner, we obtain a planar embedding of $G_i^{(1)}$ from $\hat{G}_{i-1}^{(1)}$. The proof that $G_{i-1}^{(1)}$ is planar if $G_i^{(1)}$ is planar is similar. □

Corollary 4.1 *Graph $G^{(1)}$ is planar if and only if graph G is planar. A planar embedding of G can be obtained by locally replacing the embeddings of graphs $C_{\mathcal{T}_1}^\circ, \ldots, C_{\mathcal{T}_q}^\circ$ with consistent embeddings of graphs $\mathcal{T}_1^\circ, \ldots, \mathcal{T}_q^\circ$.*

5 The Constraint Graph of a Bicomp

Given graph $G^{(1)}$, which was constructed by replacing each tricomp of graphs $G'_1, \ldots, G'_l$ with its constraint graph, we show next how to construct the constraint graph of a bicomp $\mathcal{B}$. In order to do this, we use a two-step procedure to classify the tricomps of $\mathcal{B}$ as essential or inessential. For an essential tricomp $\mathcal{T}$ of $\mathcal{B}$, we leave the constraint graph $C_\mathcal{T}$ of $\mathcal{T}$ in $G^{(1)}$ unchanged. An inessential tricomp is either completely removed from $G^{(1)}$, or it is grouped together with other inessential tricomps; each such group is replaced with a constraint graph of constant size.

Let $R(\mathcal{B})$ be the set of required vertices of $\mathcal{B}$, and $\mathcal{T}_1, \ldots, \mathcal{T}_q$ be its tricomps. We define the *tricomp tree* $T = T_3(\mathcal{B})$ as follows (see Figure 5.1): Tree T has q vertices $\tau_1, \ldots, \tau_q$, one per tricomp $\mathcal{T}_i$. There is an edge $\{\tau_i, \tau_j\} \in T$ if tricomps $\mathcal{T}_i$ and $\mathcal{T}_j$ share a virtual edge. By the recursive definition of the tricomps of a biconnected simple graph, T is indeed a tree. For each vertex $v \in R(\mathcal{B})$, we choose a tricomp $\mathcal{T}(v)$ such that $v \in \mathcal{T}(v)$. We call a tricomp $\mathcal{T}_i$ *essential* if there is a vertex $v \in R(\mathcal{B})$ such that $\mathcal{T}_i = \mathcal{T}(v)$. A tricomp $\mathcal{T}_j$ is *potentially essential* if there are two essential tricomps $\mathcal{T}_i$ and $\mathcal{T}_k$ such that vertex τ_j is on the path from τ_i to τ_k in T. All other tricomps are *inessential*. In the next section, we show that removing all inessential

tricomps from $G^{(1)}$ does not alter its (non-)planarity. Then we finish the classification of the tricomps by deciding which of the potentially essential tricomps are essential, and which are inessential. In Section 5.2, we replace all tricomps classified as inessential in this second round of classification by a small number of constant size constraint graphs. Section 5.3 puts the pieces together and shows that the final constraint graph $C_{\mathcal{B}}$ of $\mathcal{B}$ has size $O(|R(\mathcal{B})|)$.

5.1 Discarding Inessential Tricomps

Let T' be the tree obtained by removing all vertices τ_j corresponding to inessential tricomps $\mathcal{T}_j$ from T. Let $\mathcal{B}'$ be the subgraph of $\mathcal{B}$ obtained by merging all tricomps corresponding to vertices in T'. The nodes $\tau_j \in T$ corresponding to inessential tricomps $\mathcal{T}_j$ induce a set of maximal subtrees $T_1, \ldots, T_s$ of T. Each such subtree T_j is connected to T' through a single edge. Let K_j be the subgraph of $\mathcal{B}$ obtained by merging all tricomps corresponding to the nodes in tree T_j. As T_j and T' share only a single edge, K_j and $\mathcal{B}'$ share exactly one virtual edge (v_j, w_j, i_j). Let $\mathcal{B}''$ be the graph obtained by replacing graphs $K_1^\circ, \ldots, K_s^\circ$ with edges $(v_1, w_1, i_1), \ldots, (v_s, w_s, i_s)$ in $\mathcal{B}$. Alternatively, $\mathcal{B}''$ is obtained by making all virtual edges in $\mathcal{B}'$ non-virtual.

Let $\mathcal{B}_1, \ldots, \mathcal{B}_q$ be the bicomps of graphs $G'_1, \ldots, G'_l$. Then we define a sequence of graphs $G_0^{(2)}, \ldots, G_q^{(2)}$, where $G_0^{(2)} = G^{(1)}$ and $G_i^{(2)} = G_{i-1}^{(2)}[\mathcal{B}_i / \mathcal{B}_i'']$. Graph $G^{(2)}$ is defined as $G^{(2)} = G_q^{(2)}$.

Lemma 5.1 *Graph $G_i^{(2)}$ is planar if and only if graph $G_{i-1}^{(2)}$ is planar, for $1 \leq i \leq q$. A planar embedding of $G_{i-1}^{(2)}$ can be obtained by locally replacing edges in an embedding of $G_i^{(2)}$ with embeddings of inessential tricomps of $\mathcal{B}_i$.*

Proof. Let $\mathcal{B} = \mathcal{B}_i$, and let trees T, T', and $T_1, \ldots, T_s$ and graphs $\mathcal{B}'$ and $K_1, \ldots, K_s$ be defined as above. Let $\hat{G}_{i-1}^{(2)}$ be a planar embedding of graph $G_{i-1}^{(2)}$. For each subgraph K_j of $\mathcal{B}$, there is a path from v_j to w_j in K_j°. Thus, replacing K_j° by edge (v_j, w_j, i_j) in $\hat{G}_{i-1}^{(2)}$ corresponds to removing the whole graph K_j° except that path from G, and then replacing this path by a single edge. As all tricomps in K_j are inessential, v_j and w_j are the only vertices shared by K_j° and $\bar{K}_j^\circ$. Hence, two paths from vertices v_j to w_j in graph K_j° and from v_k to w_k in K_k° are internally vertex disjoint, so that replacing both paths by a single edge does not introduce an intersection in the planar embedding of $\hat{G}_{i-1}^{(2)}$. Applying this argument to graphs $K_1^\circ, \ldots, K_s^\circ$ in turn shows that $G_i^{(2)}$ is planar if $G_{i-1}^{(2)}$ is planar.

To show that $G_{i-1}^{(2)}$ is planar if $G_i^{(2)}$ is planar, recall that each graph K_j° shares only vertices v_j and w_j with $\bar{K}_j^\circ$. Hence, we can obtain an embedding of $G_{i-1}^{(2)}$ from an embedding of $G_i^{(2)}$ by replacing each edge (v_j, w_j, i_j) in $G_i^{(2)}$ with an embedding of graph K_j° which has vertices v_j and w_j on its outer face. The existence of such an embedding follows immediately from the planarity of the tricomps of a biconnected planar graph. □

Corollary 5.1 *Graph $G^{(2)}$ is planar if and only if graph $G^{(1)}$ is planar. A planar embedding of $G^{(1)}$ can be obtained by locally replacing edges in an embedding of $G^{(2)}$ with embeddings of inessential tricomps.*

Having disposed of the first set of inessential tricomps, we now classify the potentially essential tricomps of $\mathcal{B}''$ as essential or inessential. A potentially essential tricomp is essential if its corresponding vertex in T' has degree at least three; otherwise, it is inessential. Note that all vertices in T' whose corresponding tricomps are inessential have degree two. This is true because all leaves correspond to essential tricomps, and all tricomps corrsponding to internal nodes of degree at least three have been declared essential. We partition the set of nodes corresponding to inessential tricomps into maximal paths in T'.

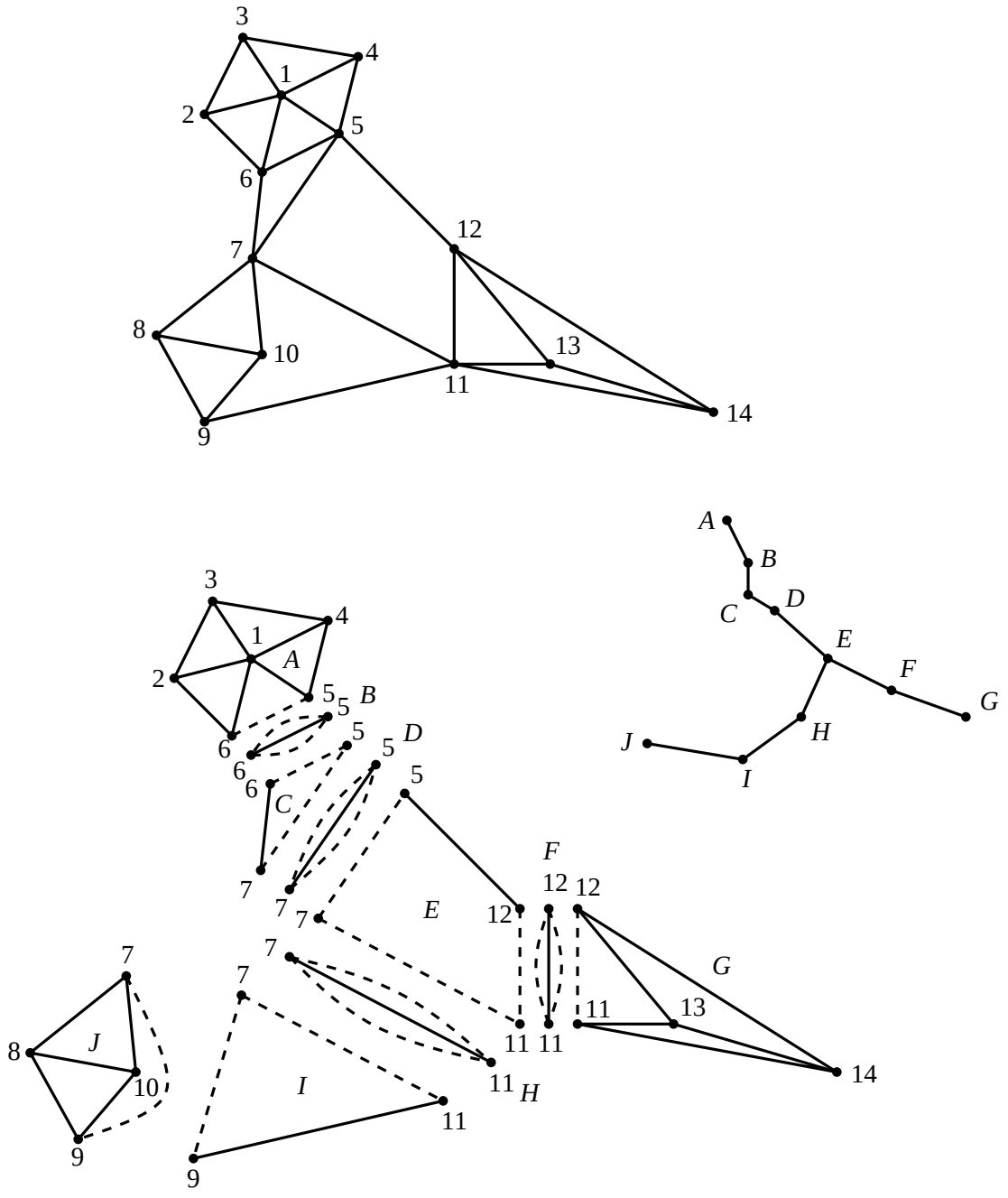


Figure 5.1
A biconnected planar graph *G*, its tricomps, and its tricomp tree. Tricomps are labeled with capital letters. Virtual edges in the tricomps are dashed.

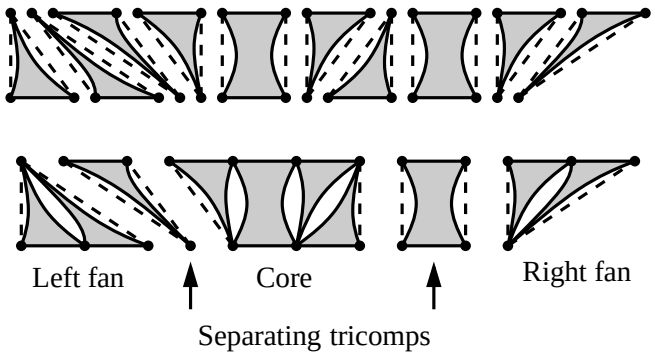


Figure 5.2
Tricomps $\mathcal{T}_1, \ldots, \mathcal{T}_s$ and the five graphs into which the merge of these tricomps is being decomposed.

Let $P = (\tau_1, \ldots, \tau_s)$ be such a path, and let τ_0 and τ_{s+1} be the two other neighbors of τ_1 and τ_s, respectively. Tricomps $\mathcal{T}_0$ and $\mathcal{T}_{s+1}$ are essential. Let (v_j, w_j, i_j) be the virtual edge shared by tricomps $\mathcal{T}_j$ and $\mathcal{T}_{j+1}$, $0 \leq j \leq s$. Let H_P be the graph obtained by merging tricomps $\mathcal{T}_1, \ldots, \mathcal{T}_s$. Then H_P contains two virtual edges: (v_0, w_0, i_0) and (v_s, w_s, i_s). Also, as all tricomps in H_P are inessential, graph H_P° shares only vertices v_0, w_0, v_s, and w_s with $\overline{H_P^\circ}$.

5.2 Compressing Chains of Inessential Tricomps

In this section, we construct a constraint graph C_P for each graph H_P induced by a path P of vertices in T' which correspond to inessential tricomps, and replace H_P° with C_P°. Graph C_P has size $O(1)$; replacing H_P° with C_P° in $\mathcal{B}$ preserves the (non-)planarity of $G^{(2)}$.

To construct C_P, we partition tricomps $\mathcal{T}_1, \ldots, \mathcal{T}_s$ into five (possibly empty) groups, and replace each such group by its own constraint graph of constant size.

The *fan* of tricomp $\mathcal{T}_0$ is defined as follows: Let j_0 be the maximal index such that $\{v_{j_0}, w_{j_0}\} \cap \{v_0, w_0\} \neq \emptyset$. Then the fan of tricomp $\mathcal{T}_0$ is the union of tricomps $\mathcal{T}_1, \ldots, \mathcal{T}_{j_0}$. The fan of $\mathcal{T}_0$ is empty if $j_0 = 0$.

Analogously, the *fan* of tricomp $\mathcal{T}_{s+1}$ is defined as follows: Let j_s be the minimal index such that $\{v_{j_s}, w_{j_s}\} \cap \{v_s, w_s\} \neq \emptyset$. Then the fan of tricomp $\mathcal{T}_{s+1}$ is the union of tricomps $\mathcal{T}_{j_s+1}, \ldots, \mathcal{T}_s$. The fan of $\mathcal{T}_{s+1}$ is empty if $j_s = s$.

If $j_0 < j_s$, let $\mathcal{T}_{j_0+1}$ be the *separating tricomp* for $\mathcal{T}_0$. If $j_0 < j_s - 1$, let $\mathcal{T}_{j_s}$ be the *separating tricomp* for $\mathcal{T}_{s+1}$. If $j_0 < j_s - 2$, let the union of tricomps $\mathcal{T}_{j_0+2}, \ldots, \mathcal{T}_{j_s-1}$ be the *core* of graph H_P. Figure 5.2 illustrates this construction. We replace each non-empty fan, separating tricomp, and core by its own constraint graph of constant size. For a separating tricomp $\mathcal{T}$, we keep the constraint graph $C_\mathcal{T}$ constructed in Section 4. The constraint graphs of fans and cores are described next.

5.2.1 The Constraint Graph of a Fan

Let $\mathcal{F}$ be a fan of some graph H_P with virtual edges (a, b, i) and (a, c, j). For a non-empty fan $\mathcal{F}$, we have to distinguish two cases. If $b = c$, the fan consists of a single bond. In this case the constraint graph of $\mathcal{F}$ is the constraint graph of the bond, which is the bond itself. Otherwise, we are interested in capturing the possible embeddings of vertices a, b, and c and edges (a, b, i) and (a, c, j). In particular, we have to capture the following possibilities:

(a) Fan $\mathcal{F}$ has a planar embedding $\hat{\mathcal{F}}$ such that there are two faces with edges (a,b,i) and (a,c,j) on their boundaries.

(b) Fan $\mathcal{F}$ has a planar embedding $\hat{\mathcal{F}}$ such that there is one face with edges (a,b,i) and (a,c,j) on its boundary, and another face with edge (a,b,i) and vertex c on its boundary. (There is a symmetric case where the second face has edge (a,c,j) and vertex b on its boundary.)

(c) Fan $\mathcal{F}$ has a planar embedding $\hat{\mathcal{F}}$ such that there is one face with edges (a,b,i) and (a,c,j) on its boundary, and another face with vertices a, b, and c on its boundary.

(d) Fan $\mathcal{F}$ has a planar embedding $\hat{\mathcal{F}}$ such that there is one face with edges (a,b,i) and (a,c,j) on its boundary.

(e) Fan $\mathcal{F}$ has a planar embedding $\hat{\mathcal{F}}$ such that there is at least one face with vertices b and c on its boundary.

(f) For every face of any embedding $\hat{\mathcal{F}}$ of $\mathcal{F}$, the required vertices on its boundary are either in $\{a,b\}$ or in $\{a,c\}$.

Figure 5.3 shows fans $\mathcal{F}$ illustrating these six possibilities and the constraint graphs we construct in each of these cases. It is easy to verify that if fan $\mathcal{F}$ satisfies one of the above conditions, then its constraint graph $C_{\mathcal{F}}$ satisfies the same condition, and vice versa.

Observe that there are always a face with vertices a and b and a face with vertices a and c on their boundaries, as $\mathcal{F}$ contains edges (a,b,i) and (a,c,j). Thus, the distinction is whether there is a face with vertices b and c on its boundary. For Cases (a)–(e), such a face exists. For Case (f), such a face does not exist. Cases (a)–(e) further distinguish whether there are faces that have all three vertices on their boundaries. In Cases (a)–(d) such a face exists; in Case (e) it does not. Now observe that if there is a face with vertices a, b, and c on its boundary, we can ensure that it has edges (a,b,i) and (a,c,j) on its boundary, by embedding these two edges inside that face. Thus, the only difference between Cases (a)–(d) is whether there is a second face with all three vertices on its boundary and whether this second face has none, one, or both of the virtual edges on its boundary. Thus, Cases (a)–(f) are the only possibilities for the structure of fan $\mathcal{F}$.

It is easy to test which of the six cases applies: To test for Case (a), we add two extra vertices x and y on the two virtual edges and connect each of the vertices a, b, c, x, and y to two vertices z_1 and z_2 representing the two faces in Case (a). Case (a) applies if and only if the resulting graph is planar. To test for Case (b), we remove edge $\{y,z_2\}$. (In order to test for the symmetric case, we remove edge $\{x,z_2\}$.) To test for Case (c), we remove both edges $\{x,z_2\}$ and $\{y,z_2\}$. To test for Case (d), we remove vertex z_2 and all incident edges. To test for Case (e), we remove edges $\{a,z_1\}$, $\{x,z_1\}$, and $\{y,z_1\}$. If none of these graphs is planar, Case (f) applies.

Let $\mathcal{F}_1,\ldots,\mathcal{F}_q$ be the fans of all graphs H_P in $G^{(2)}$. We define a sequence $G_0^{(3)},\ldots,G_q^{(3)}$ of graphs where $G_0^{(3)} = G^{(2)}$ and $G_i^{(3)} = G_{i-1}^{(3)}[\mathcal{F}_i^\circ / C_{\mathcal{F}_i}^\circ]$, for $1 \le i \le q$. Graph $G^{(3)}$ is defined as $G^{(3)} = G_q^{(3)}$.

Lemma 5.2 *Graph $G_i^{(3)}$ is planar if and only if $G_{i-1}^{(3)}$ is planar, for $1 \le i \le q$. A planar embedding of $G_{i-1}^{(3)}$ can be obtained from a planar embedding of $G_i^{(3)}$ by locally replacing the embedding of $C_{\mathcal{F}_i}^\circ$ with a consistent embedding of $\mathcal{F}_i^\circ$.*

Proof. Consider a planar embedding $\hat{G}_{i-1}^{(3)}$ of graph $G_{i-1}^{(3)}$. Let $\hat{\mathcal{F}}_i^\circ$ be the planar embedding of $\mathcal{F}_i^\circ$ induced by $\hat{G}_{i-1}^{(3)}$. We partition $\overline{\mathcal{F}}_i^\circ$ into maximal subgraphs so that each such subgraph K is embedded inside a face of $\hat{\mathcal{F}}_i^\circ$. Denote the two virtual edges in $\mathcal{F}_i$ by (a,b,j) and (a,c,k).

If $\hat{\mathcal{F}}_i$ has at least one face with vertices a, b, and c on its boundary, one of Cases (a)–(d) applies. It is an exercise to verify that in all three cases, the embeddings $\hat{\mathcal{F}}_i^\circ$ and $\hat{C}_{\mathcal{F}_i}^\circ$ as shown in Figure 5.3 have the same

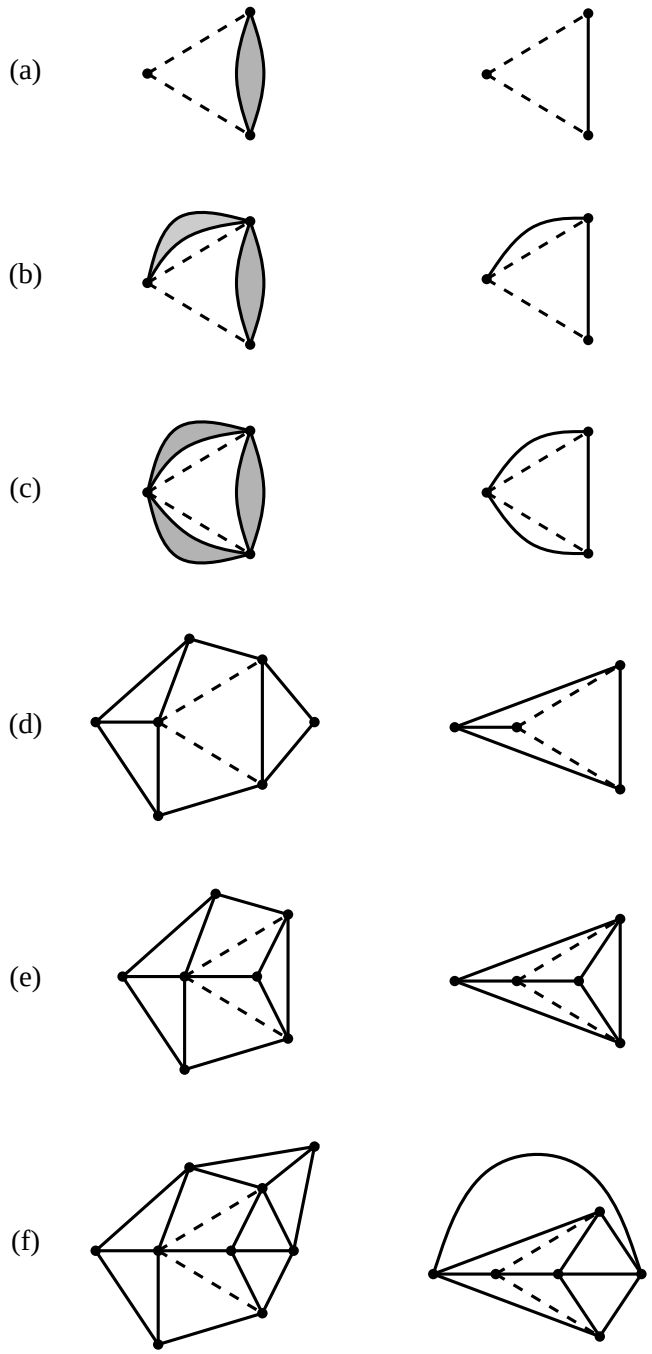


Figure 5.3
Fans illustrating the different possible constellations of virtual edges (a, b, i) and (a, c, j). The left graph in each figure is a fan $\mathcal{F}$. The right graph is its constraint graph $C_{\mathcal{F}}$.

faces with all three vertices on their boundaries. Thus, we can embed a graph K embedded inside a face of $\hat{\mathcal{F}}_i^\circ$ with all three vertices a, b, and c on its boundary in the corresponding face of $\hat{C}^\circ_{\mathcal{F}_i}$.

Now consider the faces of $\hat{\mathcal{F}}_i^\circ$ with two vertices on their boundaries. In all cases, but Cases (b) and (f), such a face f can have any two of the vertices $\{a,b,c\}$ on its boundary. However, in all cases, but Cases (b) and (f), there exists a face f' in $\hat{C}^\circ_{\mathcal{F}_i}$ such that the two vertices appear consecutively[1] along f'. Thus, we can embed a graph K embedded inside f inside face f' without intersecting any other graphs embedded inside f'. In Case (b), a face f with two required vertices on its boundary has either a and b or b and c on its boundary. Embedding $\hat{C}^\circ_{\mathcal{F}_i}$ has two faces such that a and b (resp. b and c) appear consecutively on the boundary of those faces. In Case (f), a face f with two required vertices on its boundary has either a and b or a and c on its boundary. Again, embedding $\hat{C}^\circ_{\mathcal{F}_i}$ has two faces such that a and b (resp. a and c) appear consecutivly on the boundary of those faces.

Any graph K embedded in a face f of $\hat{\mathcal{F}}_i^\circ$ which has only one required vertex on its boundary can be embedded inside any face of $\hat{C}^\circ_{\mathcal{F}_i}$ which has the same required vertex on its boundary, without creating any conflicts.

Now assume that we are given a planar embedding $\hat{G}_i^{(3)}$ of $G_i^{(3)}$. Let $\hat{C}_{\mathcal{F}_i}$ be the embedding of $C_{\mathcal{F}_i}$ induced by $\hat{G}_i^{(3)}$, and let $\hat{C}^\circ_{\mathcal{F}_i}$ be the restriction of $\hat{C}_{\mathcal{F}_i}$ to $C^\circ_{\mathcal{F}_i}$. It is easy to verify that in each of the above cases, there exists an embedding $\hat{\mathcal{F}}_i$ of $\mathcal{F}_i$ such that for every face of $\hat{C}^\circ_{\mathcal{F}_i}$, there exists a corresponding face in the restriction $\hat{\mathcal{F}}_i^\circ$ of $\hat{\mathcal{F}}_i$ to $\mathcal{F}_i^\circ$ with the same required vertices on its boundary. Moreover, if there are two faces in $\hat{C}^\circ_{\mathcal{F}_i}$ with three required vertices on their boundaries, then there are two such faces in $\hat{\mathcal{F}}_i^\circ$. Using the same arguments as above, this implies that $G_{i-1}^{(3)}$ is planar if $G_i^{(3)}$ is planar. □

Corollary 5.2 *Graph $G^{(3)}$ is planar if and only if $G^{(2)}$ is planar. A planar embedding of $G^{(2)}$ can be obtained from a planar embedding of $G^{(3)}$ by locally replacing the embeddings of graphs $C^\circ_{\mathcal{F}_1},\ldots,C^\circ_{\mathcal{F}_q}$ with consistent embeddings of graphs $\mathcal{F}_1^\circ,\ldots,\mathcal{F}_q^\circ$.*

5.2.2 The Constraint Graph of a Core

For the core $\mathcal{C}$ of a graph H_P, we need to capture the different embeddings of its two virtual edges (a,b,i) and (c,d,j). If $\{a,b\}=\{c,d\}$, $\mathcal{C}$ consists of a single bond, and we define $C_{\mathcal{C}}=\mathcal{C}$. So assume that $\{a,b\}\neq\{c,d\}$. Then there are three possibilities:

(a) There exists an embedding of $\mathcal{C}$ which has two faces with edges (a,b,i) and (c,d,j) on their boundaries.

(b) There exists an embedding of $\mathcal{C}$ which has one face with edges (a,b,i) and (c,d,j) on its boundary.

(c) There exists no embedding of $\mathcal{C}$ such that edges (a,b,i) and (c,d,j) appear on the same face.

Figure 5.4 shows cores $\mathcal{C}$ illustrating these three possibilities and the constraint graphs we construct in each of these cases. It is easy to verify that if core $\mathcal{C}$ satisfies one of the above conditions, then $C_{\mathcal{C}}$ satisfies the same condition, and vice versa. Moreover, these are the only possibilities for the structure of core $\mathcal{C}$, as there cannot be three faces with edges (a,b,j) and (c,d,k) on their boundaries. This can be shown as follows: Assume that there exists an embedding such that three faces have edges (a,b,j) and (c,d,k) on their boundaries. Each such face needs to have all endpoints of edges (a,b,i) and (c,d,j) on its boundary. However, since $|\{a,b,c,d\}|\geq 3$, this implies that the face-on-vertex graph G_F of such an embedding of $\mathcal{C}$ contains $K_{3,3}$ as a subgraph. This contradicts the planarity of G_F.

[1]Here, two vertices are consecutive if there exists no *required* vertex between them on the boundary of the face. There may be other vertices between them.

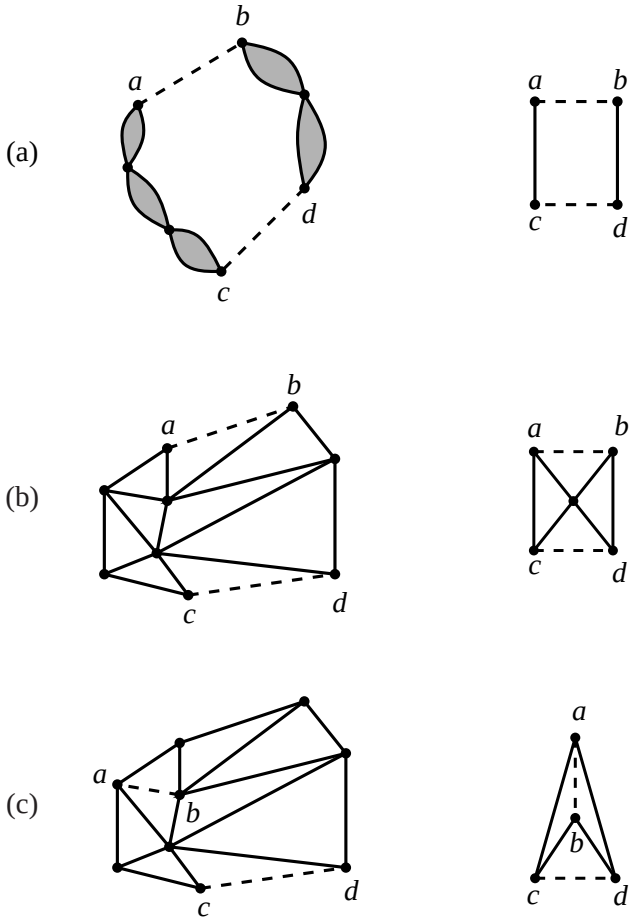


Figure 5.4
Three examples of cores illustrating the different constellations of virtual edges (a,b,j) and (c,d,k). The left graph in each figure is a core C. The right graph is its constraint graph C_C.

We test for these three possibilities in a way similar to the processing of a fan. In particular, we split edge (a,b,i) into two edges $\{a,x\}$ and $\{x,b\}$ and edge (c,d,j) into two edges $\{c,y\}$ and $\{y,d\}$. Then we add two vertices z_1 and z_2 to $\mathcal{C}$ and connect both of them to vertices a, b, c, d, x, and y. The first case applies if and only if the resulting graph is planar. To test for the second case, we remove z_2 and its incident edges from the graph and test for planarity again. If both tests fail, the third case applies.

Let $\mathcal{C}_1,\ldots,\mathcal{C}_q$ be the cores of all graphs H_P in $G^{(3)}$. Then we define a sequence of graphs $G_0^{(4)},\ldots,G_q^{(4)}$, where $G_0^{(4)} = G^{(3)}$ and $G_i^{(4)} = G_{i-1}^{(4)}[\mathcal{C}_i^\circ / C_{\mathcal{C}_i}^\circ]$, for $1 \le i \le q$. Graph $G^{(4)}$ is defined as $G^{(4)} = G_q^{(4)}$.

Lemma 5.3 *Graph $G_i^{(4)}$ is planar if and only if graph $G_{i-1}^{(4)}$ is planar, for $1 \le i \le q$. A planar embedding of $G_{i-1}^{(4)}$ can be obtained from a planar embedding of $G_i^{(4)}$ by replacing the embedding of $C_{\mathcal{C}_i}^\circ$ induced by the embedding of $G_i^{(4)}$ with a consistent embedding of $\mathcal{C}_i^\circ$.*

Proof. Let $\hat{G}_{i-1}^{(4)}$ be a planar embedding of $G_{i-1}^{(4)}$, and let $\hat{\mathcal{C}}_i^\circ$ be the embedding of $\mathcal{C}_i^\circ$ induced by $\hat{G}_{i-1}^{(4)}$. By the construction of core $\mathcal{C}_i$, none of the vertices in $\mathcal{C}_i$ is required in the bicomp $\mathcal{B}$ containing $\mathcal{C}_i$. Hence, no vertex in $\bar{\mathcal{B}}$ can be adjacent to any vertex in $\mathcal{C}_i$. Splits $s(a,b,j)$ and $s(c,d,k)$ partition $\mathcal{B}$ into three graphs $\mathcal{C}_i$, K_1, and K_2; graph K_1 shares virtual edge (a,b,j) with $\mathcal{C}_i$; graph K_2 shares virtual edge (c,d,k) with $\mathcal{C}_i$. As a result, graph K_1° shares vertices a and b with $\mathcal{C}_i^\circ$, and graph K_2° shares vertices c and d with $\mathcal{C}_i^\circ$. Thus, K_1° must be embedded in a face of $\hat{\mathcal{C}}_i^\circ$ which has vertices a and b on its boundary; K_2° must be embedded in a face of $\hat{\mathcal{C}}_i^\circ$ which has vertices c and d on its boundary. Since no vertex in $\bar{\mathcal{B}}$ is adjacent to a vertex in $\mathcal{C}_i$, all components of $\bar{\mathcal{C}}_i^\circ$ are embedded inside one of the faces of $\hat{\mathcal{C}}_i^\circ$ containing K_1° and K_2°.

The face of $\hat{\mathcal{C}}_i^\circ$ containing K_1° is the one obtained by removing edge (a,b,j) from $\hat{\mathcal{C}}_i$. The face containing K_2° is the one obtained by removing edge (c,d,k) from $\hat{\mathcal{C}}_i$. It is easy to verify that in all three cases shown in Figure 5.4, the order of vertices a,b,c,d around these faces is preserved. Hence, any subgraph of $\bar{\mathcal{C}}_i^\circ$ embedded in such a face of $\hat{\mathcal{C}}_i^\circ$ can be embedded inside the corresponding face of $\hat{\mathcal{C}}_{\mathcal{C}_i}^\circ$. As the constellations shown in Figure 5.4 are the only possibilities, as argued above, this shows that a planar embedding of $G_i^{(4)}$ can always be derived from a planar embedding of $G_{i-1}^{(4)}$. In order to show that $G_{i-1}^{(4)}$ is planar if $G_i^{(4)}$ is planar, we reverse the above argument. □

Corollary 5.3 *Graph $G^{(4)}$ is planar if and only if graph $G^{(3)}$ is planar. A planar embedding of $G^{(3)}$ can be obtained from a planar embedding of $G^{(4)}$ by replacing the embeddings of graphs $C_{\mathcal{C}_1}^\circ,\ldots,C_{\mathcal{C}_q}^\circ$ with consistent embeddings of graphs $\mathcal{C}_1^\circ,\ldots,\mathcal{C}_q^\circ$.*

5.3 The Constraint Graph of the Bicomp

The above construction replaces every bicomp $\mathcal{B}$ in G with a multigraph $C'_{\mathcal{B}}$. In order to finish the construction, we remove all multiple edges from $G^{(4)}$. Let $G^{(5)}$ be the resulting graph. The construction of $G^{(5)}$ is equivalent to the following two-step procedure: First replace every bicomp $\mathcal{B}$ with a graph $C_{\mathcal{B}}$, which is obtained from $C'_{\mathcal{B}}$ by removing multiple edges. Then remove remaining multiple edges from the union of graphs $C_{\mathcal{B}_1},\ldots,C_{\mathcal{B}_q}$, where $\mathcal{B}_1,\ldots,\mathcal{B}_q$ are the bicomps of graphs $G'_1,\ldots,G'_l$. Graph $C_{\mathcal{B}}$ is the constraint graph of bicomp $\mathcal{B}$. The following lemma is obvious.

Lemma 5.4 *Graph $G^{(5)}$ is planar if and only if $G^{(4)}$ is planar. A planar embedding of $G^{(4)}$ can be obtained from a planar embedding $\hat{G}^{(5)}$ of $G^{(5)}$ by duplicating edges in $\hat{G}^{(5)}$.*

Next we show that the constraint graph $C_{\mathcal{B}}$ of a bicomp $\mathcal{B}$ is small.

Lemma 5.5 *The constraint graph $C_{\mathcal{B}}$ of a bicomp $\mathcal{B}$ is a simple planar graph with $O(|R(\mathcal{B})|)$ vertices.*

Proof. The planarity of $C_{\mathcal{B}}$ follows immediately from the above construction. We show that there are at most $2|R(\mathcal{B})|$ essential tricomps in $\mathcal{B}$. There are two types of essential tricomps. Type-I tricomps are tricomps $\mathcal{T}(v)$, $v \in R(\mathcal{B})$. Type-II tricomps are tricomps whose corresponding vertices in T' have degree at least three, where T' is the tree constructed from the tricomp tree $T = T_3(B)$ in Section 5.1. Clearly, there are at most $|R(\mathcal{B})|$ tricomps of type I. Let T'' be the tree obtained from T' by replacing every maximal path whose internal vertices correspond to inessential tricomps with a single edge. Tree T'' contains all vertices of T' corresponding to essential tricomps. All leaves of T'' correspond to type-I tricomps, so that there are at most $|R(\mathcal{B})|$ leaves in T''. The vertices corresponding to type-II tricomps are a subset of the vertices of degree at least three in T''. There can be at most $|R(\mathcal{B})| - 1$ such vertices, as there are at most $|R(\mathcal{B})|$ leaves in T''. Thus, there are at most $2|R(\mathcal{B})| - 1$ essential tricomps in $\mathcal{B}$.

Every edge in T'' represents a (possibly empty) path of vertices of degree two in T', which correspond to inessential tricomps. For each such path, the graph H_P obtained by merging the tricomps corresponding to the vertices in P has been replaced by a constraint graph C_P of constant size. This implies that the total size of all constraint graphs not corresponding to essential tricomps is $O(|R(\mathcal{B})|)$. The constraint graph C_P corresponding to an edge in T'' shares at most four vertices with the tricomps corresponding to the endpoints of the edge. Thus, the total number of required vertices in all essential tricomps is $O(|R(\mathcal{B})|)$, which implies that the constraint graphs of these tricomps have total size $O(|R(\mathcal{B})|)$, by Lemma 4.2. As merging all constraint graphs can only reduce the number of vertices in the resulting graph, graph $C'_{\mathcal{B}}$ has $O(|R(\mathcal{B})|)$ vertices. Graph $C_{\mathcal{B}}$ is obtained from $C'_{\mathcal{B}}$ by removing edges. □

6 The Constraint Graph of a Connected Component

So far we have replaced every bicomp $\mathcal{B}$ of graphs $G'_1, \ldots, G'_l$ by a small constraint graph $C_{\mathcal{B}}$. In order to obtain constraint graphs $C_1, \ldots, C_l$, we remove some inessential bicomps altogether and replace chains of inessential bicomps by constraint graphs of constant size. The construction is similar to the construction of the constraint graph $C_{\mathcal{B}}$ of a bicomp $\mathcal{B}$ from the constraint graphs of its tricomps. That is, we first classify the bicomps of a graph G'_j as essential, potentially essential, or inessential, and remove all inessential bicomps. Then we finish the classification of potentially essential bicomps based on the degree of their corresponding vertices in the bicomp-cutpoint-tree of G'_j. The remaining inessential bicomps form chains in G'_j sharing only two vertices with the rest of G'_j. We replace each such chain with a constraint graph of constant size. Next we describe this construction in detail.

Let G'_j be one of the graphs $G'_1, \ldots, G'_l$, let S_j be the set of separator vertices in G'_j, and let $\mathcal{B}_1, \ldots, \mathcal{B}_q$ be the bicomps of G'_j. The *bicomp-cutpoint-tree* $T = T_2(G'_j)$ is defined as follows: Tree T contains all cutpoints $v_1, \ldots, v_r$ of G'_j and one bicomp vertex β_i, for every bicomp $\mathcal{B}_i$ of G'_j. There is an edge $\{v_k, \beta_i\}$ in T if cutpoint v_k is contained in bicomp $\mathcal{B}_i$. For every vertex $v \in S_j$, we choose a bicomp $\mathcal{B}(v)$ such that $v \in \mathcal{B}(v)$. As T contains bicomp nodes as well as cutpoints, we classify the nodes of T, rather than the bicomps of G'_j, as essential, potentially essential, or inessential. A node β_i in T is *essential* if there exists a vertex $v \in S_j$ such that $\mathcal{B}(v) = \mathcal{B}_i$. A node v is *potentially essential* if there are two essential nodes u and w in T such that v is on the path from u to w in T. All other nodes of T are *inessential*.

In the next section, we show that removing all bicomps corresponding to inessential nodes in T from $G^{(5)}$ preserves the (non-)planarity of $G^{(5)}$. Then we classify the potentially essential nodes as either essential or inessential. In Section 6.2, we replace every maximal chain of bicomps corresponding to inessential nodes with a constraint graph of constant size, and show that this preserves the (non-)planarity of the graph. In Section 6.3, we show that the resulting constraint graph C_j of G'_j has size $O(|S_j|)$.

6.1 Discarding Inessential Bicomps

Let C'_j be the graph obtained from G'_j by replacing every bicomp $\mathcal{B}$ of G'_j with its constraint graph $C_{\mathcal{B}}$. Let T' be the tree obtained by removing all inessential nodes from T, and let C''_j be the subgraph of C'_j obtained by removing all constraint graphs $C_{\mathcal{B}_i}$ from C'_j which correspond to bicomp nodes β_i that were removed from T.

We define a sequence $G^{(6)}_0, \ldots, G^{(6)}_l$ of graphs, where $G^{(6)}_0 = G^{(5)}$ and $G^{(6)}_i = G^{(6)}_{i-1}[C'_i/C''_i]$, for $1 \leq i \leq l$. Graph $G^{(6)}$ is defined as $G^{(6)} = G^{(6)}_l$.

Lemma 6.1 *Graph $G^{(6)}_i$ is planar if and only if graph $G^{(6)}_{i-1}$ is planar, for $1 \leq j \leq l$. A planar embedding of $G^{(6)}_{i-1}$ can be obtained from a planar embedding $\hat{G}^{(6)}_i$ by locally replacing the embedding $\hat{C}''_j$ of C''_j induced by $\hat{G}^{(6)}_i$ with a consistent planar embedding of C'_j.*

Proof. Graph $G^{(6)}_i$ is obtained from $G^{(6)}_{i-1}$ by removing vertices and edges from $G^{(6)}_{i-1}$. Thus, if $G^{(6)}_{i-1}$ is planar, $G^{(6)}_i$ is obviously planar.

To show that $G^{(6)}_{i-1}$ is planar if $G^{(6)}_i$ is planar, we partition the graph $C'_j - C''_j$ into its connected components. Each such component K is composed of inessential bicomps of C'_j and shares only one cutpoint v with C'_j. Hence, graph K shares only vertex v with $\bar{K}$ and can be embedded inside any face of $G^{(6)}_i$ which has vertex v on its boundary. As this is true for all components if $C'_j - C''_j$, graph $G^{(6)}_{i-1}$ is planar if graph $G^{(6)}_i$ is planar. □

Corollary 6.1 *Graph $G^{(6)}$ is planar if and only if $G^{(5)}$ is planar. A planar embedding of $G^{(5)}$ can be obtained from a planar embedding $\hat{G}^{(6)}$ of $G^{(6)}$ by replacing the embeddings of graphs $C''_1, \ldots, C''_l$ in $\hat{G}^{(6)}$ with consistent embeddings of graphs $C'_1, \ldots, C'_l$.*

Having disposed of the first set of bicomps corresponding to inessential nodes in T, we now classify the potentially essential nodes of T' as either essential or inessential. A potentially essential node is essential if it has degree at least three in T'; otherwise, it is inessential. Note that all inessential nodes have degree two, since all leaves of T' are essential, and all internal nodes of degree at least three are essential. Thus, we can partition the set of inessential nodes in T' into maximal paths. For each such path P containing at least one bicomp node β_i, let $H_P = \mathcal{B}_{i_1} \cup \cdots \cup \mathcal{B}_{i_q}$, where $\beta_{i_1}, \ldots, \beta_{i_q}$ are the bicomp nodes in P. Next we replace each such graph H_P by a constraint graph C_P of constant size.

6.2 Compressing Chains of Inessential Bicomps

Let H_P be a graph corresponding to a path P of inessential nodes in T'. As all bicomps in H_P are inessential, and every node in P has degree two, graph H_P shares exactly two vertices a and b with $\bar{H}_P$. If H_P has an embedding such that vertices a and b are on the boundary of the same face, graph C_P consists of the single edge $\{a,b\}$. Otherwise, we replace H_P by the graph C_P shown in Figure 6.1. Graph C_P is triconnected and has the property that vertices a and b are not on the boundary of the same face in the unique embedding $\hat{C}_P$ of C_P. The test which of the two cases applies can be carried out in linear time: If the graph $(V(H_P), E(H_P) \cup \{\{a,b\}\})$ is planar, there exists a planar embedding of H_P such that vertices a and b appear on the same face. Otherwise, there exists no such embedding.

Let $T'_1, \ldots, T'_l$ be the trees obtained from trees $T_2(G'_1), \ldots, T_2(G'_l)$ using the construction in Section 6.1. Let $P_1, \ldots, P_q$ be the maximal paths of inessential vertices in trees $T'_1, \ldots, T'_l$. We define a sequence of graphs $G^{(7)}_0, \ldots, G^{(7)}_q$ as follows: $G^{(7)}_0 = G^{(6)}$. For $1 \leq i \leq q$, $G^{(7)}_i = G^{(7)}_{i-1}[H_{P_i}/C_{P_i}]$. The approximate graph A of G is defined as $A = G^{(7)}_q$.

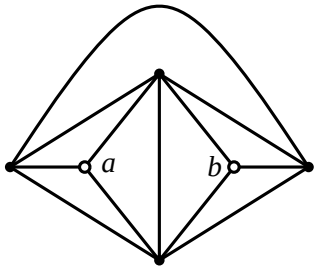


Figure 6.1
The constraint graph of a "twisted" chain of bicomps.

Lemma 6.2 *Graph $G_i^{(7)}$ is planar if and only if graph $G_{i-1}^{(7)}$ is planar, for $1 \leq i \leq q$. A planar embedding of $G_{i-1}^{(7)}$ can be obtained from a planar embedding of $G_i^{(7)}$ by locally replacing the embedding $\hat{C}_{P_i}$ of graph C_{P_i} with a consistent embedding of graph H_{P_i}.*

Proof. First assume that $G_{i-1}^{(7)}$ is planar. Let $\hat{G}_{i-1}^{(7)}$ be a planar embedding of $G_{i-1}^{(7)}$, and let $\hat{H}_{P_i}$ be the planar embedding of H_{P_i} induced by $\hat{G}_{i-1}^{(7)}$. We partition $\bar{H}_{P_i}$ into maximal subgraphs such that each of these subgraphs is embedded in a different face of $\hat{H}_{P_i}$. As all bicomps in H_{P_i} are inessential, such a subgraph can contain only cutpoints a_i and b_i. If there are subgraphs of $\bar{H}_{P_i}$ containing both a_i and b_i, $\hat{H}_{P_i}$ has a face with both vertices a_i and b_i on its boundary. Hence, C_{P_i} consists of edge $\{a_i, b_i\}$, and all subgraphs of $\bar{H}_{P_i}$ containing a_i and b_i can be embedded without intersections in the only face of $\hat{C}_{P_i}$. Subgraphs of $\bar{H}_{P_i}$ containing only one of vertices a_i and b_i can be embedded inside any face of $\hat{C}_{P_i}$ which has the respective vertex on its boundary. Hence, $G_i^{(7)}$ is planar if $G_{i-1}^{(7)}$ is planar.

Now assume that $G_i^{(7)}$ is planar. Let $\hat{G}_i^{(7)}$ be a planar embedding of $G_i^{(7)}$. If C_{P_i} consists of a single edge, then H_{P_i} has an embedding $\hat{H}_P$ with vertices a_i and b_i on the same face. A simple transformation guarantees that the outer face of $\hat{H}_{P_i}$ has vertices a_i and b_i on its boundary. Then we replace edge $\{a_i, b_i\}$ in $\hat{G}_i^{(7)}$ with embedding $\hat{H}_{P_i}$. This produces a planar embedding of $G_{i-1}^{(7)}$, as a_i and b_i are the only vertices shared by H_{P_i} and $\bar{H}_{P_i}$.

If C_{P_i} is the graph shown in Figure 6.1, every component of $\bar{H}_{P_i}$ embedded inside a face of $\hat{C}_{P_i}$ contains only one of a_i and b_i. Hence, it can be embedded inside any face of an embedding $\hat{H}_{P_i}$ of H_{P_i} which has the respective vertex on its boundary. Thus, graph $G_{i-1}^{(7)}$ is planar if graph $G_i^{(7)}$ is planar. □

Corollary 6.2 *Graph A is planar if and only if graph $G^{(6)}$ is planar. A planar embedding of $G^{(6)}$ can be obtained from a planar embedding $\hat{A}$ of A by locally replacing the embeddings $\hat{C}_{P_1}, \ldots, \hat{C}_{P_q}$ of graphs $C_{P_1}, \ldots, C_{P_q}$ with consistent embeddings of graphs $H_{P_1}, \ldots, H_{P_q}$.*

6.3 The Constraint Graph of the Component

The construction of the previous sections replaces each graph G'_j with its constraint graph C_j. In this section we show that C_j is small.

Lemma 6.3 *The constraint graph C_j of graph G'_j is a simple planar graph with $O(|S_j|)$ vertices, for $1 \leq j \leq l$.*

Proof. The planarity of C_j follows immediately from the above construction. In order to show that C_j has $O(|S_j|)$ vertices, let $T = T_2(G'_j)$ be the bicomp-cutpoint-tree of G'_j. Let T' be the tree constructed from T in Section 6.1. We partition the essential nodes of T' into two classes: Type-I nodes are bicomp nodes β_i such that $\mathcal{B}_i = \mathcal{B}(v)$, for some vertex $v \in S_j$. Type-II nodes have degree at least three in T'. There are at most $|S_j|$ type-I nodes. As in the proof of Lemma 5.5, the number of type-II nodes can be bounded by the number of leaves in T', which is $|S_j|$ because all leaves of T' are of type I. Thus, there are at most $2|S_j|$ essential nodes in T'.

Let T'' be the tree obtained from T' by replacing every maximal path whose internal nodes are inessential by a single edge. Then the nodes of T'' are the essential nodes in T', so that T'' has at most $2|S_j|$ nodes and at most $2|S_j|-1$ edges. Every edge in T'' corresponds to a path P in T', which in turn corresponds to a possibly empty graph H_P in C'_j. Each such graph H_P is being replaced by a constraint graph C_P of constant size. Every essential bicomp in C'_j contains at most as many cutpoints as edges incident to its corresponding node in T''. Thus, the total number of required vertices in all essential bicomps is at most $5|S_j|$. By Lemma 5.5, each such bicomp is represented by a planar constraint graph whose size is linear in the number of its required vertices. Thus, C_j has $O(|S_j|)$ vertices. □

7 The Approximate Graph

The approximate graph A of G is the graph obtained after applying the replacement procedures of Sections 4 through 6 to G. The following two lemmas show that graph A has the desired properties.

Lemma 7.1 *Graph A is planar if and only if G is planar.*

Proof. This follows immediately from Corollaries 4.1, 5.1, 5.2 and 5.3, Lemma 5.4, and Corollaries 6.1 and 6.2. □

Lemma 7.2 *Graph A has size $O(N/(DB))$.*

Proof. In Step 3 of Algorithm 2.1, graph G is partitioned into $O(N/(DB)^2)$ graphs $G_1,\ldots,G_k$ such that $|\partial G_i| \le DB$, for $1 \le i \le k$. In particular, $\sum_{i=1}^{k} |\partial G_i| = O(N/(DB))$. As graphs $G'_1,\ldots,G'_l$ are the connected components of graphs $\tilde{G}_1,\ldots,\tilde{G}_k$, this implies that $\sum_{j=1}^{l} |S_j| = \sum_{i=1}^{k} |\partial G_i| = O(N/(DB))$. By Lemma 6.3, $|C_j| = O(|S_j|)$, for $1 \le j \le l$, so that $\sum_{j=1}^{l} |C_j| = O(\sum_{j=1}^{l} |S_j|) = O(N/(DB))$. Graph A consists of graphs $C_1,\ldots,C_l$ and the subgraph $G[S]$ of G induced by the separator vertices in S. Graph $G[S]$ has $O(N/(DB))$ vertices. Thus, $|A| \le \sum_{j=1}^{l} |C_j| + |G[S]| = O(N/(DB))$. □

8 Constructing the Final Embedding

Given a planar embedding of A, Lemma 7.1 implies that G is planar. In this section we describe how to derive a planar embedding $\hat{G}$ of G from an embedding $\hat{A}$ of the approximate graph A. Conceptually, the construction is fairly simple: Replace the embeddings of graphs $C_1,\ldots,C_l$ induced by $\hat{A}$ one by one with consistent embeddings of graphs $G'_1,\ldots,G'_l$. This intuitively simple process presents a few technicalities that have to be dealt with, in order to obtain a valid embedding of G.

Section 8.2 defines formally what we mean by an embedding of G'_j which is "consistent" with an embedding of C_j, and shows how to derive such an embedding. The "replacement" of embedding $\hat{C}_j$ with the computed embedding $\hat{G}'_j$ is done as follows: First partition $\hat{C}_j$ into maximal subgraphs such that each of them is embedded inside a different face of $\hat{C}_j$. Then place each such subgraph inside an appropriate face of $\hat{G}'_j$, without changing its embedding.

An important constraint on the replacement of graphs $C_1,\ldots,C_l$ with graphs $G'_1,\ldots,G'_l$ is given by the fact that it would be computationally too expensive to extract the embedding of graph C_j immediately before the replacement of C_j with G'_j. Thus, we extract *all* embeddings $\hat{C}_1,\ldots,\hat{C}_l$ of graphs $C_1,\ldots,C_l$ induced by $\hat{A}$ before starting to replace them with graphs $G'_1,\ldots,G'_l$. The construction of embedding $\hat{G}'_j$ depends on the embedding $\hat{C}_j$ of C_j. Thus, we have to ensure that replacing embeddings $\hat{C}_1,\ldots,\hat{C}_{j-1}$ with embeddings $\hat{G}'_1,\ldots,\hat{G}'_{j-1}$ in $\hat{A}$ does not change the embedding $\hat{C}_j$ of C_j induced by $\hat{A}$; otherwise, the extracted embedding would be invalid at the time when C_j is replaced with G'_j. The construction in Section 8.2 takes this into account. Before we describe the construction of $\hat{G}$ from $\hat{A}$ in detail, we explain how the planar embedding $\hat{A}$ of A and the final embedding $\hat{G}$ of G are to be represented, and how to extract graphs $C_1,\ldots,C_l$ and their embeddings $\hat{C}_1,\ldots,\hat{C}_l$ I/O-efficiently.

8.1 Extracting the Embeddings of Constraint Graphs

Given the partition of G into graphs $G'_1,\ldots,G'_j$ and $G[S]$, every edge in G belongs to exactly one of these graphs. We label every edge in G with the name of the subgraph containing it. Similarly, every edge in A belongs to one of the graphs $C_1,\ldots,C_l$ and $G[S]$. Edges in $G[S]$ are in both G and A. The edges in $C_1,\ldots,C_l$ can easily be labelled as belonging to one of these graphs while constructing graphs $C_1,\ldots,C_l$ from graphs $G'_1,\ldots,G'_l$.

We represent the embedding $\hat{A}$ of A as a collection of *interlaced edge cycles*. That is, every edge $e=\{v,w\}\in A$ stores four pointers: two pointers $\mathrm{succ}^v_A(e)$ and $\mathrm{pred}^v_A(e)$ to its two neighbors clockwise and counterclockwise around v, respectively, and two pointers $\mathrm{succ}^w_A(e)$ and $\mathrm{pred}^w_A(e)$ to its two neighbors around w. Our goal is to modify these pointers to obtain interlaced edge cycles representing a valid planar embedding $\hat{G}$ of G. In order to be able to extract the embedding of only a subset of the edges incident to any vertex in A, it is convenient to have the edges incident to each vertex v numbered clockwise around v. That is, in addition to pointers $\mathrm{pred}^v_A(e)$, $\mathrm{succ}^v_A(e)$, $\mathrm{pred}^w_A(e)$, and $\mathrm{succ}^w_A(e)$, every edge $e=\{v,w\}$ in A should store two labels $\nu_v(e)$ and $\nu_w(e)$ representing the numbers of edge e in the clockwise orders of the edges around vertices v and w, respectively. Such a labelling of the edges can be derived from the interlaced edge cycles representing $\hat{A}$ as follows:

Represent each edge $e=\{v,w\}$ in A by two triples $(v,e,\mathrm{succ}^v_A(e))$ and $(w,e,\mathrm{succ}^w_A(e))$. Sort the resulting set of triples by their first components. The result is a concatenation of lists, each representing a circular linked list of edges clockwise around a vertex of A. Replacing one of the triples $(v,e,\mathrm{succ}^v_A(e))$ in each list by the triple $(v,e,\textbf{null})$, we obtain a collection of regular linked lists. Now apply the list-ranking procedure of [9] to all of these lists simultaneously. The result is an assignment of a label $\nu_v(e)$ to each triple $(v,e,\cdot)$, where $\nu_v(e)$ is the number of edge e in the order of edges clockwise around vertex v. Sorting these triples by their second components and scanning the resulting list, we can now compute triples $(e,\nu_v(e),\nu_w(e))$, for all edges $e=\{v,w\}$ in A. This procedure takes $O(\mathrm{sort}(N/(DB)))$ I/Os.

Graphs $C_1,\ldots,C_l$ and their embeddings $\hat{C}_1,\ldots,\hat{C}_l$ are now easily extracted: In order to extract graphs $C_1,\ldots,C_l$, sort the edges in A by their component labels. This produces a partition of $E(A)$ into sets $E(G[S]),E(C_1),\ldots,E(C_l)$. Given graph C_j, a representation of its embedding as interlaced edge cycles can be extracted by reversing the construction of the previous paragraph. In particular, we create two triples $(v,\nu_v(e),e)$ and $(w,\nu_w(e),e)$, for each edge $e=\{v,w\}$ in C_j, and sort the resulting list lexicographically. As a result, all edges incident to a vertex $v\in C_j$ are stored consecutively, sorted clockwise around v. In a single scan we replace every triple $(v,\nu_v(e),e)$ with a triple $(e,\mathrm{pred}^v_{C_j}(e),\mathrm{succ}^v_{C_j}(e))$. Sorting these triples by their first components and scanning the resulting list, we compute a list of quintuples $(e,\mathrm{pred}^v_{C_j}(e),\mathrm{succ}^v_{C_j}(e),\mathrm{pred}^w_{C_j}(e),\mathrm{succ}^w_{C_j}(e))$, for each edge $e=\{v,w\}$ in C_j. The whole construction takes $O(\mathrm{sort}(N/(DB)))$ I/Os, for all graphs $C_1,\ldots,C_l$.

In order to be able to perform the replacement of $\hat{C}_j$ with an embedding $\hat{G}'_j$ efficiently, we augment

graph C_j with edges $\text{pred}_A^v(e)$, $\text{succ}_A^v(e)$, $\text{pred}_A^w(e)$, and $\text{succ}_A^w(e)$, for each edge $e = \{v,w\} \in C_j$. Let D_j be the graph obtained by augmenting C_j in this manner, and let $\hat{D}_j$ be its planar embedding induced by $\hat{A}$. Observe that for every (copy of a) required vertex v on the boundary of a face f of $\hat{C}_j$, graph D_j contains the first and last edges, e' and e'', in clockwise order around v which are embedded inside f. If K is the subgraph of $\bar{C}_j$ embedded inside f, all edges in K that are incident to v appear between two such edges e' and e'' in the clockwise order around v. As we replace $\hat{C}_j$ with an embedding $\hat{G}'_j$ of G'_j without changing the embedding of K, the edges in $E(D_j) \setminus E(C_j)$ are the only edges in K whose neighbors in the embedding change as a consequence of the replacement, and all edges in G'_j have their neighbors either in G'_j or in $E(D_j) \setminus E(C_j)$. Thus, graph D_j and its embedding $\hat{D}_j$ provide sufficient information to perform the replacement of $\hat{C}_j$ with $\hat{G}'_j$. Moreover, as every edge in C_j has four neighbors in $\hat{A}$, D_j contains at most five times as many edges as C_j, so that D_j fits into internal memory, and the total size of graphs $D_1, \ldots, D_l$ is $O(\sum_{j=1}^{l} |C_j|) = O(N/(DB))$.

8.2 Replacing the Embedding of a Constraint Graph

Given graphs $D_1, \ldots, D_l$ and their embeddings $\hat{D}_1, \ldots, \hat{D}_l$, we now replace the embeddings $\hat{C}_1, \ldots, \hat{C}_l$ of graphs $C_1, \ldots, C_l$ with consistent embeddings of graphs $G'_1, \ldots, G'_l$. We perform this replacement one graph at a time. In this section, we are concerned with deriving the embedding $\hat{G}'_j$ of G'_j from $\hat{C}_j$ and replacing $\hat{C}_j$ with $\hat{G}'_j$. Section 8.3 shows how to exchange information about updates of the interlaced edge cycles resulting from these replacements between graphs $D_1, \ldots, D_l$. This ensures that subsequent replacements can be performed correctly.

Given an embedding $\hat{C}_j$ of C_j, the goal of the construction in this section is to construct an embedding $\hat{G}'_j$ of G'_j so that the subgraphs of $\bar{C}_j$ embedded in the faces of $\hat{C}_j$ can be embedded inside appropriate faces of $\hat{G}'_j$. This goal is achieved by undoing the compression steps of Sections 4 through 6, one by one, and maintaining a planar embedding of the current graph as well as a mapping of the subgraphs of $\bar{C}_j$ to the faces of the embedding. We call the current embedding and the mapping of subgraphs of $\bar{C}_j$ to the faces of the embedding *consistent* with $\hat{C}_j$ if the following invariant holds:

(I1) Let $K_1, \ldots, K_s$ be the maximal subgraphs of $\bar{C}_j$ so that each of them is embedded inside a different face of $\hat{C}_j$. Then each of the graphs $K_1, \ldots, K_s$ is embedded completely inside one face of the current embedding. Let $e_1, \ldots, e_q$ be the edges in $E(K_1) \cup \cdots \cup E(K_s)$ incident to a vertex $v \in C_j$. If edges $e_1, \ldots, e_q$ appear in this order clockwise around v in $\hat{A}$, then edges $e_1, \ldots, e_q$ appear in this order clockwise around v in the current embedding.

This invariant ensures that the embeddings of graphs $C_{j+1}, \ldots, C_l$ are not changed by the replacement of C_j with G'_j. Given that we do not modify the embeddings of graphs $K_1, \ldots, K_s$ when replacing $\hat{C}_j$ with $\hat{G}'_j$, the following invariant is equivalent to Invariant (I1):

(I2) Let $E_1, \ldots, E_s$ be the maximal subsets of $E(D_j) \setminus E(C_j)$ such that the edges in each subset are embedded inside a different face of $\hat{C}_j$. Then the edges in each of these subsets are embedded inside the same face of the current embedding. Let $e_1, \ldots, e_q$ be the edges in $E(D_j) \setminus E(C_j)$ incident to a vertex $v \in C_j$. If edges $e_1, \ldots, e_q$ appear in this order clockwise around v in $\hat{D}_j$, then edges $e_1, \ldots, e_q$ appear in this order clockwise around v in the current embedding.

Next we provide the details of the reconstruction of G'_j from C_j along with an embedding $\hat{G}'_j$ of G'_j; this construction maintains Invariant (I2). Sections 8.2.1 through 8.2.7 describe the construction of $\hat{G}'_j$. Section 8.2.8 discusses the changes to the interlaced edge cycles representing $\hat{A}$ that need to be made, in order to obtain interlaced edge cycles representing the embedding $\hat{A}[C_j/G'_j]$ of graph $A[C_j/G'_j]$ obtained by replacing $\hat{C}_j$ with the constructed embedding $\hat{G}'_j$.

8.2.1 Introducing Parallel Edges

Recall that the construction of the constraint graphs of bicomps removes all parallel edges that may have been introduced by the removal of inessential tricomps. These edges need to be re-introduced, so that they can later be replaced by the (group of) tricomps they represent. For every group of parallel edges with endpoints v and w, we embed the required additional copies of edge $\{v,w\}$ parallel to the only edge $\{v,w\}$ in C_j so that the faces bounded by these edges do not contain any vertices. As this changes neither the order of edges in $E(D_j) \setminus E(C_j)$ around their endpoints nor places edges that were in the same face into different faces, Invariant (I2) is preserved.

8.2.2 Replacing the Embedding of a Triconnected Component

The next step is to replace the embeddings $\hat{C}^\circ_{\mathcal{T}_1}, \ldots, \hat{C}^\circ_{\mathcal{T}_q}$ of the kernels of constraint graphs $C_{\mathcal{T}_1}, \ldots, C_{\mathcal{T}_q}$ of essential or separating tricomps $\mathcal{T}_1, \ldots, \mathcal{T}_q$ in G'_j with embeddings $\hat{\mathcal{T}}^\circ_1, \ldots, \hat{\mathcal{T}}^\circ_q$ of the kernels of these tricomps.

The treatment of tricomp $\mathcal{T}$ depends on its type. If $\mathcal{T}$ is a bond, then $C_\mathcal{T} = \mathcal{T}$, and we choose $\hat{\mathcal{T}}^\circ = \hat{C}^\circ_\mathcal{T}$. If $\mathcal{T}$ is a cycle, every path in $\mathcal{T}$ whose internal vertices are not required has been replaced by a single edge in $C_\mathcal{T}$. Now we reverse this operation, replacing each such edge by its corresponding path. This replacement can easily be done while maintaining Invariant (I2). The process of replacing the embedding $\hat{C}^\circ_\mathcal{T}$ of $C^\circ_\mathcal{T}$ with an embedding $\hat{\mathcal{T}}^\circ$ of $\mathcal{T}^\circ$ is only slightly more complicated in the case when $\mathcal{T}$ is a triconnected simple graph.

First recall that in this case $\mathcal{T}$ and $C_\mathcal{T}$ both have a unique planar embedding. The construction in Section 4 preserves the order of faces with at least two required vertices on their boundaries around the required vertices of $\mathcal{T}^\circ$. Thus, every subgraph of $\bar{C}^\circ_\mathcal{T}$ embedded inside such a face of $\hat{C}^\circ_\mathcal{T}$ will be embedded inside the corresponding face of $\hat{\mathcal{T}}^\circ$. This preserves the order of edges in these graphs incident to a required vertex of $\mathcal{T}^\circ$ clockwise around that vertex. Any graph embedded inside a face of $\hat{C}^\circ_\mathcal{T}$ with only one required vertex v on its boundary shares only vertex v with $C^\circ_\mathcal{T}$. Hence, it can be embedded inside any face of $\hat{\mathcal{T}}^\circ$ which has vertex v on its boundary. This gives us enough freedom to embed these subgraphs in a way that preserves the order of all edges in $\bar{C}^\circ_\mathcal{T}$ around the required vertices in $C^\circ_\mathcal{T}$. Thus, Invariant (I2) is being preserved.

8.2.3 Replacing the Embedding of a Core

The next step is the replacement of the embeddings of the constraint graphs of cores with embeddings of the cores. Let $\mathcal{C}$ be a core. We treat the three different cases depicted in Figure 5.4 separately. In Case (a), two edges in the current embedding need to be replaced by the embeddings of two graphs sharing two vertices each with the rest of the graph. This is easily done in a manner preserving Invariant (I2). In Case (b), the whole graph $\bar{C}_\mathcal{C}$ is embedded in the outer face of $C^\circ_\mathcal{C}$. Thus, we choose an embedding of $\mathcal{C}$ such that virtual edges (a,b,i) and (c,d,j) are on its outer boundary and then embed $\bar{C}_\mathcal{C}$ in the outer face of $\hat{\mathcal{C}}^\circ$. In Case (c), finally, graph $\bar{C}_\mathcal{C}$ consists of two subgraphs, one of which is embedded in the face of $\hat{C}^\circ_\mathcal{C}$ with vertices a and b on its boundary; the other is embedded in the face with vertices c and d on its boundary. Since these two subgraphs do not share any vertices, embedding each of them in the corresponding face of $\hat{\mathcal{C}}$ maintains Invariant (I2). Thus, in all three cases $\hat{C}_\mathcal{C}$ can be replaced with an embedding of $\mathcal{C}$ which preserves Invariant (I2).

8.2.4 Replacing the Embedding of a Fan

Next we replace the embeddings of the constraint graphs of fans with embeddings of the respective fans. As for cores, we distinguish the different possible configurations shown in Figure 5.3. In Cases (a)–(c), edges in the constraint graphs on the right have to be replaced with the corresponding subgraphs on the left. Each of these subgraphs shares only the two endpoints of the corresponding edge with the rest of D_j. Thus, this replacement can easily be done in a way that preserves Invariant (I2). In Cases (d)–(f), it is easily verified that the order of faces with at least two required vertices on their boundaries around the required vertices of the fan is the same in the embedding of the fan as in the embedding of its constraint graph. Thus, all graphs embedded in faces of $\hat{C}_{\mathcal{F}}$ with at least two required vertices on their boundaries can be embedded inside the corresponding faces of $\hat{\mathcal{F}}$. This preserves Invariant (I2). As in the case of a triconnected component, a graph embedded inside a face of $\hat{C}_{\mathcal{F}}$ with at most one required vertex on its boundary can be embedded inside any face of $\hat{\mathcal{F}}$ with that vertex on its boundary. Hence, we have the freedom to arrange these graphs so that Invariant (I2) is not violated.

8.2.5 Introducing Inessential Tricomps

Inessential tricomps that were removed in Section 5.1 were grouped into maximal groups corresponding to complete subtrees in the tricomp tree of the bicomp containing them. The subgraph K obtained by merging the tricomps in one such group shares exactly one virtual edge (a,b,i) with the rest of G, and was consequently replaced by a non-virtual edge (a,b,i). In order to re-introduce this subgraph into the embedding, we have to replace edge (a,b,i) with an embedding of K° that has vertices a and b on the outer face. This preserves Invariant (I2).

8.2.6 Replacing the Embedding of a Chain of Inessential Bicomps

Having dealt with the embeddings of essential and inessential tricomps, we have to re-introduce all inessential bicomps that were removed from G. The first group of inessential bicomps are those that were replaced by constraint graphs of constant size in Section 6.2. The second group are those that were completely removed from G in Section 6.1. The bicomps in the first group form chains such that each chain K shares exactly two vertices, a and b, with the rest of G. Depending on whether a and b can appear on the same face of an embedding of K, K was replaced by a single edge $\{a,b\}$ or by a constraint graph of constant size which does not allow a and b to appear on the boundary of the same face. In the former case, we have to replace edge $\{a,b\}$ with an embedding of K with vertices a and b on the outer face. As before, this preserves Invariant (I2). In the latter case, no subgraph of $\overline{K}$ can contain both a and b. Thus, we can choose any embedding $\hat{K}$ of K and embed the subgraphs of $\overline{K}$ incident to a and b in the faces of $\hat{K}$ incident to vertices a and b in a manner that preserves their order around these vertices.

8.2.7 Introducing Inessential Bicomps

Finally, all bicomps that were completely removed from G'_j were grouped into maximal connected subgraphs such that each such subgraph shares one vertex with the rest of G. Each such subgraph K sharing a vertex v with $\overline{K}$ can be embedded inside any face of the current embedding which has vertex v on its boundary, without violating Invariant (I2).

8.2.8 Updating the Interlaced Edge Cycles

After finishing the replacement steps in Sections 8.2.1 through 8.2.7, we obtain an embedding $\hat{G}'_j$ of G'_j and an embedding of the edges in $E(D_j) \setminus E(C_j)$ inside the faces of $\hat{G}'_j$. It remains to integrate this information

with the embedding $\hat{A}$ of A, in order to obtain an embedding $\hat{A}[C_j/G'_j]$ of $A[C_j/G'_j]$. In $\hat{A}[C_j/G'_j]$, every edge e in G'_j has both its neighbors around both its endpoints in $E(G'_j) \cup (E(D_j) \setminus E(C_j))$. Thus, if G' is the graph induced by the edges in $E(G'_j) \cup (E(D_j) \setminus E(C_j))$, and $\hat{G}'$ is the embedding of G' derived in Sections 8.2.1 through 8.2.7, then the pointers to be stored with e in the interlaced edge cycles representing $\hat{A}[C_j/G'_j]$ are the same as the ones stored with e in the interlaced edge cycles representing $\hat{G}'$. Similarly, as our construction ensures that the embeddings of subgraphs of $\overline{C_j}$ embedded inside different faces of $\hat{C}_j$ are not changed, all edges in $E(\overline{C_j}) \setminus E(D_j)$ have the same neighbors in $\hat{A}[C_j/G'_j]$ as in $\hat{A}$. Finally, let $e = \{v,w\}$ be an edge in $E(D_j) \setminus E(C_j)$. We discuss the necessary updates for edge e, using its pointer $\text{succ}^v_A(e)$ to its successor clockwise around v as an example. The other three pointers are updated in a similar fashion.

Observe that since $e \in E(D_j) \setminus E(C_j)$, either $\text{succ}^v_A(e) \in C_j$ or $\text{pred}^v_A(e) \in C_j$, or both. Let K be the subgraph of $\overline{C_j}$ containing edge e and let $A' = A[C_j/G'_j]$. If $\text{succ}^v_A(e) \in C_j$, then $\text{succ}^v_{A'}(e) = \text{succ}^v_{G'}(e)$. Otherwise, $\text{succ}^v_A(e) \in K$. As the embedding of graph K is not modified by the replacement of $\hat{C}_j$ with $\hat{G}'_j$, $\text{succ}^v_{A'}(e) = \text{succ}^v_A(e)$ in this case.

8.3 Iterative Replacement of Subgraphs

In the previous section, we have shown that an embedding $\hat{A}[C_j/G'_j]$ of graph $A[C_j/G'_j]$ can be computed from an embedding $\hat{A}$ of graph A by locally modifying the predecessor and successor pointers of edges in D_j and G'_j. Now we use the procedure described in the previous section to produce a sequence of graphs $A_0,\ldots,A_l$ and embeddings $\hat{A}_0,\ldots,\hat{A}_l$ of these graphs, where $A_0 = A$ and $A_i = A_{i-1}[C_i/G'_i]$, for $1 \le i \le l$. Embedding $\hat{A}_0$ is the embedding $\hat{A}$ of graph $A_0 = A$. The embedding $\hat{A}_i$ of graph A_i is computed from the embedding $\hat{A}_{i-1}$ of graph A_{i-1} by applying the procedure of the previous section, in order to replace $\hat{C}_i$ with a consistent embedding $\hat{G}'_i$ of G'_i. The final graph A_l is graph G, so that $\hat{G} = \hat{A}_l$ is the desired embedding of graph G.

There are two issues that need to be addressed, in order to make this iterative replacement of graphs $C_1,\ldots,C_l$ with graphs $G'_1,\ldots,G'_l$ work: (1) When replacing $\hat{C}_i$ with $\hat{G}'_i$, the neighbors of the edges in C_i clockwise and counterclockwise around their endpoints are not necessarily the same in $\hat{A}$ and $\hat{A}_{i-1}$. Hence, graph D_i must be updated, in order to correctly represent this neighborhood information, before it can be used to construct $\hat{G}'_i$ from $\hat{C}_i$, and derive embedding $\hat{A}_i$ from $\hat{A}_{i-1}$. (2) Let p_v, s_v, p_w, s_w be the four neighbors of an edge e in G'_i in the embedding $\hat{A}_i$ obtained after replacing $\hat{C}_i$ with $\hat{G}'_i$. If one of these neighbors, say p_v, is contained in a graph C_j, $j > i$, then p_v will be replaced by another edge p'_v when graph C_j is replaced with graph G'_j. Thus, immediately after replacing C_i and G'_i, the counterclockwise neighbor of edge e around vertex v in the final embedding $\hat{G}$ is not known. Hence, we need a criterion to decide when the neighborhood relationship between two edges cannot be broken as a result of subsequent replacements, so that we can add pointers between these two edges to the final embedding. We address these two problems next.

8.3.1 Updating the Augmented Constraint Graph

In order to update graph D_i so that it contains the correct neighbors of the edges in C_i in the current embedding $\hat{A}_{i-1}$ of A_{i-1}, we use a priority queue Q to collect information about the necessary updates of D_i; before replacing C_i with G'_i, we retrieve this information from Q and make the necessary changes in D_i.

Recall that every edge in A is labelled as belonging to one of the graphs $C_1,\ldots,C_l$ or $G[S]$. If there is an edge $e = \{v,w\} \in E(D_j) \setminus E(C_j)$ such that $e \in E(C_i)$, for $i > j$, and $\text{pred}^v(e)$ changes as a result of replacing C_j with G'_j, then we put a quintuple $(e, v, \text{“pred”}, j, \text{pred}^v(e))$ into Q and give it priority i. For a successor change the quintuple is of the form $(e, v, \text{“succ”}, j, \text{succ}^v(e))$. When replacing graph C_i with graph G'_i, we retrieve all quintuples with priority i from Q. As we have already replaced graphs $C_1,\ldots,C_{i-1}$ with graphs $G'_1,\ldots,G'_{i-1}$ when this happens, all entries with lower priority have already been retrieved from Q. Hence,

retrieving all entries with priority i from Q amounts to repeated application of delete-min operations until the first entry with priority $i+1$ is retrieved. This entry is then put back into Q.

Next we use the quintuples retrieved from Q to update graph D_i. We sort the list of retrieved quintuples lexicographically, so that all quintuples $(e, v, \text{“pred”}, \cdot, \cdot)$ and all quintuples $(e, v, \text{“succ”}, \cdot, \cdot)$ are stored consecutively, for each edge $e = \{v, w\}$. These quintuples are sorted by their fourth component which can be interpreted as the time when this quintuple was queued. Hence, the quintuple with the largest fourth component is the most recent update of the predecessor of edge e in the clockwise order around vertex v, so that its fifth component represents the correct predecessor of e around v in $\hat{A}_{i-1}$. The same is true for quintuples $(e, v, \text{“succ”}, \cdot, \cdot)$. Hence, we scan the sorted list and discard all quintuples $(e, v, \text{“pred”}, \cdot, \cdot)$ and $(e, v, \text{“succ”}, \cdot, \cdot)$, except the last one, for each pair (e, v). The result is a list L_i containing quintuples $(e, v, \text{“pred”}, \cdot, \text{pred}^v_{A_{i-1}}(e))$ and $(e, v, \text{“succ”}, \cdot, \text{pred}^v_{A_{i-1}}(e))$, for all edges whose neighbors around their endpoints are different in $\hat{A}$ and $\hat{A}_{i-1}$. The size of this list is at most the size of D_i. Hence, graphs G'_i, D_i, and list L_i fit into internal memory. We use list L_i to update graph D_i, and then proceed to the construction of embedding $\hat{G}'_i$ from $\hat{C}_i$.

The total number of I/Os spent on queuing and dequeuing quintuples in Q, as well as sorting the retrieved entries before replacing graph C_i with graph G'_i, is $O(\text{sort}(T))$, for all graphs $G'_1, \ldots, G'_l$, where T is the total number of quintuples produced by changing the neighbors of edges in sets $E(D_i) \setminus E(C_i)$, $1 \leq i \leq l$. This number, however, is proportional to the total number of edges in sets $E(D_i) \setminus E(C_i)$, $1 \leq i \leq l$, as the replacement of graph C_i with graph G'_i can change at most all four neighbors of an edge in $E(D_i) \setminus E(C_i)$. Hence, $T = O(N/(DB))$, and it takes $O(\text{sort}(N/(DB)))$ I/Os to maintain graphs $D_1, \ldots, D_l$ and their embeddings, which provide the required information to replace embeddings $\hat{C}_1, \ldots, \hat{C}_l$ with embeddings $\hat{G}'_1, \ldots, \hat{G}'_l$.

8.3.2 Adding Pointers to the Final Edge Lists

It remains to address the problem of producing the interlaced edge cycles representing the final embedding $\hat{G}$ of G. The problem is that the successor of an edge $e \in G'_i$ is changed after replacing C_i with graph G'_i if this successor is an edge $e' \in C_j$, $j > i$. This change of successor happens when graph C_j is replaced with graph G'_j. Thus, when graph C_i is replaced with graph G'_i, we cannot write the successor of edge e to disk yet. The following simple criterion guarantees that we write the neighbor pointers for an edge e only when they do not change any more: If an edge $e \in G'_i$ has its successor in a graph C_j, $j > i$, this successor will change. Thus, we do not write it to disk yet. If on the other hand, edge e has its successor in a graph G'_j, $j \leq i$, or in $G[S]$, then this neighborhood relation cannot change any more as a result of replacing graphs $C_{i+1}, \ldots, C_l$ with graphs $G'_{i+1}, \ldots, G_l$. This is true because edge e and its successor are embedded inside the same faces of embeddings $\hat{C}_{i+1}, \ldots, \hat{C}_l$, and we do not change the embeddings of the maximal subgraphs embedded inside the faces of embedding $\hat{C}_j$ when replacing C_j with G'_j. Thus, we can write the successor pointer of e and the predecessor pointer of its successor to disk as representing the neighborhood relationship in the final embedding $\hat{G}$ of G.

Once all graphs $C_1, \ldots, C_l$ have been replaced with graphs $G'_1, \ldots, G'_l$, the algorithm has produced a list of $4|E|$ quadruples $(e, v, \text{“pred”}, \text{pred}^v_G(e))$ and $(e, v, \text{“succ”}, \text{succ}^v_G(e))$, four quadruples per edge. We sort these quadruples lexicographically, so that the quadruples representing the four neighbors of an edge e are stored consecutively. Now we scan this sorted list to produce the interlaced edge cycles containing quintuples $(e, \text{pred}^v_G(e), \text{succ}^v_G(e), \text{pred}^w_G(e), \text{succ}^w_G(e))$, which represent the final embedding $\hat{G}$ of G.

We summarize this section in the following lemma, which concludes the proof of Theorem 2.1.

Lemma 8.1 *Given a planar embedding $\hat{A}$ of graph A, it takes $O(\text{sort}(N))$ I/Os and linear space to perform the local replacement of embeddings $\hat{C}_1, \ldots, \hat{C}_l$ with consistent embeddings $\hat{G}'_1, \ldots, \hat{G}'_l$. The result is a planar embedding $\hat{G}$ of graph G.*

9 Lower Bound

In this section, we prove an $\Omega(\text{perm}(N))$ I/O lower bound for computing a planar embedding of a planar graph G. We also describe a simple simulation technique which reduces the I/O-complexity of our algorithm to $O(\text{perm}(N))$, thereby matching the lower bound.

9.1 The Lower Bound

The proof of the lower bound of $\Omega(\text{perm}(N))$ I/Os for computing a planar embedding of a planar graph G uses a reduction from the problem of permuting a list of N items $x_1,\ldots,x_N$. In particular, we show that if a representation of a planar embedding of a planar graph G as interlaced edge cycles can be computed in $o(\text{perm}(N))$ I/Os, then it takes $o(\text{perm}(N))$ I/Os to compute the desired permutation of items $x_1,\ldots,x_N$. This contradicts the lower bound of $\Omega(\text{perm}(N))$ I/Os shown for this problem in [32].

Lemma 9.1 *Given an algorithm $\mathcal{A}$ that computes a planar embedding of a planar graph with N vertices in $O(I(N))$ I/Os, there exists an algorithm $\mathcal{A}'$ that permutes a list of N data items in $O(I(N))$ I/Os.*

Proof. We assume that the input to algorithm $\mathcal{A}'$ is given as follows: Let $x_1,\ldots,x_N$ be N data items to be arranged in the order $x_{\sigma(1)},\ldots,x_{\sigma(N)}$, for some permutation $\sigma:[1,N]\rightarrow[1,N]$. Then algorithm $\mathcal{A}'$ is presented with two lists $L_1=\langle(1,x_1),\ldots,(1,x_N)\rangle$ and $L_2=\langle(\sigma(1),y_1),\ldots,(\sigma(N),y_N)\rangle$. The goal of algorithm $\mathcal{A}'$ is to compute a list $L=\langle(x_{\sigma(1)},y_1),\ldots,(x_{\sigma(N)},y_N)\rangle$. In order to achieve this, algorithm $\mathcal{A}'$ computes a graph G whose planar embedding is unique, and such that list L can be extracted from the interlaced edge cycles representing the embedding $\hat{G}$ of G in $O(I(N))$ I/Os. The construction of graph G takes $O(\text{scan}(N))$ I/Os, so that we can compute list L in $O(I(N))$ I/Os by constructing G, computing interlaced edge cycles representing $\hat{G}$, and extracting L from $\hat{G}$.

The vertex set V of graph G consists of four sets V_0,V_1,V_2,V_3: Set V_0 contains a special central vertex z; set V_1 contains all elements of L_1; set V_2 contains all elements of L_2; and set V_3 contains $2N$ vertices numbered 1 through $2N$. The edge set E of G contains edges $\{v,z\}$, for all $v\in V_1\cup V_2$, edges $\{2i-1,(i,x_i)\}$, for $1\leq i\leq N$, edges $\{2\sigma(i),(\sigma(i),y_i)\}$, for $1\leq i\leq N$, and edges $\{1,2\},\{2,3\},\ldots,\{2N-1,2N\},\{2N,1\}$. Graph G is shown in Figure 9.1.

The vertex set of G can be constructed in $O(\text{scan}(N))$ I/Os: append vertex z and vertices $1,\ldots,2N$ to the concatenation of lists L_1 and L_2. In order to represent the edge set of G, we compute the adjacency lists of all vertices in G. The adjacency list of every vertex $(i,x_i)\in V_1$ contains vertices $2i-1$ and z. These lists can easily be constructed in a single scan over L_1. The adjacency list of every vertex $(\sigma(i),y_i)\in V_2$ contains vertices $2\sigma(i)$ and z. Again it takes a single scan over L_2 to construct these lists. The adjacency list of vertex z is the concatenation of lists L_1 and L_2, which can be produced in $O(\text{scan}(N))$ I/Os. The adjacency list of a vertex $2i-1$, $1\leq i\leq N$, contains vertices $2i-2$, $2i$, and (i,x_i). These lists can be produced in a single scan over list L_1. The adjacency list of a vertex $2\sigma(i)$, $1\leq i\leq N$, contains vertices $2\sigma(i)-1$, $2\sigma(i)+1$, and $(\sigma(i),y_i)$. These lists can be produced in a single scan over list L_2. Thus, graph G can be constructed in $O(\text{scan}(N))$ I/Os. Moreover, the embedding of graph G shown in Figure 9.1 is unique. Hence, after computing $\hat{G}$, $\text{succ}^z(\{z,(i,x_i)\})=\{z,(i,y_{\sigma^{-1}(i)})\}$. We scan the interlaced edge cycles representing $\hat{G}$ and discard all quintuples except those belonging to edges $\{z,v\}$, $v\in V_1\cup V_2$. Each remaining quintuple is transformed into the pair $(x_i,y_{\sigma^{-1}(i)})$. The resulting list is a permutation of the desired list L. In order to compute list L, we reverse the I/O-operations that have been performed to arrange items $y_1,\ldots,y_N$ in the current order. This reversal performs the same number of I/O-operations as the construction of $\hat{G}$, i.e., $O(I(N))$ I/Os. Hence, list L can be extracted from $\hat{G}$ in $O(\text{scan}(N)+I(N))=O(I(N))$ I/Os. □

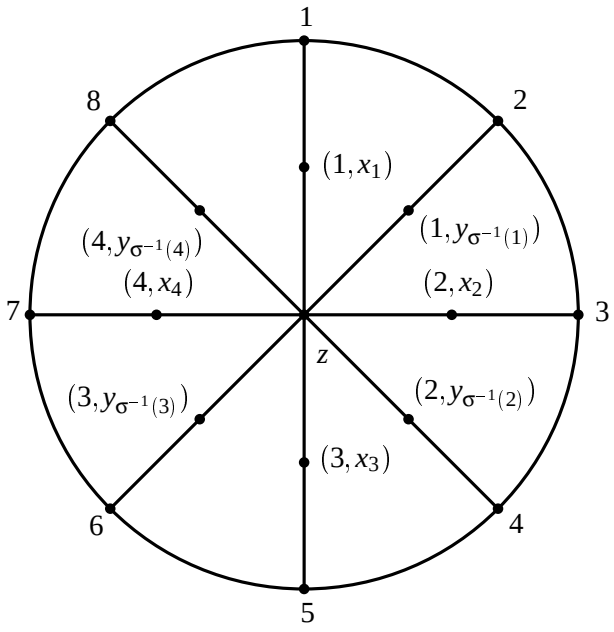


Figure 9.1
The graph G constructed in the proof of Lemma 9.1.

Corollary 9.1 *It takes $\Omega(\text{perm}(N))$ I/Os to compute an embedding of a planar graph with N vertices, if this embedding is to be represented as interlaced edge cycles.*

A fairly straightforward adaptation of the proof of [22] that outerplanar embedding takes $\Omega(\text{perm}(N))$ I/Os shows that it takes $\Omega(\text{perm}(N))$ I/Os to compute a planar embedding, if the embedding is to be represented as a numbering of the edges in G clockwise around each vertex. We believe that a representation of the embedding as interlaced edge cycles is a weaker representation of the embedding than such a numbering, so that Corollary 9.1 is stronger.

9.2 Achieving Optimality

The following lemma provides a simple simulation technique which allows us to improve the I/O-complexity of Algorithm 2.1 so that it matches the lower bound proved in the previous section.

Lemma 9.2 *Given two algorithms $\mathcal{A}$ and $\mathcal{A}'$ solving a problem $\mathcal{P}$ in $I(N)$ and $I'(N)$ I/Os using $S(N)$ and $S'(N)$ space, respectively, there exists an algorithm $\mathcal{A}''$ solving problem $\mathcal{P}$ in $O(\min(I(N), I'(N)))$ I/Os and $O(S(N)+S'(N))$ space, provided that $\max(I(N), I'(N)) = \Omega(\text{scan}(N))$.*

Proof. Given an instance P of problem $\mathcal{P}$, create two identical copies P_1 and P_2 of instance P. This takes $O(\text{scan}(N))$ I/Os. Now run algorithm $\mathcal{A}$ on instance P_1, and simultaneously run algorithm $\mathcal{A}'$ on instance P_2. When algorithm $\mathcal{A}$ cannot proceed without performing an I/O-operation, let algorithm $\mathcal{A}$ perform this I/O-operation and then switch to algorithm $\mathcal{A}'$. When algorithm $\mathcal{A}'$ cannot proceed without performing an I/O-operation, let algorithm $\mathcal{A}'$ perform this I/O-operation and then switch back to algorithm $\mathcal{A}$. Stop this procedure as soon as one of the two algorithms finishes. The I/O-complexity of this procedure is $O(\min(I(N), I'(N)))$. The required space is $O(S(N)+S'(N))$. □

Corollary 9.2 *Given a graph $G = (V, E)$ it takes $O(\text{perm}(|V| + |E|))$ I/Os to test whether G is planar and to compute a planar embedding $\hat{G}$ of G if the answer is affirmative.*

Proof. Let $\mathcal{A}$ be Algorithm 2.1, and $\mathcal{A}'$ be the linear time planarity algorithm of [16]. Algorithm $\mathcal{A}$ takes $O(\text{sort}(N))$ I/Os to compute an embedding $\hat{G}$ of G. Algorithm $\mathcal{A}'$ takes linear time and hence $O(N)$ I/Os to solve this task. Both algorithms use linear space. Hence, by the previous lemma, the simultaneous simulation of both algorithms takes $O(\text{perm}(N))$ I/Os and $O(N/B)$ blocks of external memory to test whether graph G is planar and to compute a planar embedding if the answer is affirmative. □

10 Conclusions

In this paper, we have provided the last missing piece to obtain I/O-efficient algorithms for a number of fundamental problems on planar graphs. As our algorithm uses the separator algorithm of [33], it inherits the constraint that $M \geq (DB)^2 \log^2(DB)$. It is a challenging open problem to design algorithms for computing planar separators and planar embeddings that take $O(\text{sort}(N))$ I/Os using less memory.

References

1. J. Abello, A. L. Buchsbaum, and J. Westbrook. A functional approach to external graph algorithms. In *Proceedings of the 6th European Symposium on Algorithms*, pages 332–343, 1998.

2. P. Agarwal, L. Arge, T. Murali, K. Varadarajan, and J. Vitter. I/O-efficient algorithms for contour-line extraction and planar graph blocking. In *Proceedings of the 10th ACM-SIAM Symposium on Discrete Algorithms*, pages 117–126, 1998.

3. L. Aleksandrov and H. Djidjev. Linear algorithms for partitioning embedded graphs of bounded genus. *SIAM Journal of Discrete Mathematics*, 9:129–150, 1996.

4. L. Arge, G. S. Brodal, and L. Toma. On external memory MST, SSSP, and multi-way planar separators. In *Proceedings of SWAT'2000*, 2000.

5. L. Arge, U. Meyer, L. Toma, and N. Zeh. On external-memory planar depth first search. In *Proceedings of the 7th International Workshop on Algorithms and Data Structures (WADS'2001)*, volume 2125 of *Lecture Notes in Computer Science*, pages 471–482. Springer Verlag, 2001.

6. K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and Systems Science*, 13:335–379, 1976.

7. A. Broder, R. Kumar, F. Manghoul, P. Raghavan, S. Rajagopalan, R. Stata, A. Tomkins, and J. Wiener. Graph structure in the web: Experiments and models. *Computer Networks and ISDN Systems*.

8. A. L. Buchsbaum and J. R. Westbrook. Maintaining hierarchical graph views. In *Proceedings of the Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 566–575, 2000.

9. Y.-J. Chiang, M. T. Goodrich, E. F. Grove, R. Tamassia, D. E. Vengroff, and J. S. Vitter. External-memory graph algorithms. In *Proceedings of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1995.

10. F. Chin, J. Lam, and I. Chen. Efficient parallel algorithms for some graphs problems. *Communications of the ACM*, 25(9):659–665, 1982.

11. A. Crauser, K. Mehlhorn, and U. Meyer. Kürzeste-Wege-Berechnung bei sehr großen Datenmengen. In O. Spaniol, editor, *Promotion tut not: Innovationsmotor "Graduiertenkolleg"*, volume 21 of *Aachener Beiträge zur Informatik*. Verlag der Augustinus Buchhandlung, 1997.

12. S. Even and R. E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.

13. D. Fussell, V. Ramachandran, and R. Thurimella. Finding triconnected components by local replacements. In *Proc. ICALP'89*, volume 372 of *Lecture Notes in Computer Science*, pages 379–393. Springer Verlag, 1989.

14. F. Harary. *Graph Theory*. Addison-Wesley, 1969.

15. J. Hopcroft and R. E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2:135–158, 1973.

16. J. Hopcroft and R. E. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21(4):549–568, 1974.

17. D. Hutchinson, A. Maheshwari, and N. Zeh. An external memory data structure for shortest path queries. In *Proceedings of the 5th ACM-SIAM Computing and Combinatorics Conference*, volume 1627 of *Lecture Notes in Computer Science*, pages 51–60. Springer Verlag, July 1999. To appear in Discrete Applied Mathematics.

18. J. JáJá and J. Simon. Parallel algorithms in graph theory: Planarity testing. *SIAM Journal on Computing*, 11(2):313–328, 1982.

19. P. Klein and J. Reif. An efficient parallel algorithm for planarity. *Journal of Computer and System Sciences*, 37:190–246, 1988.

20. V. Kumar and E. J. Schwabe. Improved algorithms and data structures for solving graph problems in external memory. In *Proceedings of the 8th IEEE Sumposium on Parallel and Distributed Computing*, October 1996.

21. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium (Rome 1966)*, pages 215–232, New York, 1967. Gordon and Breach.

22. A. Maheshwari and N. Zeh. External memory algorithms for outerplanar graphs. In *Proceedings of the 10th International Symposium on Algorithms and Computation*, volume 1741 of *Lecture Notes in Computer Science*, pages 307–316. Springer Verlag, December 1999.

23. A. Maheshwari and N. Zeh. I/O-efficient algorithms for graphs of bounded treewidth. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2001)*, pages 89–90, 2001.

24. K. Mehlhorn and P. Mutzel. On the embedding phase of the hopcroft and tarjan planarity testing algorithm. *Algorithmica*, 16:233–242, 1996.

25. U. Meyer. External memory bfs on undirected graphs with bounded degree. In *Proceedings of the 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA'2001)*, pages 87–88, 2001.

26. G. Miller and J. Reif. Parallel tree contraction and its applications. In *Proceedings of the 26th IEEE Annual Symposium on Foundations of Computer Science*, pages 478–489, 1985.

27. K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *Proceedings of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, January 1999.

28. V. Ramachandran and J. H. Reif. Planarity testing in parallel. *Journal of Computer and System Sciences*, 49(3):517–561, 1994.

29. R. Tarjan. Depth-first search and linear graph algorithms. *SIAM Journal on Computing*, 1:146–159, 1972.

30. R. Tarjan and U. Vishkin. Finding biconnected components and computing tree functions in logarithmic parallel time. *SIAM Journal on Computing*, 14(4):862–874, 1985.

31. U. Ullman and M. Yannakakis. The input/output complexity of transitive closure. *Annals of Mathematics and Artificial Intellligence*, 3:331–360, 1991.

32. J. Vitter and E. Shriver. Algorithms for parallel memory I: Two-level memories. *Algorithmica*, 12(2–3):110–147, 1994.

33. N. Zeh. I/O-efficient planar separators and applications. Technical Report TR-01-02, School of Computer Science, Carleton University, Ottawa, Canada, 2001.