数理解析研究所講究録
1093 巻 1999 年 33-38

Common-face embeddings of planar graphs with applications *

陳 致中 (Zhi-Zhong Chen), Xin He, Ming-Yang Kao
Tokyo Denki University, SUNY at Buffalo, Yale University

1 Introduction

It is a fundamental problem in mathematics to embed a graph into a given space while optimizing certain objectives required by applications. A graph is called *planar* if it can be embedded on the plane so that any pair of edges can only intersect at their endpoints; a *plane* graph is a planar one together with such an embedding. A classical variant of the problem is to test whether a given graph is planar and in case it is, to find a planar embedding. This planarity problem can be solved in linear time.

In this paper, we initiate the study into the following new planarity problem. Let $\mathcal{G}$ be a planar graph. Assume that $\mathcal{G}$ is *simple*. Let $\mathcal{M}$ be a sequence $\mathcal{C}_1, \ldots, \mathcal{C}_q$, where each $\mathcal{C}_i$ is a family of vertex subsets of $\mathcal{G}$. A plane embedding Φ of $\mathcal{G}$ *satisfies* $\mathcal{C}_i$ if the boundary of some face in Φ intersects every set in $\mathcal{C}_i$. Φ *satisfies* $\mathcal{M}$ if it satisfies all $\mathcal{C}_i$. $\mathcal{G}$ *satisfies* $\mathcal{M}$ if $\mathcal{G}$ has an embedding that satisfies $\mathcal{M}$. The CFE problem is the following:

- **Input**: $\mathcal{G}$ and $\mathcal{M}$.
- **Question**: Does $\mathcal{G}$ satisfy $\mathcal{M}$?

We show that the CFE problem is NP-complete. For the special case where every vertex subset in $\mathcal{M}$ induces a connected subgraph of $\mathcal{G}$, we give an $O(\alpha \log \alpha)$-time algorithm, where α is the input size. The CFE problem arises naturally from topological inference [1]. For instance, a less general and efficient version of our algorithm for the special case has been employed to design fast algorithms for reconstructing maps from scrambled partial data in geometric information systems. In this application, each vertex subset in $\mathcal{M}$ describes a recognizable geographical feature, and each family in $\mathcal{M}$ is a set of features that are known to be near each other. Similarly, our algorithm for the special case can compute a constrainted layout of VLSI modules, where each vertex subset consists of the ports of a module, and each subset family specifies a set of modules that are required to be close to each other.

2 The main results

Theorem 2. 1 *The CFE problem is NP-complete.*

The *size* $|\mathcal{C}_i|$ of $\mathcal{C}_i$ is the total cardinality of the sets in $\mathcal{C}_i$. The *size* $|\mathcal{M}|$ of $\mathcal{M}$ is $|\mathcal{C}_1|+\cdots+|\mathcal{C}_q|$. $|G|$ denotes the total number of vertices and edges in a graph G. Let $\alpha = |\mathcal{G}| + |\mathcal{M}|$. The next theorem is the main theorem of this paper.

Theorem 2. 2 *If every vertex subset in $\mathcal{M}$ induces a connected subgraph of $\mathcal{G}$, then the CFE problem can be solved in $O(\alpha \log \alpha)$ time.*

Proof: We consider three special cases:
Case M1: $\mathcal{G}$ is connected.
Case M2: $\mathcal{G}$ is biconnected.
Case M3: $\mathcal{G}$ is triconnected.
Theorem 3. 8 solves Case M3. Theorem 4. 1 reduces this theorem to Case M1. §5 reduces Case M1 to M2. Theorem 6. 1 uses Theorem 3. 8 to solve Case M2. □

The remainder of the paper assumes that every vertex subset of $\mathcal{G}$ in $\mathcal{M}$ induces a connected subgraph of $\mathcal{G}$.

3 Case M3

This section assumes that $\mathcal{G}$ is triconnected. Then, $\mathcal{G}$ has a unique combinatorial embedding up to the choice of the exterior face. Thus, the CFE problem reduces in linear time to that of finding all the faces in the embedding whose boundaries intersect every set in some $\mathcal{C}_i$. The naive algorithm takes $\Theta(|\mathcal{G}||\mathcal{M}|)$ time. We solve the latter problem more efficiently by recursively solving the ACF problem defined below.

Throughout this section, for technical convenience, the vertices of a plane graph are indexed by distinct positive integers. The faces are indexed by positive integers or -1. The faces indexed by positive integers have distinct indices and are called the *positive* faces. Those indexed by -1 are the *negative* faces.

Let $\mathcal{H}$ be a plane graph. A *vf-set* of $\mathcal{H}$ is a set of vertices and positive faces in $\mathcal{H}$. A *vf-family* is a family of vf-sets; a *vf-sequence* is a sequence of vf-families. Let $\mathcal{N}$ be a vf-sequence $\mathcal{D}_1, \ldots, \mathcal{D}_q$ of $\mathcal{H}$. Also, let $\mathcal{D} = \{S_1, \ldots, S_d\}$ be a vf-family of $\mathcal{H}$.

*A preliminary version was presented at SODA'99.

Let $\mathcal{F}_i$ be the set of faces in S_i. Let U_i be the set of vertices in S_i.

- $|\mathcal{D}| = |S_1| + \cdots + |S_d|$; similarly, $|\mathcal{N}| = |\mathcal{D}_1| + \cdots + |\mathcal{D}_q|$.
- $\Lambda_v(\mathcal{D})$ is the set of vertices in the intersection of the vf-sets in $\mathcal{D}$.
- $\Lambda_f(\mathcal{H}, \mathcal{D})$ is the set of positive faces F of $\mathcal{H}$ such that for each S_i, F is a face in S_i or its boundary intersects $S_i - \Lambda_v(\mathcal{D})$.
- $\mathrm{ACF}(\mathcal{H}, \mathcal{D}) = \Lambda_v(\mathcal{D}) \cup \Lambda_f(\mathcal{H}, \mathcal{D})$.

The ACF problem is the following:

- **Input**: $\mathcal{H}$ and $\mathcal{N}$.
- **Output**: $\mathrm{ACF}(\mathcal{H}, \mathcal{D}_1), \ldots, \mathrm{ACF}(\mathcal{H}, \mathcal{D}_q)$.

To solve the ACF problem recursively, $\mathcal{H}$ need not be simple or triconnected. Furthermore, those faces that are indexed by -1 are ruled out as final output during recursions. To solve the problem efficiently, each vertex in $\Lambda_v(\mathcal{D}_i)$ is meant as a succinct representation of all the faces whose boundaries contain that vertex. Similarly, the positive faces in the input $\mathcal{D}_i$ and the output are represented by their indices.

The next lemma relates the CFE problem and the ACF problem.

Lemma 3. 1 *Let the faces of $\mathcal{G}$ be indexed by positive integers. Then, the output to the CFE problem is "yes" iff for all C_i, $\mathrm{ACF}(\mathcal{G}, C_i) \neq \emptyset$.*

3.1 A counting lemma

Lemma 3. 2 *1. Let v_1 and v_2 be distinct vertices in $\mathcal{G}$. Let F_1 and F_2 be distinct faces in $\mathcal{G}$. Then, both v_1 and v_2 are on the boundaries of both F_1 and F_2 iff v_1 and v_2 form a boundary edge of both F_1 and F_2.*

2. Given a set U of vertices in $\mathcal{G}$, there are $O(|U|)$ faces in $\mathcal{G}$ whose boundaries each contain at least two vertices in U.

3. Given a set $\mathcal{F}$ of faces in $\mathcal{G}$, there are $O(|\mathcal{F}|)$ vertices in $\mathcal{G}$ which are each on the boundaries of at least two faces in $\mathcal{F}$.

Corollary 3. 3 *If $\mathcal{H}$ is simple and triconnected, then the output of the ACF problem has size $O(|\mathcal{N}|)$.*

3.2 A simplification technique

To solve the ACF problem efficiently, we simplify the input graph by removing unnecessary edges and vertices as follows.

For a vf-set S of $\mathcal{H}$, the topological subgraph $\mathcal{H}\Diamond S$ of $\mathcal{H}$ constructed as follows is said to *simplify* $\mathcal{H}$ over S.

Let U and $\mathcal{F}$ be the sets of vertices and positive faces in S, respectively. Let $\mathcal{F}_U$ be the set of the positive faces in $\mathcal{H}$ whose boundaries each contain at least two distinct vertices in U. Let V' and E' be the sets of boundary vertices and edges of the faces in $\mathcal{F} \cup \mathcal{F}_U$, respectively. Let $\mathcal{H}'$ be the plane subgraph of $\mathcal{H}$ consisting of $V' \cup U$ and E'.

Let U' be the set of vertices which are of degree at least three in $\mathcal{H}'$; note that each vertex in U' appears on the boundaries of at least two faces in $\mathcal{F} \cup \mathcal{F}_U$. A *critical* path P in $\mathcal{H}'$ is a maximal path such that (1) every internal vertex of P appears only once in it, and (2) no internal vertex of P is in $U \cup U'$. By the choice of U', every internal vertex of a critical path is of degree 2 in $\mathcal{H}'$. We use this property to further simplify $\mathcal{H}'$. Let $\mathcal{H}\Diamond S$ be the plane graph obtained from $\mathcal{H}'$ by replacing each critical path with an edge between its endpoints. This edge is embedded by the same curve in the plane as the path is. For technical consistency, if a critical path forms a cycle and its endpoint is not in $U \cup U'$, then we replace it with a self-loop for the vertex of the cycle with the smallest index.

Each vertex in $\mathcal{H}\Diamond S$ is given the same index as in $\mathcal{H}$. The closure of the interior of each face of $\mathcal{H}\Diamond S$ is the union of those of several faces or just one in $\mathcal{H}$. Let F be a face in $\mathcal{H}\Diamond S$ and F' be one in $\mathcal{H}$. Let σ (resp., σ') denote the closure of the interior of F (resp., F'). If $\sigma = \sigma'$, then F and F' are regarded as the same face, and F is assigned the same index in $\mathcal{H}\Diamond S$ as F' is in $\mathcal{H}$. For technical conciseness, these two faces are identified with each other. If σ is the union of the closures of the interiors of two or more faces in $\mathcal{H}$, F is not the same as any face in $\mathcal{H}$ and is indexed by -1.

Lemma 3. 4 *1. Given $\mathcal{H}$ and S, we can compute $\mathcal{H}\Diamond S$ in $O(|\mathcal{H}| + |S|)$ time.*

2. Let S' be a vf-set of $\mathcal{H}\Diamond S$. If $S' \subseteq S$, then $\mathcal{H}\Diamond S' = (\mathcal{H}\Diamond S)\Diamond S'$.

3. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $S \subseteq S^*$, then $|\mathcal{H}\Diamond S| = O(|S|)$.

3.3 Algorithms for ACF

To solve the ACF problem recursively, we use simplification to reduce the number of $\mathcal{D}_i$ and the number of sets in each $\mathcal{D}_i$.

For brevity, let $\mathcal{H}\Diamond\mathcal{D} = \mathcal{H}\Diamond(S_1 \cup \cdots \cup S_d)$; similarly, $\mathcal{H}\Diamond\mathcal{N} = \mathcal{H}\Diamond(\mathcal{D}_1 \cup \cdots \cup \mathcal{D}_q)$. Given a vf-set S^* of $\mathcal{H}$, we say $\mathcal{D} \leq S^*$ if $S_i \subseteq S^*$ for all S_i; we say $\mathcal{N} \leq S^*$ if $\mathcal{D}_i \leq S^*$ for all $\mathcal{D}_i$.

Lemmas 3. 5 and 3. 6 below reduce to 1 the number of $\mathcal{D}_i$ in $\mathcal{N}$ in the ACF problem.

Lemma 3. 5 *Assume $q \geq 2$. Let $\mathcal{N}_\ell = \mathcal{D}_1, \ldots, \mathcal{D}_{\lceil q/2 \rceil}$. Let $\mathcal{N}_r = \mathcal{D}_{\lceil q/2 \rceil + 1}, \ldots, \mathcal{D}_q$. Let $\mathcal{H}_\ell = \mathcal{H}\Diamond\mathcal{N}_\ell$ and $\mathcal{H}_r = \mathcal{H}\Diamond\mathcal{N}_r$.*

1. Given $\mathcal{H}$ and $\mathcal{N}$, we can compute $\mathcal{H}_\ell$ and $\mathcal{H}_r$ in $O(|\mathcal{H}|+|\mathcal{N}|)$ total time.

2. For $1 \le i \le \lceil q/2 \rceil$, $\mathcal{H}\Diamond\mathcal{D}_i = \mathcal{H}_\ell\Diamond\mathcal{D}_i$. Similarly, for $\lceil q/2\rceil+1 \le i \le q$, $\mathcal{H}\Diamond\mathcal{D}_i = \mathcal{H}_r\Diamond\mathcal{D}_i$.

3. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \le S^*$, then $|\mathcal{H}_\ell| = O(|\mathcal{D}_1|+\cdots+|\mathcal{D}_{\lceil q/2\rceil}|)$ and $|\mathcal{H}_r| = O(|\mathcal{D}_{\lceil q/2\rceil+1}|+\cdots+|\mathcal{D}_q|)$.

Lemma 3.6 *Assume $q \ge 1$. Let $\mathcal{H}_i = \mathcal{H}\Diamond\mathcal{D}_i$.*

1. $\mathrm{ACF}(\mathcal{H},\mathcal{D}_i) = \mathrm{ACF}(\mathcal{H}_i,\mathcal{D}_i)$.

2. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \le S^*$, then $|\mathcal{H}_i| = O(|\mathcal{D}_i|)$.

3. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \le S^*$, then given $\mathcal{H}$ and $\mathcal{N}$, we can compute all $\mathcal{H}_i$ in $O(|\mathcal{H}|+|\mathcal{N}|\log(q+1))$ total time.

Lemma 3.7 *Let $\mathcal{D}_\ell = \{S_1,\ldots,S_{\lceil d/2\rceil}\}$ and $\mathcal{D}_r = \{S_{\lceil d/2\rceil+1},\ldots,S_d\}$. Let $\mathcal{H}_\ell = \mathcal{H}\Diamond\mathcal{D}_\ell$ and $\mathcal{H}_r = \mathcal{H}\Diamond\mathcal{D}_r$. Let $\mathcal{D}' = \{\mathrm{ACF}(\mathcal{H}_\ell,\mathcal{D}_\ell), \mathrm{ACF}(\mathcal{H}_r,\mathcal{D}_r)\}$.*

1. $\mathrm{ACF}(\mathcal{H},\mathcal{D}) = \mathrm{ACF}(\mathcal{H},\mathcal{D}')$.

2. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{D} \le S^*$, then given $\mathcal{H}$ and $\mathcal{D}$, $\mathrm{ACF}(\mathcal{H},\mathcal{D})$ can be computed in $O(|\mathcal{H}|+|\mathcal{D}|\log(d+1))$ time.

Theorem 3.8 *1. Let d be the maximum number of vf-sets in any $\mathcal{D}_i$ in $\mathcal{N}$. If $\mathcal{H}$ simplifies $\mathcal{G}$ over a vf-set S^* with $\mathcal{N} \le S^*$, then the ACF problem can be solved in $O(|\mathcal{H}|+|\mathcal{N}|\log(d+q))$ time.*

2. Let d be the maximum number of vertex sets in any $\mathcal{C}_i$ in $\mathcal{M}$. Case M3 of the CFE problem can be solved in $O(|\mathcal{G}|+|\mathcal{M}|\log(d+q))$ time.

4 Reduction to Case M1

Let G be a graph. Let $V(G)$ denote the set of vertices in G. A *cut* vertex of G is one whose removal increases the number of connected components in G; a *block* is a maximal subgraph with no cut vertex. Let $\Psi(G)$ denote the forest whose vertices are the cut vertices v and the blocks B of G and whose edges are those $\{v, B\}$ with $v \in B$. $\Psi(G)$ is a tree if G is connected. A set U is *G-local* if $U \subseteq V(G)$. A family $\mathcal{C}$ of sets is *G-local* if every set in $\mathcal{C}$ is G-local. For a vertex subset W of G, let $G - W$ denote the graph obtained from G by deleting the vertices in W.

Let $\mathcal{G}_1,\ldots,\mathcal{G}_k$ be the connected components of $\mathcal{G}$. A family $\mathcal{C}_h$ in $\mathcal{M}$ is *global* if for every $i \in \{1,\ldots,k\}$, $\mathcal{C}_h$ is not $\mathcal{G}_i$-local. Let H be an edge-labeled graph defined as follows. The vertices of H are $\mathcal{G}_1,\ldots,\mathcal{G}_k$. For each global $\mathcal{C}_h$, H contains a cycle C possibly of length 2 where (1) the vertices of C are those $\mathcal{G}_i$ such that some set in $\mathcal{C}_h$ is $\mathcal{G}_i$-local and (2) the edges of C are all labeled $\mathcal{C}_h$.

Let B_1, ..., B_p be the blocks of H. Then, for each global $\mathcal{C}_h$, exactly one B_j contains all the edges labeled $\mathcal{C}_h$. For every B_j, let $\mathcal{U}_j$ be the family consisting of all sets U such that some edge of B_j is labeled $\mathcal{C}_h$ with $U \in \mathcal{C}_h$. For each $\mathcal{G}_i$, let $\mathcal{M}_i$ be the sequence consisting of the $\mathcal{G}_i$-local families in $\mathcal{M}$ as well as the family $\mathcal{U}_{j,i} = \{U \in \mathcal{U}_j \mid U \text{ is } \mathcal{G}_i\text{-local}\}$ for each B_j with $\mathcal{G}_i \in V(B_j)$.

Theorem 4.1 *$\mathcal{G}$ satisfies $\mathcal{M}$ iff every $\mathcal{G}_i$ satisfies $\mathcal{M}_i$. Consequently, Theorem 2.2 holds if it holds for Case M1.*

5 Reducing Case M1 to M2

This section assumes that $\mathcal{G}$ is connected.

Let w be a cut vertex of $\mathcal{G}$. Let W_1, ..., W_k be the vertex sets of the connected components of $\mathcal{G} - \{w\}$. Let $\mathcal{G}_i$ be the subgraph of $\mathcal{G}$ induced by $\{w\}\cup W_i$. $\mathcal{G}_1,\ldots,\mathcal{G}_k$ are the *augmented components* induced by w. For each $\mathcal{C}_h$ in $\mathcal{M}$, let $U_{h,1},\ldots,U_{h,t_h}$ be the sets in $\mathcal{C}_h$ containing w; possibly $t_h = 0$. $\mathcal{C}_h$ is *w-global* if for each $\mathcal{G}_i$, $\mathcal{C}_h - \{U_{h,1},\ldots,U_{h,t_h}\}$ is not $\mathcal{G}_i$-local; otherwise, $\mathcal{C}_h$ is *w-local*.

Lemma 5.1 *1. Assume $\mathcal{C}_h - \{U_{h,1},\ldots,U_{h,t_h}\}$ is $\mathcal{G}_i$-local for some $\mathcal{G}_i$. Then, $\mathcal{G}$ satisfies $\mathcal{M}$ iff $\mathcal{G}$ satisfies $\mathcal{M}$ with $\mathcal{C}_h$ replaced by $(\mathcal{C}_h - \{U_{h,1},\ldots,U_{h,t_h}\}) \cup \{U_{h,1}\cap V(\mathcal{G}_i),\ldots,U_{h,t_h}\cap V(\mathcal{G}_i)\}$.*

2. Assume that $\mathcal{C}_h$ is w-global. Then, $\mathcal{G}$ satisfies $\mathcal{M}$ iff $\mathcal{G}$ satisfies $\mathcal{M}$ with $\mathcal{C}_h$ replaced by $\mathcal{C}_h - \{U_{h,1},\ldots,U_{h,t_h}\}$.

By Lemma 5.1, we may assume that (1) each set in a w-global family in $\mathcal{M}$ does not contain w and (2) each set in a family in $\mathcal{M}$ is $\mathcal{G}_i$-local for some $\mathcal{G}_i$. Let H be an edge-labeled graph constructed as follows. The vertices of H are $\mathcal{G}_1$, ..., $\mathcal{G}_k$. For each w-global family $\mathcal{C}_h$, H has a cycle C possibly of length 2 where (1) the vertices of C are those $\mathcal{G}_i$ such that at least one set in $\mathcal{C}_h$ is $\mathcal{G}_i$-local and (2) the edges of C are all labeled $\mathcal{C}_h$.

Let B_1, ..., B_p be the blocks of H. Clearly, for each w-global family $\mathcal{C}_h \in \mathcal{M}$, exactly one block of H contains all the edges labeled $\mathcal{C}_h$. For each B_j, let $\mathcal{U}_j$ be the family consisting of $\{w\}$ and all $U \subseteq V(\mathcal{G})$ such that some edge of B_j is labeled $\mathcal{C}_h$ with $U \in \mathcal{C}_h$. For each $\mathcal{G}_i$, let $\mathcal{M}_i$ be the sequence consisting of the $\mathcal{G}_i$-local families in $\mathcal{M}$ as well as the family $\mathcal{U}_{j,i} = \{U \in \mathcal{U}_j \mid U \subseteq V(\mathcal{G}_i)\}$ for each B_j with $\mathcal{G}_i \in V(B_j)$.

Lemma 5.2 *$\mathcal{G}$ satisfies $\mathcal{M}$ iff every $\mathcal{G}_i$ satisfies $\mathcal{M}_i$.*

By Lemma 5.2, Case M1 can be reduced to Case M2 in quadratic time. The inefficiency comes from the one-by-one removal of cut vertices. Using the union-find data structure and splay trees, we can remove all the cut vertices in almost linear time.

6 Case M2

We here assume $\mathcal{G}$ is biconnected, and prove:

Theorem 6. 1 *Theorem 2. 2 holds for Case M2.*

6.1 SPQR decompositions

A *planar st-graph* G is an acyclic plane digraph such that G has exactly one source s and exactly one sink t, and both vertices are on the exterior face. These two vertices are the *poles* of G.

A *split pair* of G is either a pair of adjacent vertices or a pair of vertices whose removal disconnects G. A *split component* of a split pair $\{u, v\}$ is either an edge (u, v) or a maximal subgraph C of G such that C is a planar uv-graph and $\{u, v\}$ is not a split pair of C. A split pair $\{u, v\}$ of G is *maximal* if there is no other split pair $\{u', v'\}$ in G with $\{u, v\}$ in a split component of $\{u', v'\}$.

The *decomposition tree* T of G is a rooted ordered tree recursively defined in four cases as follows. The nodes of T are of four types S, P, Q, and R. Each node μ of T has an associated planar st-graph ske(μ), called the *skeleton* of μ. Also, μ is associated with an edge in the skeleton of the parent ϕ of μ, called the *virtual edge* of μ in ske(ϕ).

Case Q: G is a single edge from s to t. Then, T is a Q-node whose skeleton is G.

Case S: G is not biconnected. Let $c_1, \ldots, c_{k-1}$ with $k \geq 2$ be the cut vertices of G. Since G is a planar st-graph, each c_i is in exactly two blocks G_i and G_{i+1} with $s \in G_1$ and $t \in G_k$. Then, T's root is an S-node μ, and ske(μ) consists of the chain $e_1, \ldots, e_k$ and the edge (s, t), where the edge e_i goes from c_{i-1} to c_i, $c_0 = s$, and $c_k = t$.

Case P: $\{s, t\}$ is a split pair of G with at least two split components. Then, T's root is a P-node μ, and ske(μ) consists of $k + 1$ parallel edges $e_1, \ldots, e_{k+1}$ from s to t.

Case R: Otherwise. Let $\{s_1, t_1\}, \ldots, \{s_k, t_k\}$ with $k \geq 1$ be the maximal split pairs of G. Let G_i be the union of the split components of $\{s_i, t_i\}$. Then, T's root is an R-node μ, and ske(μ) is the simple triconnected graph obtained from G by replacing each G_i with an edge e_i from s_i to t_i and inserting the edge (s, t).

In the last three cases, μ has children $\chi_1, \ldots, \chi_k$ in this order, such that each χ_i is the root of the decomposition tree of G_i. The virtual edge of χ_i is the edge e_i in ske(μ). G_i is called the *pertinent graph* pert(χ_i) of χ_i as well as the *expansion graph* of e_i. G is the pertinent graph of T's root. Also, no child of an S-node is an S-node, and no child of a P-node is a P-node.

The *allocation nodes* of a vertex v of G are the nodes of T whose skeleton contains v; note that v has at least one allocation node.

In the above description of T, for each non-leaf node μ, an additional edge is added into ske(μ) between its two poles (which does not correspond to any child of μ). This additional edge has no effect on our algorithm. From now on, we ignore this edge in ske(μ).

For each non-S-node μ in T, pert(μ) is called a *block* of G [2], which differs from that in §4 and §5. For a block $B = $ pert(μ), let node$(B) = \mu$. For an ancestor ϕ of node(B), the *representative* of B in ske(ϕ) is the edge in ske(ϕ) whose expansion graph contains B.

Let μ be an R- or P-node in T with children $\chi_1, \ldots, \chi_b$. For each $k \in \{1, \ldots, b\}$, let e_k be the virtual edge of χ_k in ske(μ). If χ_k is an S-node, pert(χ_k) is a chain consisting of two or more blocks. If χ_k is an R-nod or P-node, pert(χ_k) is a single block. For each $k \in \{1, \ldots, b\}$, we say that the blocks in pert(χ_k) are *on edge* e_k. The *minor blocks* of pert(μ) are the blocks on e_1, ..., those on e_b.

6.2 Basic ideas

An *st-orientation* of a planar graph is an orientation of its edges together with an embedding such that the resulting digraph is a planar st-graph.

Fact 1 (see [2]) If an n-vertex planar graph has an st-orientation, then every embedding, where s and t are on the exterior face, of this graph can be obtained from this orientation through a sequence of $O(n)$ following operations:

- Select one of the two possible flips of an R-node's skeleton around its poles.
- Permute the skeletons of a P-node's children with respect to their common poles.

Let $\{s, t\}$ be an edge of $\mathcal{G}$. Since $\mathcal{G}$ is biconnected, we convert $\mathcal{G}$ to a planar st-graph in $O(n)$ time for technical convenience. For the remainder of §6, let T be the decomposition tree of $\mathcal{G}$. Also, let $\mathcal{C}_i = \{U_{i,1}, \ldots, U_{i,r_i}\}$.

Let μ be a node of T. T_μ denotes the subtree of T rooted at μ and dep(μ) denotes the distance from T's root to μ.

- $U_{i,j}$ is *contained* in pert(μ) if the vertices of $U_{i,j}$ are all in pert(μ); $U_{i,j}$ is *strictly contained* in pert(μ) if in addition, no pole of pert(μ) is in $U_{i,j}$.
- Let done$(U_{i,j})$ be the deepest node μ in T such that $U_{i,j}$ is strictly contained in pert(μ), if such a node exists. If no such μ exists, then $U_{i,j}$ contains a pole of $\mathcal{G}$ and let done$(U_{i,j})$ be T's root.
- A family $\mathcal{C}_i$ *straddles* pert(μ) if at least one set in $\mathcal{C}_i$ is strictly contained in pert(μ), and at least one set in $\mathcal{C}_i$ has no vertex in pert(μ).

• Let done($\mathcal{C}_i$) be the deepest node μ in T such that for every $U_{i,j} \in \mathcal{C}_i$, at least one vertex of $U_{i,j}$ is in pert(μ).

• Let sub(μ) $= \{U_{i,j} \mid \text{done}(U_{i,j}) = \mu\}$ and fam(μ) $= \{\mathcal{C}_i \mid \text{done}(\mathcal{C}_i) = \mu\}$.

• If μ is a P- or R-node, let xfam(μ) $=$ fam(μ) $\cup$ ($\cup_{\chi_k}$fam(χ_k)) and xsub(μ) $=$ sub(μ) $\cup$ ($\cup_{\chi_k}$sub(χ_k)), where χ_k ranges over all S-children of μ.

In a fixed embedding of a block B, the poles of B divide the boundary of its exterior face into two paths side$_1$(B) and side$_2$(B), called the two *sides* of B. $U_{i,j}$ is *two-sided* for B if both side$_1$(B) and side$_2$(B) intersect $U_{i,j}$. In particular, $U_{i,j}$ is two-sided for B if it contains a pole of B. $U_{i,j}$ is *side-1* (resp., *side-2*) for B if only side$_1$(B) (resp., side$_2$(B)) intersects $U_{i,j}$. Assume that B is a minor block of pert(μ) for some μ. Let e_k be the representative of B in ske(μ). In a fixed embedding of ske(μ), e_k separates two faces F and F'. When embedding pert(μ), we can embed side$_1$(B) towards either F or F', referred to as the two *orientations* of B in pert(μ).

A family $\mathcal{C}_i$ is *side-0* (resp., *side-1* or *side-2*) *exterior-forcing* for B if done($\mathcal{C}_i$) is an ancestor of node(B) in T and some $U_{i,j} \in \mathcal{C}_i$ strictly contained in B is two-sided (resp., side-1 or side-2) for B. For $p = 0$, 1, 2, define

• ext$_p$(B) $= \min\{\text{dep}(\text{done}(\mathcal{C}_i)) \mid \mathcal{C}_i$ is side-p exterior-forcing for $B\}$, if there is side-p exterior-forcing for B;

• ext$_p$(B) $= \infty$ otherwise.

Assume ext$_p$(B) $\neq \infty$. Let $\mu = \text{node}(B), \phi_1, \phi_2, \ldots, \phi_h$ be the path in T from μ to ϕ_h, where dep(ϕ_h) $=$ ext$_p$(B). For each $\ell \in \{1, \ldots, h-1\}$, the representative of B in ske(ϕ_ℓ) must be an exterior edge in any satisfying embedding of ske(ϕ_ℓ). In addition, if $p = 1$ or 2, side$_p$(B) must be embedded towards the exterior face of the embedding of pert(ϕ_ℓ).

Since (s, t) is an edge of $\mathcal{G}$, the root ρ of T is a P-node and has a child Q-node ϕ representing (s, t). A subtle difference between ρ and each non-root node of T is that the two sides of $\mathcal{G}$ = pert(ρ) is actually on the same face. To eliminate this difference, we delete ϕ from T; afterwards, if ρ has only one child, we further delete ρ from T. From here onwards, T denotes this modified tree.

6.3 The CFE algorithm

The CFE algorithm processes T from bottom up. A *ready* node μ of T is either (1) a leaf node or (2) a P- or R-node such that the non-S-children of μ and the children of every S-child of μ all have been processed. The CFE algorithm processes the ready nodes of T in an arbitrary order. An S-node is processed when its parent is processed. We detail how to process μ as follows.

For the case where μ is a leaf node of T, note that pert(μ) is a single edge of $\mathcal{G}$. Since no $U_{i,j}$ is strictly contained in pert(μ), fam(μ) = sub(μ) $= \emptyset$. Therefore, we simply set ext$_p$(pert(μ)) $= \infty$ for $p = 0, 1, 2$,

We next consider the case where μ is a non-leaf ready node. Before μ is processed, an embedding of every minor block of pert(μ) is already fixed, except for a possible flip around its poles. Moreover, for each minor block B of pert(μ) and each $p \in \{0, 1, 2\}$, ext$_p$(B) is known. When processing μ, the CFE algorithm checks whether some embedding Φ_μ of pert(μ) satisfies the following two conditions:

• Φ_μ satisfies every $\mathcal{C}_i$ in xfam(μ).

• For each $\mathcal{C}_i$ straddling pert(μ) and each $U_{i,j} \in \mathcal{C}_i$ strictly contained in pert(μ), at least one vertex of $U_{i,j}$ is embedded on the exterior face of Φ_μ. (*Remark.* This ensures the existence of an embedding of pert(done($\mathcal{C}_i$)) satisfying $\mathcal{C}_i$ later.)

If no such Φ_μ exists, then $\mathcal{G}$ cannot satisfy $\mathcal{M}$ and the CFE algorithm outputs "no" and stops. Otherwise, it finds such an Φ_μ and fixes it except for a possible flip around its poles. It also computes ext$_p$(pert(μ)) for $p = 0$, 1, 2.

To detail how to process μ, we classify the sets $U_{i,j}$ in each $\mathcal{C}_i$ into four types and define a set img($U_{i,j}, \mu$) for each type as follows.

Type 1: $U_{i,j}$ contains at least one pole of ske(μ). Then, done($U_{i,j}$) is an ancestor of μ. Let img($U_{i,j}, \mu$) $= \{v \in U_{i,j} \mid v$ is a vertex in ske(μ)$\}$.

Type 2: $U_{i,j}$ contains at least one vertex but no pole of ske(μ). Then, done($U_{i,j}$) $= \mu$. Let img($U_{i,j}, \mu$) as in the case of type 1.

Type 3: $U_{i,j}$ is strictly contained in pert(χ) for some S-node χ of μ and $U_{i,j}$ contains at least one vertex in ske(χ). Then, done($U_{i,j}$) $= \chi$. Let img($U_{i,j}, \mu$) $= \{$virtual edge of χ in ske(μ)$\}$.

Type 4: $U_{i,j}$ is strictly contained in a minor block B of pert(μ). Then, done($U_{i,j}$) is node(B) or its descendent. Let img($U_{i,j}, \mu$) $= \{$ representative of B in ske(μ)$\}$.

Each element of img($U_{i,j}, \mu$) is called an *image* of $U_{i,j}$ in ske(μ). The remainder of §6.3 details how to process of μ.

6.3.1 Processing an S-child

When processing μ, for each S-child χ of μ, we need to find an embedding of pert(χ) satisfying certain conditions. We call this process the *S-procedure* and describe it below.

Let χ be an S-child of μ. Then, ske(χ) is a path. Let e_1, ..., e_b be the edges in ske(χ). For each $k \in \{1, \ldots, b\}$, let B_k be the expansion graph of e_k. Before the S-procedure is called on χ, the following requirements are met:

- For each $k \in \{1, \ldots, b\}$, an embedding of B_k has been fixed, except for a possible flip around its poles.
- For some $k \in \{1, \ldots, b\}$ and $p \in \{1, 2\}$, side$_p(B_k)$ is required to face either the left or the right side of ske(χ).

Our only choice for embedding pert(χ) is to flip B_1, ..., B_b around their poles. We need to check whether for some combination of flippings of B_1, ..., B_b, (1) the resulting embedding satisfies every $C_i \in$ fam(χ) and (2) the second requirement above is met.

The S-procedure consists of the following five stages:

Stage S1 constructs an auxiliary graph $D = (V_D, E_D)$ with $V_D = \{k_p \mid 1 \leq k \leq b,\ p = 1, 2\}$ as follows. For each $C_i \in$ fam(χ), insert a path P_i into D to connect all $k_p \in V_D$ such that for some type-4 $U_{i,j} \in C_i$, (a) img$(U_{i,j}, \chi) = \{e_k\}$ and (b) $U_{i,j}$ is side-p for B_k. To avoid confusion, we call the elements of V_D *points*, and the connected components of D *clusters*. Those points $k_p \in V_D$ such that side$_p(B_k)$ is required to face the left side of ske(χ) are called *L-points*. *R-points* are defined similarly. For each cluster C of D, all side$_p(B_k)$ where k_p ranges over all the points in C must be embedded toward the same side of ske(χ). Also, each type-3 $U_{i,j}$ in C_i contains a vertex in ske(χ) which is on both sides of ske(χ). For this reason, such sets were not considered when constructing D.

Stage S2 checks whether there is a cluster of D containing both an L-point and an R-point. If such a cluster exists, then S2 outputs "no" and stops. Otherwise, each cluster C consists of either L-points only or R-points only. In the former (resp., latter) case, we call C an *L-cluster* (resp., *R-cluster*).

Stage S3 constructs another auxiliary graph $RD = (V_{RD}, E_{RD})$ from D as follows. The vertices of RD are the clusters of D. For each $k \in \{1, \ldots, b\}$, there is an edge $\{C_1, C_2\}$ in RD, where C_1 (resp., C_2) is the cluster of D containing point k_1 (resp., k_2). RD may have self-loops.

Stage S4 checks whether RD is bipartite. If it is not, then S4 outputs "no" and stops. Otherwise, for each connected component K of RD, the clusters in K can be uniquely partitioned into two independent subsets $V_{K,1}$ and $V_{K,2}$ of clusters. If $V_{K,1}$ or $V_{K,2}$ contains both an L-cluster and an R-cluster, S4 outputs "no" and stops. Otherwise, V_{RD} can be partitioned into two independent subsets V^L_{RD} and V^R_{RD} of clusters such that all L-clusters are in V^L_{RD} and all R-clusters are in V^R_{RD}. Let $V^L_D = \{k_p \mid k_p$ is in a cluster in $V^L_{RD}\}$ and $V^R_D = \{k_p \mid k_p$ is in a cluster in $V^R_{RD}\}$.

Stage S5 embeds side$_p(B_k)$ toward the left side of ske(χ) for each $k_p \in V^L_D$.

6.3.2 μ is an R-node

In this case, ske(μ) is a simple triconnected graph and has a unique embedding. Let χ_1, ..., χ_b be the children of μ in T. For each $k \in \{1, \ldots, b\}$, let $B_{k,1}$, ..., B_{k,s_k} be the minor blocks of pert(μ) in pert(χ_k). When χ_k is an R- or P-node, $s_k = 1$. To process μ, the CFE algorithm proceeds in five stages.

6.3.3 μ is a P-node

In this case, ske(μ) consists of parallel edges $e_1, e_2, \ldots, e_b$ between its two poles with $b \geq 2$. Let χ_1, ..., χ_b be the children of μ in T. For each $k \in \{1, \ldots, b\}$, let $B_{k,1}$, ..., B_{k,s_k} be the minor blocks of pert(μ) in pert(χ_k). When embedding ske(μ), edges e_1 through e_b can be embedded in any order. The CFE algorithm first finds a proper embedding of ske(μ) in three stages. It then tries to embed pert(μ) based on the embedding of ske(μ).

References

[1] Z.-Z. Chen, M. Grigni and C.H. Papadimitriou. Planar Map Graphs. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 514-523, 1998.

[2] G. Di Battista and R. Tamassia. On-Line Planarity Testing. SIAM J. Comput., 25(5):956-997, 1996.