ELSEVIER

Theoretical Computer Science 181 (1997) 247–266

Theoretical Computer Science

Algorithmic graph embeddings[1]

Jianer Chen*

Department of Computer Science, Texas A&M University, College Station, TX 77843-3112, USA

Abstract

The complexity of embedding a graph into a variety of topological surfaces is investigated. A new data structure for graph embeddings is introduced and shown to be superior to the previously known data structures. In particular, the new data structure efficiently supports all on-line operations for general graph embeddings. Based on this new data structure, very efficient algorithms are developed to solve the problem "given a graph G and an integer k, construct a genus k embedding for the graph G" for a large range of integers k and for a large class of graphs G.

1. Introduction

Graph embedding is a fundamental yet difficult problem, and it has many applications in diverse problem domains. The most studied is graph planar embeddings. The well-known Hopcroft–Tarjan algorithm shows that a planar embedding of a planar graph can be constructed in linear time [21]. Graph planar embeddings have been studied extensively in a variety of areas such as computational geometry [25] and graph drawing [11].

On the other hand, the computational complexity of constructing embeddings of a graph into non-planar surfaces is not well understood yet. The complexity of the graph minimum genus problem remained as a basic open problem in Garey and Johnson's list [18] for ten years until Thomassen proved that the problem is NP-complete [27]. Algorithms have also been developed for embedding a graph into a variety of surfaces. It is demonstrated by Furst et al. [17] that a maximum genus embedding of a graph can be constructed in time $O(n^4 \log^6 n)$. Filotti [13] described an $O(n^6)$ time algorithm for embedding cubic graphs of minimum genus 1 into the torus. Filotti et al. [14] derived an $O(n^{O(k)})$ time algorithm that embeds graphs of minimum genus $\leqslant k$ into

* Tel.: +1 (409) 845 4259; fax: +1 (409) 847 8578; e-mail: chen@cs.tamu.edu; supported in part by the National Science Foundation under Grant CCR-9110824.

[1] A preliminary version of this work was reported at The *1st Ann. Internat. Computing and Combinatorics Conf.* (COCOON'95), Lecture Notes in Computer Science, Vol. 959, pp. 151–160.

PII S0304-3975(96)00273-3

a surface of genus k, which was improved recently by Djidjev and Reif [12] who developed an algorithm of time $O(2^{O(k)!}n^{O(1)})$. Frederickson and others have considered the complexity of graph embeddings that give the minimum "hammock number" or minimum "face cover", and studied their applications to other computational graph problems [15, 16]. Recently, Mohar described a linear time algorithm for embedding graphs of crosscap number 1 into the projective plane [24], and Chen described a linear time algorithm for embedding graphs of bounded average genus [2]. Chen et al. have also studied approximation algorithms for graph minimum genus embeddings [10].

An open problem posed by Furst et al. [17] is to determine the complexity of graph embeddings into a general surface. There are several theoretical and practical reasons why this problem should be studied. First, the distribution of graph embeddings into topological surfaces provides a very useful isomorphism heuristic [5–7, 19]. Secondly, embedding a graph into a certain surface helps efficiently solving other computational graph problems [2, 4, 8, 15–17, 25]. Finally, study of graph embeddings has direct applications in the areas such as circuit layout and VLSI design.

In this paper, we will develop a number of efficient algorithms that embed a graph into a variety of surfaces. We start by introducing a new data structure for graph embeddings. We demonstrate that the new data structure is superior to the previously known data structures for graph embeddings. In particular, the new data structure efficiently supports *all* on-line operations for general graph embeddings. Based on this new data structure, we present an $O(m \log n)$ time greedy algorithm that constructs an embedding of genus at least $(m - n + 1)/8$ for a graph G of n vertices and m edges. Then we show how we can continuously move in time $O(m \log n)$ for a graph from one embedding to another embedding. This implies, in particular, that there is an $O(m \log n)$ time algorithm that, given a planar graph G of n vertices and m edges and given an integer $k \leqslant (m-n+1)/8$, constructs an embedding of genus k for G. We will also study the complexity of embedding a graph into a surface of "average genus", which is an important topological invariant of a graph [5–7]. We present an $O(m^2)$ time randomized algorithm that, given a graph G and an integer k, either reports with a very small error probability that k is not equal to the average genus of G, or produces an embedding of genus k for G. These computational results demonstrate a very rich and interesting structure for computational complexity of graph embeddings.

The paper is organized as follows. Section 2 is an introduction to the preliminaries and definitions for the theory of graph embeddings. The new data structure is described in Section 3. An efficient algorithm is presented in Section 4 to embed a graph into a high genus surface. Section 5 discusses the complexity of embedding a graph into a surface of genus k, for a variety of k. Concluding remarks are given in Section 6.

2. Preliminaries

It is assumed that the reader is somewhat familiar with the fundamentals of graph embeddings. For further description, see [20].

Unless stated explicitly, all graphs in our discussion are supposed to be connected simple graphs in which each vertex is of degree at least 3. This assumption does not lose any generality because "smoothing" a degree 2 vertex or removing a degree 1 vertex has no effect on topological properties of a graph [20]. An *embedding* must have the "cellularity property" that the interior of every face is simply connected.

A *rotation* at a vertex v is a cyclic permutation of the edge-ends incident on v. A list of rotations, one for each vertex of the graph, is called a *rotation system*.

An embedding of a graph G in an orientable surface induces a rotation system, as follows: the rotation at vertex v is the cyclic permutation corresponding to the order in which the edge-ends are traversed in an orientation-preserving tour around v. Conversely, by the Heffter–Edmonds principle, every rotation system induces a unique embedding of G into an orientable surface [20]. This bijectivity enables us to study graph embeddings based on graph rotation systems, a more combinatorial structure. We will interchangeably use the phrases "an embedding of a graph" and "a rotation system of a graph".

Edge insertion and edge deletion are among the most important and fundamental operations for graph embeddings. Most proposed graph embedding algorithms [10, 13, 14, 17, 21] are based on edge insertion and edge deletion. We first discuss the effect of inserting a new edge into an embedded graph.

Let $\rho(G)$ be an embedding of a connected graph G. If a subpath e_1e_2 appears in the boundary walk of a face f in $\rho(G)$, where $e_1 = (v_1, w)$ and $e_2 = (w, v_2)$ are two edges, then we say that e_1we_2 is a *corner* of the face. If the content is apparent, we simply say that the vertex w is a corner of the face f.

Suppose that we insert a new edge $e = (u, v)$ into the embedding $\rho(G)$, where u and v are vertices of G. There are two possible cases.

If the edge-ends of e at u and v are inserted between two corners of the same face f, then the new edge e splits the face f into two faces. More precisely, if the boundary walk around the face f in $\rho(G)$ is of the form $u\alpha v\beta u$, where α and β are subwalks, then the new edge e splits the boundary walk of f into two walks: $u\alpha veu$ and $v\beta uev$, resulting in two new faces. In this case, the embedding genus remains the same.

If the edge-ends of e at u and v are inserted between corners of two different faces f_1 and f_2, then both these faces are merged by e into one larger face. In particular, suppose that the edge e runs from the corner of u in face boundary walk $u\alpha u$ of f_1 to the corner of v in face boundary walk $v\beta v$ of f_2, then the merged face has boundary walk $uev\beta veu\alpha u$. In this case, the embedding genus is increased by 1.

Edge deletion is the inverse operation of edge insertion. Let e be an edge to be deleted from the embedding $\rho(G)$. If e is on the boundary of two different faces αe and $e\beta$, where α and β are subwalks, then deleting the edge e will merge these two faces into a larger face $\alpha\beta$ and keep the same embedding genus. If the two sides of e are on the boundary of the same face $e\alpha e\beta$, then deleting the edge e will result in two new faces α and β and decrease the embedding genus by 1.

Data structures for sets from a universe with a total ordering have been studied extensively [1]. The discussions can be easily generalized to sequences from a universe

without a total ordering. Let $\mathcal{U}$ be a universe that may not have a total ordering. A *sequence* is an ordered list of elements from $\mathcal{U}$, in which each element of $\mathcal{U}$ may appear more than once. Each element appearance in the sequence will be called a *node*. For each sequence, we also assume a *root* that contains the name of the sequence.

With straightforward modifications in the discussion for data structures for sets from a universe with a total ordering [1], a sequence can be represented by a concatenable data structure, such as a 2-3 tree, which supports each of the following operations in time $O(\log n)$ on a sequence of n nodes:

- S-INSERT(r, v, x): insert the element x immediately before the node v in the sequence rooted at r;
- S-DELETE(r, v): delete the node v from the sequence rooted at r;
- CONCAT(r_1, r_2): take as input the roots r_1 and r_2 of the sequences L_1 and L_2, respectively, and output the root of the concatenated sequence L_1L_2;
- SPLIT(r, v): here v is a node in the sequence L rooted at r such that $L = L_1vL_2$. The output of this operation are roots of the two sequences L_1 and L_2;
- ROOT(v): return the root of the sequence containing the node v;

and each of the following operations in $O(1)$ time:

- NEXT(v): return the node following the node v in the sequence containing v (the first node is supposed to follow the last node in a sequence);
- PREC(v): return the node preceding the node v in the sequence containing v (the last node is supposed to precede the first node in a sequence).

3. Data structures

We will evaluate the performance of a data structure for graph embeddings based on the following graph embedding operations:

- FACE-TRACE(f): output a boundary walk of the face f.
- VERTEX-TRACE(v): output the edges incident on the vertex v in the (circular) ordering of the rotation at v.
- COFACIAL(c_1, c_2): return *true* if the two face corners c_1 and c_2 belong to the same face of the current embedding and *false* otherwise.
- INSERT(c_1, c_2, e): insert the new edge e between the face corners c_1 and c_2.
- DELETE(e): delete the edge e from the current embedding.
- GENUS$(\rho(G))$: report the genus of the current embedding $\rho(G)$.

A number of data structures have been proposed. Most of these data structures are only valid for planar embeddings of a graph. Tamassia [26] proposed a data structure that supports a special case of DELETE operation and each of the other operations in $O(\log n)$ time on planar embedded graphs. Very recently, Italiano et al. [22] proposed a new data structure that supports each of the above operations in $O(\log^2 n)$ time on planar embedded graphs.

None of these data structures seems to generalize to general graph embeddings. Frequently, a rotation system of a graph is represented in the *edge-list form*, which for

each vertex v contains the list of its incident edges, arranged in the order according to the rotation at v. It is not difficult to see that this representation does not efficiently support many embedding operations. Another data structure, *the doubly connected edge-list* (DCEL) that has been widely used in computational geometry [25], can be directly used for representing a rotation system of a graph. Suppose that $G = (V, E)$ is a graph and $\rho(G)$ is a rotation system of G. A DCEL for $\rho(G)$ is a table of $|E|$ *edge items*. There is a one-to-one correspondence between edges of G and edge items in the DCEL. Each edge item consists of six fields V_1, V_2, F_1, F_2, P_1, and P_2. For an edge e, the fields $e(V_1)$ and $e(V_2)$ specify the two endpoints of e, the fields $e(F_1)$ and $e(F_2)$ give the "right face" and "left face" of the edge e when we traverse the edge e from the endpoint $e(V_1)$ to the endpoint $e(V_2)$. The field $e(P_1)$ (resp. $e(P_2)$) points to the edge item in the DCEL for the edge that follows the edge e in the rotation at $e(V_1)$ (resp. $e(V_2)$) in $\rho(G)$.

Based on the DCEL structure, the operation FACE-TRACE(f) can be done in time linear in the size of the face f, and the operation VERTEX-TRACE(v) takes time linear in the degree of the vertex v [25]. Moreover, the DCEL and the edge-list form of a graph rotation system can be converted from one to the other in linear time [25].

Unfortunately, the DCEL structure does not seem to support the operations INSERT and DELETE efficiently. An INSERT or a DELETE operation requires to update the fields F_1 and/or F_2 for all edges on the boundary walks of the related faces, which may take time up to $O(n^2)$.

One possible modification is to ignore the fields F_1 and F_2 in a DCEL structure. Indeed, a DCEL structure without the fields F_1 and F_2 still gives a unique rotation system for a graph. However, the resulting structure then does not seem to efficiently support the operations COFACIAL and GENUS.

We introduce a new data structure and show that the new data structure efficiently supports all the operations listed at the beginning of this section.

Each face is given by a sequence of vertices and edges that corresponds to a boundary traversing of the face. The vertex appearances and the edge appearances in the sequence will be called *vertex nodes* and *edge nodes*, respectively. The sequence is represented by a concatenable data structure (e.g., a 2-3 tree).

Definition. Let $\rho(G)$ be an embedding of a graph $G = (V, E)$ with face set F. A *doubly linked face-list* (DLFL) for the embedding $\rho(G)$ is a 4-tuple $L = \langle \mathcal{F}, \mathcal{V}, \mathcal{E}, g \rangle$, where the *face list* $\mathcal{F}$ consists of a set of $|F|$ sequences, each corresponds to a face in the embedding $\rho(G)$. Moreover, the roots of the sequences are connected by a circular doubly linked list. The *vertex array* $\mathcal{V}$ has $|V|$ items, each $\mathcal{V}[i]$ is a linked list of pointers to the vertex nodes of the sequences in $\mathcal{F}$ that are labeled by the corresponding vertex. The *edge array* $\mathcal{E}$ has $|E|$ items, each $\mathcal{E}[i]$ consists of two pointers to the two edge nodes of the sequences in $\mathcal{F}$ that are labeled by the corresponding edge. The integer g is the genus of the embedding $\rho(G)$.

The following theorem shows the relationship between DLFL and DCEL.

Theorem 3.1. *The DLFL structure and the DCEL structure of an embedding of a graph can be converted from one to the other in linear time.*

Proof. Let $\rho(G)$ be an embedding of a graph G with n vertices and m edges. Let the faces of $\rho(G)$ be $f_1, f_2, \ldots, f_h$, with sizes $n_1, n_2, \ldots, n_h$, respectively.

Suppose that $\rho(G)$ is given by a DCEL structure. Then for each i, in time $O(n_i)$, we can construct a boundary walk for the face f_i [25]. It is also well known that in time $O(n_i)$ we can construct a concatenable data structure for the boundary walk of f_i. The vertex array and the edge array in the DLFL structure for $\rho(G)$ can also be constructed during the face traversing. The genus of $\rho(G)$ can be easily computed using Euler's equation. Therefore, the time needed to construct a DLFL structure from a DCEL structure for the embedding $\rho(G)$ is bounded by $O(\sum_{i=1}^{h} n_i) = O(m)$.

Conversely, suppose that $\rho(G)$ is given by a DLFL structure. Then traversing the boundary walk for each face f_i can be obviously done in time $O(n_i)$ because the operation NEXT on a sequence can be done in time $O(1)$. By traversing boundary walks for all faces in $\rho(G)$, we can decide for each edge e of G the fields V_1, V_2, F_1, F_2, P_1, and P_2 in the DCEL structure for $\rho(G)$. For example, suppose that e_1e_2 is a subpath on the boundary walk of a face f in $\rho(G)$, where $e_1 = (v,u)$ and $e_2 = (u,w)$ are two edges in G, then we can assign $e_1(V_1) = v, e_1(V_2) = u$. Once the fields $e_1(V_1)$ and $e_1(V_2)$ are decided, we have immediately $e_1(F_1) = f$ and $e_1(P_2) = e_2$. In this way, all fields of the DCEL will be filled out after the boundary traversing of all faces in $\rho(G)$. Therefore, the DCEL structure for $\rho(G)$ can be constructed in time $O(\sum_{i=1}^{h} n_i) = O(m)$ from the DLFL structure for $\rho(G)$. □

Corollary 3.2. *A DLFL structure for an embedding of a graph can be built in linear time and requires linear storage.*

For those operations that are supported by the DCEL structure efficiently, the DLFL structure has equally good performance.

Theorem 3.3. *Based on the DLFL structure, the operation* FACE-TRACE(f) *can be done in time linear to the size of the face f, and the operation* VERTEX-TRACE(v) *can be done in time linear to the degree of the vertex v.*

Proof. Let v be an arbitrary node in the sequence for the face f in a DLFL structure for $\rho(G)$. Using the NEXT operation on the sequence, which can be done in time $O(1)$, we can easily traverse the boundary walk of f in time linear to the size of the face f.

Now we consider the operation VERTEX-TRACE(v). Let e_0ve_1 be an arbitrary face corner in the embedding $\rho(G)$, which can be found in constant time from the vertex array of the DLFL structure for $\rho(G)$. Note that we must have $e_1 = (v,w)$ for some vertex w. Start from this edge e_1 incident on v. From the edge array of the DLFL structure, we can find in constant time the other edge node labeled e_1 and the corresponding

face corner e_1ve_2. Therefore, the edge e_2 must be the edge following the edge e_1 in the rotation at the vertex v. Now apply the same technique to find the edge following the edge e_2, and so on, until we come back to the edge e_1. Since in constant time we can move from the current edge to the next edge, we conclude that the operation VERTEX-TRACE(v) can be done in time linear to the degree of the vertex v. □

It is obvious that the operation GENUS($\rho(G)$) can be done in constant time based on a DLFL structure because a DLFL structure keeps the value of the genus for the embedding $\rho(G)$. Now we show that the DLFL structure also supports the other embedding operations efficiently.

Theorem 3.4. *The DLFL structure supports the operations COFACIAL, INSERT, and DELETE in logarithmic time.*

Proof. Let $\rho(G)$ be an embedding of a graph G. First note that since the size of a face in $\rho(G)$ cannot be larger than twice of the number m of edges of the graph G, the logarithm of the size of a face in $\rho(G)$ is always bounded by $O(\log m)$.

Also note that a circular rotation $\alpha v\beta \to \beta\alpha v$ of the sequence $\alpha v\beta$, where α and β are subsequences and v is a node, can be done in time $O(\log|\alpha v\beta|)$ when the node v is given. In fact, the circular rotation can be implemented by the SPLIT operation of the sequence $\alpha v\beta$ on the node v to produce two sequences α and β, followed by the CONCAT operations on the sequences β, α, and v.

For the operation COFACIAL(c_1, c_2), where $c_1 = e_1v_1e_1'$ and $c_2 = e_2v_2e_2'$ are two face corners in $\rho(G)$, we apply the operations ROOT(v_1) and ROOT(v_2) to find the roots r_1 and r_2 of the corresponding sequences in the DLFL structure, respectively. This can be done in time $O(\log m)$. Now the two corners c_1 and c_2 belong to the same face in $\rho(G)$ if and only if $r_1 = r_2$.

Now consider the operation INSERT(c_1, c_2, e), where e is an edge to be inserted between the face corners $c_1 = e_1v_1e_1'$ and $c_2 = e_2v_2e_2'$. We first apply the operations ROOT(v_1) and ROOT(v_2) to find the roots r_1 and r_2 of the sequences in the DLFL structure that contain the corners c_1 and c_2. There are two possible cases.

If $r_1 = r_2$, then the corners c_1 and c_2 belong to the same face f so inserting the edge e will split the face f into two faces and unchange the embedding genus. More precisely, suppose that the boundary walk of face f is $B_f = \alpha e_1v_1e_1'\beta e_2v_2e_2'$, where α and β are subwalks, then inserting the edge e will result in two faces with the boundary walks $B_f' = \alpha e_1v_1ev_2e_2'$ and $B_f'' = \beta e_2v_2ev_1e_1'$, respectively. After a possible circular rotation in $O(\log m)$ time on the sequence for the face f in the DLFL structure, we obtain the sequence B_f. Now the two sequences B_f' and B_f'' can be obtained as follows: first we apply SPLIT to the sequence B_f on node e_1' to produce two subsequences αe_1v_1 and $\beta e_2v_2e_2'$. Apply another SPLIT to the sequence $\beta e_2v_2e_2'$ on node e_2' to produce a sequence βe_2v_2. Then the sequence B_f' can be obtained by applying the operation CONCAT on the sequences αe_1v_1, e, v_2, and e_2', and the sequence B_f'' can be obtained by applying the operation CONCAT on the sequences βe_2v_2, e,

v_1, and e'_1. The other updatings on the DLFL structure are straightforward. Since we have only applied a constant number of sequence operations, each taking time at most $O(\log m)$, we conclude that in this case, the INSERT operation can be done in time $O(\log m)$.

If $r_1 \neq r_2$, then the face corners c_1 and c_2 belong to different faces $\alpha e_1 v_1 e'_1$ and $\beta e_2 v_2 e'_2$, where α and β are subwalks. Therefore, inserting the edge e will merge these two faces into a larger face

$$\alpha e_1 v_1 e v_2 e'_2 \beta e_2 v_2 e v_1 e'_1$$

and increase the embedding genus by 1. The sequence for this larger face can be obtained by first applying the operation SPLIT on the sequences $\alpha e_1 v_1 e'_1$ and $\beta e_2 v_2 e'_2$ to produce sequences $\alpha e_1 v_1$ and $\beta e_2 v_2$, then applying the operation CONCAT to concatenate the sequences $\alpha e_1 v_1, e, v_2, e'_2, \beta e_2 v_2, e, v_1$, and e'_1. Since only a constant number of sequence operations are applied, each taking time $O(\log m)$, in this case the INSERT operation can also be done in time $O(\log m)$.

The DELETE(e) can be done similarly. Given the edge e to be deleted from the embedded graph, we first get from the edge array of the DLFL structure the two edge nodes labeled e and use ROOT operation to check whether the two edge nodes belong to the same face in the embedding. If the two edge nodes belong to different faces αe and βe of the embedding, then deleting the edge e will merge the two faces into a single face $\alpha\beta$ with the embedding genus unchanged. The sequence for this new face can be obtained by a constant number of SPLIT and CONCAT operations on the sequences αe and βe. If the two edge nodes labeled e belong to the same face $\delta e \gamma e$, then deleting the edge e will break the face into two faces δ and γ and decrease the embedding genus by 1. The sequences for these two faces can be obtained by a constant number of SPLIT operations on the sequence $\delta e \gamma e$. Therefore, the operation DELETE(e) can be done in time $O(\log m)$.

This completes the proof of the theorem. □

4. Embeddings on high genus surfaces

The problem "given a graph G and an integer k, construct for G an embedding of genus k" in its general form is NP-hard [27]. In fact, even for planar graphs, it is unclear how an embedding of genus $k > 0$ can be constructed. In this section, we present an efficient algorithm that, given a graph G of n vertices and m edges, constructs for G an embedding of genus at least $\beta(G)/4$, where $\beta(G) = m - n + 1$ is the *cycle rank* of the graph G. Note that by our assumption that every vertex of G has degree at least 3, the cycle rank $\beta(G)$ is at least as large as $n/2 + 1$. We will use this result in the next section to show how an embedding of genus k is constructed for a large class of graphs and for a large range of integers k.

Let v be a vertex of degree d in a graph G. We say that we *split* the vertex v into two vertices v_1 and v_2 of degree $d' + 1$ and $d - d' + 1$, respectively, if we replace

the vertex v in the graph G by two adjacent new vertices v_1 and v_2 such that the vertex v_1 is of degree $d'+1$ and adjacent to v_2 and d' of the original neighbors of v, and that the vertex v_2 is of degree $d-d'+1$ and adjacent to v_1 and the rest $d-d'$ original neighbors of v. Note that a splitting operation on a vertex of a graph does not change the cycle rank of the graph. The inverse operation of vertex splitting is edge *contraction* which deletes an edge $e=\{u,v\}$ as well as the two vertices u and v and adds a new vertex w that is adjacent to all original neighbors of u and v.

The edge contraction operation can be extended to an embedding of the graph G. Let $\rho(G)$ be an embedding of the graph G such that the rotations at the two ends u and v of the edge e are

$$u: v, u_1, u_2, \ldots, u_s \quad \text{and} \quad v: u, v_1, v_2, \ldots, v_t$$

respectively. The contraction of the edge e in the embedding $\rho(G)$ is an embedding of the graph from the graph G by contracting the edge e in which every vertex has the same rotation as in $\rho(G)$ except the new vertex w, whose rotation is $w: u_1, \ldots, u_s, v_1, \ldots, v_t$. It is easy to see that edge contraction operation does not change the embedding genus.

The following definitions apply as well to disconnected graphs in which vertex degree may be less than 3. An *adjacency matching* in a graph H is a partition of edges of H into groups of one or two edges such that if two edges are in the same group then they have an endpoint in common. A *maximum adjacency matching* in H is an adjacency matching that maximizes the number of two-edge groups. A *maximal suspended chain* C in H is a simple path in H such that all interior vertices of C have degree 2 in H and both end-vertices of C have degree not equal to 2.

Let G be a 3-regular graph and H a spanning subgraph of G. For each degree 2 vertex v in H, the unique edge in $G-H$ that has v as an endpoint will be called "the edge associated with v".

The algorithm is presented in Fig. 1. The correctness of the algorithm is established based on the following lemmas.

Lemma 4.1. *The graph H_0 in Step 4 has a one-face embedding $\rho(H_0)$ whose genus is equal to the number of two-edge groups in the maximum adjacency matching $\mathcal{M}$.*

Proof. Let $Q=\{e_1, e_1', \ldots, e_q, e_q'\}$, where e_i and e_i' are in the same two-edge group in the matching $\mathcal{M}$. We start with an arbitrary embedding of the spanning tree T, which is a one-face embedding. We then inductively insert the two edges e_i and e_i' in each two-edge group in $\mathcal{M}$ into the one-face embedding such that the first edge e_i splits the face into two faces, and the second edge e_i' merges the two new faces into a single large face and increases the embedding genus by 1. This is always possible since after inserting the first edge e_i that splits the single face, the two sides of the edge e_i must be on the boundary of the two new faces. Thus, at each endpoint of the edge e_i, there are at least two face corners that belong to different faces. Now the second edge e_i' shares a common endpoint v_i with e_i. Thus, after an arbitrary insertion of the other endpoint

Algorithm 1.

Input: A graph G of n vertices and m edges.

Output: An embedding for G of genus at least $(m-n+1)/4$.

1. Use vertex splitting to convert the graph G into a 3-regular graph H;
2. Construct a spanning tree T of H;
3. Construct a maximum adjacency matching $\mathcal{M}$ in the co-tree $H-T$.
4. Let Q be the set of all edges that are in the two-edge groups in the matching $\mathcal{M}$. Construct a one-face embedding $\rho(H_0)$ for the graph $H_0 = T + Q$;
5. Let $\mathcal{L} = \{C_1, C_2, \ldots, C_h\}$ be the list of all maximal suspended chains in H_0;
6. **for** each maximal suspended chain C_i in the list $\mathcal{L}$

   case 1. C_i has at most two interior vertices.
   If there is a way to insert the edges associated with the interior vertices of C_i and increase the embedding genus, then insert the edges and increase the embedding genus. Let H_i and $\rho(H_i)$ be the resulting graph and embedding, respectively
   Otherwise, let $H_i = H_{i-1}$ and $\rho(H_i) = \rho(H_{i-1})$.
   case 2. C_i has more than two interior vertices.
   Pick the first three consecutive interior vertices v, u, and w on the chain C_i such that the vertex u is between the vertices v and w on the chain. Insert the edge associated with u. If inserting u does not increase the embedding genus, insert one of the edges associated with v and w to increase the embedding genus. Let H_i and $\rho(H_i)$ be the resulting graph and embedding, respectively.
   If any of the above edge insertion breaks a maximal suspended chain C_j, $i < j$, in the list $\mathcal{L}$, then replace the chain C_j in the list $\mathcal{L}$ by the two shorter chains and increase h by 1.
7. Arbitrarily insert the edges of H that are not used in Step 6. Let the resulting embedding of the graph H be $\rho(H)$.
8. In the embedding $\rho(H)$, contract those edges resulted from the vertex splitting operations in Step 1. The resulting embedding is an embedding for the graph G.

Fig. 1. Constructing an embedding on a high genus surface.

of e_i', we can always properly insert the endpoint v_i of e_i' so that the two endpoints of e_i' are inserted into face corners belonging to different faces. Consequently, the second edge e_i' merges the two new faces into a single large face and increases the embedding genus by 1. After the insertions of all the edges in the set Q in this way, we end up with a one-face embedding $\rho(H_0)$ of genus q for the graph $H_0 = T + Q$. □

Lemma 4.2. *The graph H_h constructed by Step 6 in Algorithm 1 is a spanning subgraph of H in which no vertex has degree 1. Moreover, for each degree 2 vertex in H_h, at most one of its neighbors is of degree 2.*

Proof. First observe that since the graph G is simple, the 3-regular graph H is also a simple graph. The graph H_h is a spanning subgraph of H because H_h contains $T+Q$ as a subgraph and T is a spanning tree of the graph H.

Suppose that v is a degree 1 vertex in H_h. The vertex v cannot have a selfloop because the graph H is simple. Let e_1 and e_2 be the two distinct edges in $H-H_h$ that share v as a common endpoint. Note that e_1 and e_2 are not contained in the set Q. Thus, the two-edge groups in $\mathcal{M}$ plus the pair $\{e_1,e_2\}$ would form a larger adjacency matching in the co-tree $H-T$, contradicting the definition of $\mathcal{M}$.

Finally, as done by **case 2** in Step 6, for every three consecutive interior vertices on a maximal suspended chain in the list $\mathcal{L}$, at least one associated edge is inserted, thus at least one of these three vertices has degree 3 in the graph H_h. Also note that each edge inserted in Step 6 is a maximal suspended chain without interior vertices. In conclusion, the graph H_h contains no maximal suspended chain with more than two interior vertices. □

We need to verify the validity of **case 2** in Step 6. Let e_v, e_u, and e_w be the three edges associated with the three vertices v, u, and w, respectively, in **case 2** of Step 6.

Lemma 4.3. *In* **case 2** *of Step 6 in Algorithm 1, either a proper insertion of the edge e_u increases the embedding genus, or an insertion of the edge e_u followed by a proper insertion of one of the edges e_v and e_w increases the embedding genus.*

Proof. Since C_i is a maximal suspended chain in the graph H_{i-1}, we can talk about the two "sides" of the chain C_i in the embedding $\rho(H_{i-1})$. If the two sides of the chain C_i belong to different faces in the embedding $\rho(H_{i-1})$, we can insert the edge e_u so that the embedding genus increases. On the other hand, if the two sides of the chain C_i belong to the same face of the embedding $\rho(H_{i-1})$ and inserting the edge e_u does not increase the embedding genus, then the edge e_u must split a face in the embedding $\rho(H_{i-1})$ into two new faces f_1 and f_2 such that one side of the chain C_i is split by e_u into two subwalks W_1 and W_2, where W_i belongs to the boundary walk of the new face f_i, $i=1,2$, and that each of the subwalks W_1 and W_2 contains one of the vertices v and w (note that the other endpoint of the edge e_u can be neither v nor w because the graph H is a simple graph). See Fig. 2. On the other hand, the

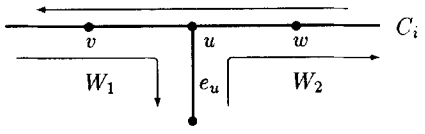

Fig. 2. Inserting the edge e_u splits a face.

entire other side of the chain C_i still belongs to the same face in the new embedding. Therefore, for at least one of the vertices v and w, its two corners in the new embedding belong to different faces. Thus, we can insert the associated edge for this vertex properly to increase the embedding genus. □

Lemma 4.4. *The embedding $\rho(H)$ of the graph H constructed by Step 7 of Algorithm 1 has genus at least $\beta(H)/8$.*

Proof. Suppose that the graph H has n_0 vertices. We count how many edges are not inserted in Step 6. By Lemma 4.2, the graph H_h is a spanning subgraph of H with no degree 1 vertices.

Since the graph H is 3-regular, every vertex in the graph H_h has degree either 2 or 3. Let n_2 and n_3 be the number of vertices of degree 2 and 3, respectively, of the graph H_h. Then $n_2 + n_3 = n_0$. We say that a degree 2 vertex v in H_h is *covered* by a degree 3 vertex w in H_h if v and w are adjacent in H_h. By Lemma 4.2, in the graph H_h, each degree 2 vertex is adjacent to at most one degree 2 vertex. Thus, every degree 2 vertex in H_h is covered by at least one degree 3 vertex. Since each degree 3 vertex in the graph H_h can cover at most three degree 2 vertices, we have $n_3 \geqslant n_2/3$. Thus, $n_0 \geqslant n_2 + n_2/3$. This gives $3n_0/4 \geqslant n_2$. Since each edge in $H - H_h$ is associated with exactly two degree 2 vertices in the graph H_h, we conclude that the number of edges in $H - H_h$ is bounded by $3n_0/8$. In other words, the cycle rank $\beta(H_h)$ of the graph H_h is at least $\beta(H) - 3n_0/8$.

By Lemma 4.1, the genus of the embedding $\rho(H_0)$ is equal to the number of two-edge groups in the matching $\mathcal{M}$. Moreover, according to Step 6 of Algorithm 1, the embedding genus is increased by at least 1 by inserting at most two edges in $H - H_0$. Therefore, the genus of the embedding $\rho(H_h)$ is at least half of the cycle rank $\beta(H_h)$.

Since the embedding $\rho(H)$ constructed by Step 7 of Algorithm 1 has genus at least as large as that of the embedding $\rho(H_h)$, we conclude that the genus of the embedding $\rho(H)$ is at least $\beta(H_h)/2 \geqslant \beta(H)/2 - 3n_0/16$. Since the graph H is 3-regular, $\beta(H) = n_0/2 + 1$. Thus, the embedding $\rho(H)$ has genus at least $n_0/16 + \frac{1}{2}$, which is larger than $\beta(H)/8$. □

The above analysis enables us to derive the following theorem, which claims that given a graph, an embedding of high genus for the graph can always be constructed efficiently. A number of applications of this theorem will be demonstrated in the next section.

Theorem 4.5. *Algorithm 1 runs in time $O(m \log n)$ and constructs for a given graph G an embedding of genus at least $\beta(G)/8$.*

Proof. Since vertex splitting does not change the cycle rank of a graph, the 3-regular graph H constructed in Step 1 has the same cycle rank as the graph G. Moreover, since edge contraction on an embedding does not change the embedding genus, the

embedding for the graph G constructed in Step 8 of Algorithm 1 has the same genus as the embedding $\rho(H)$. By Lemma 4.4, the embedding of G constructed by Algorithm 1 has genus at least $\beta(G)/8$.

Now we study the complexity of Algorithm 1. First note that the number of edges in the graph H is bounded by $O(m)$, where m is the number of edges in the graph G. Steps 1 and 2 can be easily done in time $O(m)$ under any reasonable data structure for graphs. To construct a maximum adjacency matching in the co-tree $H - T$, note that since H is 3-regular, each connected component in $H - T$ is either a simple cycle or a simple path. Moreover, it is easy to see that an adjacency matching in $H - T$ is maximum if and only if it leaves at most one edge in a one-edge group for each connected component in $H - T$. Thus, the maximum adjacency matching M can be constructed in linear time.

To construct the embedding given by Step 7, we start with an arbitrary embedding of the spanning tree T, which can be constructed in linear time. Now insertion and deletion of an edge in Steps 4–7 takes time $O(\log n)$ using the data structure DLFL introduced in Section 3. Also note that in Step 6, there are only four possible ways to insert an associated edge since each endpoint of the edge has degree 2 in the current embedding. Therefore, the conditions in Step 6 can be tested in $O(\log n)$ time. In conclusion, each associated edge can be checked by **case 1** at most twice and each checking takes time $O(\log n)$, and **case 2** inserts at least one associated edge in time $O(\log n)$ for each execution of the **for** loop body in Step 6.

The edge contraction in Step 8 is particularly easy if we adopt the data structure DLFL: we only need to traverse all faces in the embedding and contract the edges resulted from Step 1 on the boundary of each face.

This completes the proof that the time complexity of Algorithm 1 is bounded by $O(m \log n)$. □

Algorithm 1 constructs an embedding for a graph on a high genus surface. In fact, this embedding is a good approximation of a maximum genus embedding of the graph.

Corollary 4.6. *There is an $O(m \log n)$ time algorithm that constructs an embedding for a graph G such that the embedding genus is at least $\frac{1}{4}$ of the maximum genus of the graph G.*

Proof. By Euler's formula, the maximum genus of a graph G is bounded by $\beta(G)/2$. □

5. Embeddings on the surface of genus k

Using the new data structure introduced in Section 3 and the efficient algorithm developed in Section 4, we will show in this section that, in contrast with the general NP-hardness of the graph minimum genus problem, for a large range of integers k

Algorithm 2.

Input: Embeddings $\rho_1(G)$ and $\rho_2(G)$ and an integer γ between γ_1 and γ_2, where γ_1 and γ_2 are genera of $\rho_1(G)$ and $\rho_2(G)$, respectively.

Output: An embedding $\rho(G)$ of G of genus γ.

1. If $\gamma < \min\{\gamma_1, \gamma_2\}$ or $\gamma > \max\{\gamma_1, \gamma_2\}$, Stop.
2. If $\gamma = \gamma_1$ then output $\rho_1(G)$ and Stop
   Otherwise, let $\rho(G) = \rho_1(G)$;
3. **for** each vertex v of the graph G

   Let the rotation at v in the embedding $\rho_2(G)$ be

   $$v: u_1, u_2, \ldots, u_r.$$

   for $i = 2$ **to** r **do**

   Delete the edge $e_i = \{v, u_i\}$ from the embedding $\rho(G)$, then reinsert the edge e_i into the embedding so that the edge e_i follows the edge e_{i-1} in the rotation at the vertex v, and the position of the other endpoint of e_i is unchanged.
   If γ equals the genus of the current embedding $\rho(G)$, output $\rho(G)$ and Stop.

Fig. 3. Moving from one embedding to another embedding.

and for a large class of graphs G, the problem "given a graph G and an integer k, construct a genus k embedding for G" can be solved efficiently.

We start with a very useful algorithm that efficiently moves continuously from one embedding of a graph to another embedding of the same graph. This algorithm will be crucial in many cases for finding graph embeddings on a surface of a given genus.

Theorem 5.1. *There is an* $O(m \log n)$ *time algorithm that, given two embeddings* $\rho_1(G)$ *and* $\rho_2(G)$ *of a graph* G *of* m *edges and* n *vertices and given an integer* γ *such that* γ *is between the genera of* $\rho_1(G)$ *and* $\rho_2(G)$, *constructs an embedding of genus* γ *for the graph* G.

Proof. Consider the algorithm given in Fig. 3.

Since each edge deletion on an embedding can decrease the embedding genus by at most 1 and never increase the embedding genus, and since each edge insertion to an embedding can increase the embedding genus by at most 1 and never decrease the embedding genus, each execution of the body of the inner **for** loop in Algorithm 2 can change the embedding genus by at most 1. Therefore, Algorithm 2 must stop at some point and end up with an embedding $\rho(G)$ of G of genus exact γ if γ is between the genera γ_1 and γ_2 of the embeddings $\rho_1(G)$ and $\rho_2(G)$, respectively.

Note that if the data structure DLFL introduced in Section 3 is used for the embeddings $\rho_1(G)$ and $\rho_2(G)$, then the genera γ_1 and γ_2 can be computed in constant time.

Algorithm 2 consists of traversing the rotation at each vertex in the embedding $\rho_2(G)$ and rearranging the rotation at each vertex in the embedding $\rho_1(G)$.

The traversing of the rotation at a vertex takes time proportional to the degree of the vertex, if the data structure DLFL is used. Thus, the total time spent by Algorithm 2 on vertex rotation traversing is bounded by $O(m)$, where m is the number of edges of the graph G.

Each edge endpoint at a vertex in the embedding $\rho_1(G)$ is rearranged by an edge deletion followed by an edge insertion. Therefore, the number of edge insertions and edge deletions performed on each vertex of G is proportioned to the degree of the vertex. Thus, the total number of edge insertions and edge deletions performed by Algorithm 2 is proportional to the number m of edges of the graph G. Using the data structure DLFL, each such operation takes time $O(\log n)$.

We conclude that the total running time of Algorithm 2 is bounded by $O(m \log n)$. □

Corollary 5.2. *There is an algorithm such that given two embeddings $\rho_1(G)$ and $\rho_2(G)$ of genus γ_1 and γ_2, respectively, for a graph $G, \gamma_1 \leqslant \gamma_2$, the algorithm constructs in time $O(\max\{|\gamma_1-\gamma_2|m, m \log n\})$ $\gamma_2-\gamma_1+1$ embeddings $\rho_{\gamma_1}(G)$, $\rho_{\gamma_1+1}(G), \ldots, \rho_{\gamma_2}(G)$ for the graph G, such that the genus of the embedding $\rho_i(G)$ is i, for $\gamma_1 \leqslant i \leqslant \gamma_2$.*

Proof. According to Theorem 5.1, moving from embedding $\rho_1(G)$ to embedding $\rho_2(G)$ takes time $O(m \log n)$. Printing out an embedding for the graph G takes time $O(m)$. □

The algorithm in Corollary 5.2 is optimal when $|\gamma_2 - \gamma_1| = \Omega(\log n)$ since printing out a single embedding of the graph G would take time at least $\Omega(m)$.

Theorem 5.3. *There is an $O(n \log n)$ time algorithm such that given a planar graph G and an integer $k \leqslant \beta(G)/8$, the algorithm constructs an embedding of genus k for the graph G.*

Proof. The algorithm first constructs a planar embedding for the graph G in linear time [21], then constructs an embedding of genus at least $\beta(G)/8$ for the graph G in time $O(m \log n) = O(n \log n)$, according to Theorem 4.5. Now the theorem follows directly from Theorem 5.1. □

If both minimum genus embedding and maximum genus embedding of a graph can be constructed efficiently, then so is its any "intermediate genus" embedding.

Theorem 5.4. *Let $\mathscr{C}$ be a class of graphs whose minimum genus embedding and maximum genus embedding can be constructed in time $O(m \log n)$, then the problem "given a graph G and an integer k, construct a genus k embedding for G" can be solved in time $O(m \log n)$ for the class $\mathscr{C}$.*

Theorem 5.4 covers many important graph classes, in particular many graph classes studied in the interconnection networks, such as complete graphs, hypercubes, star graphs, and all 4-connected planar graphs. (For a comprehensive discussion of these graph classes, we refer our readers to Leighton's authoritative book [23].)

Theorem 5.5. *Let $\mathscr{D}$ be a class of graphs whose minimum genus embedding can be constructed in polynomial time, then the problem "given a graph G and an integer k, construct a genus k embedding for G" can be solved in polynomial time for the class $\mathscr{D}$.*

Proof. This is because that a maximum genus embedding of a graph G can be constructed in time $O(n^4 \log^6 n)$ [17]. □

Theorem 5.5 covers in particular the class of graphs whose minimum genus is bounded by a fixed constant [12, 14].

Now we study the graph embeddings that are related to the "average genus" of the graphs. Each graph G is associated with a sequence of integers $g_0, g_1, g_2, \ldots$, called the *genus distribution* of G, where g_i is the number of embeddings of genus i for the graph G. The *average genus* of G is defined to be the value

$$\gamma_{\text{avg}}(G) = \frac{\sum_{i=0} i \cdot g_i}{\sum_{i=0} g_i}.$$

Thus, an embedding of genus larger than $\gamma_{\text{avg}}(G)$ can be regarded as an "upper genus embedding" while an embedding of genus smaller than $\gamma_{\text{avg}}(G)$ can be regarded as a "lower genus embedding". The average genus of a graph plays an important role in the recent study of topological invariants of graphs [2, 5–7, 19].

Intuitively, the average genus of a graph can be computed by probabilistic sampling. In the following, we will discuss how this idea can be precisely formulated. We first prove two lemmas. We will denote by $\Gamma_G = \sum_{i-0} g_i$ the total number of embeddings of the graph G.

Lemma 5.6. *For any real number $\varepsilon > 0$, there are at least $(\varepsilon/(1+\varepsilon))\Gamma_G$ embeddings of the graph G that are of genus $\leqslant (1+\varepsilon)\gamma_{\text{avg}}(G)$.*

Proof. Assume the contrary. Then there are more than $\Gamma_G - (\varepsilon/(1+\varepsilon))\Gamma_G = (1/(1+\varepsilon))\Gamma_G$ embeddings of G of genus larger than $(1+\varepsilon)\gamma_{\text{avg}}(G)$. Thus

$$\gamma_{\text{avg}}(G) \geqslant \frac{\sum_{i>(1+\varepsilon)\gamma_{\text{avg}}(G)} i \cdot g_i}{\Gamma_G} > \frac{(1+\varepsilon)\gamma_{\text{avg}}(G)\Gamma_G/(1+\varepsilon)}{\Gamma_G} = \gamma_{\text{avg}}(G).$$

This contradiction proves the lemma. □

Lemma 5.7. *For any real number $\varepsilon > 0$, there are at least $(\varepsilon/4)\Gamma_G$ embeddings of the graph G that are of genus $\geqslant (1-\varepsilon)\gamma_{\text{avg}}(G)$.*

Proof. Assume the contrary. Then there are less than $(\varepsilon/4)\Gamma_G$ embeddings of G of genus $\geqslant(1-\varepsilon)\gamma_{\text{avg}}(G)$. Let $\gamma_{\max}(G)$ be the maximum genus of the graph G. Recently, Chen et al. [9] have proved that $\gamma_{\max}(G) < 4\gamma_{\text{avg}}(G)$. We have

$$\begin{aligned}\gamma_{\text{avg}}(G) &= \frac{\sum_{i<(1-\varepsilon)\gamma_{\text{avg}}(G)} i \cdot g_i + \sum_{i\geqslant(1-\varepsilon)\gamma_{\text{avg}}(G)} i \cdot g_i}{\Gamma_G} \\ &\leqslant \frac{(1-\varepsilon)\gamma_{\text{avg}}(G)\sum_{i<(1-\varepsilon)\gamma_{\text{avg}}(G)} g_i + \gamma_{\max}\sum_{i\geqslant(1-\varepsilon)\gamma_{\text{avg}}(G)} g_i}{\Gamma_G} \\ &< \frac{(1-\varepsilon)\gamma_{\text{avg}}(G)\Gamma_G + 4\gamma_{\text{avg}}(G)(\varepsilon/4)\Gamma_G}{\Gamma_G} \\ &= \gamma_{\text{avg}}(G).\end{aligned}$$

This contradiction proves the lemma. □

We present two randomized algorithms with small error probability that construct graph embeddings related to graph average genus. We will use *round*$(\gamma_{\text{avg}}(G))$ to denote the unique integer in the semi-open interval $(\gamma_{\text{avg}}(G) - 0.5, \gamma_{\text{avg}}(G) + 0.5]$ on the real line.

Theorem 5.8. *For any real number $\delta > 0$, there is a time $O(m^2)$ randomized algorithm such that given a graph G and an integer k, the algorithm either reports $k \neq$ round$(\gamma_{\text{avg}}(G))$ with error probability less than δ, or constructs an embedding of genus k for the graph G.*

Proof. Let e be the base of natural logarithm. Let c be a constant such that $e^{-c} < \delta/2$.

Let m be the number of edges of the graph G. By Lemma 5.6, there are at least $[(1/m)/(1+1/m)]\Gamma_G = [1/(m+1)]\Gamma_G$ embeddings of the graph G that have genus $\leqslant(1+1/m)\gamma_{\text{avg}}(G)$. Therefore, if we randomly pick $4cm$ embeddings of G, the probability that all these embeddings are of genus larger than $(1+1/m)\gamma_{\text{avg}}(G)$ is bounded by

$$(1-1/(m+1))^{4cm} \leqslant [(1-1/(m+1))^{m+1}]^c < e^{-c} < \delta/2.$$

By Lemma 5.7, there are at least $[1/(4m)]\Gamma_G$ embeddings of the graph G that have genus $\geqslant(1-1/m)\gamma_{\text{avg}}(G)$. Thus, if we randomly pick $4cm$ embeddings of G, the probability that all these embeddings are of genus less than $(1-1/m)\gamma_{\text{avg}}(G)$ is bounded by

$$(1-1/4m)^{4cm} \leqslant e^{-c} < \delta/2.$$

Therefore, if we randomly pick $4cm$ embeddings of the graph G, with probability larger than $1-\delta$, we will have one embedding $\rho_1(G)$ of genus $\gamma_1 \leqslant (1+1/m)\gamma_{\text{avg}}(G)$

and one embedding $\rho_2(G)$ of genus $\gamma_2 \geqslant (1 - 1/m)\gamma_{\text{avg}}(G)$. Note that $\gamma_{\text{avg}}(G) \leqslant (m - n + 1)/2 < m/2$.[2] Therefore, $\gamma_1 < \gamma_{\text{avg}}(G) + 0.5$ and $\gamma_2 > \gamma_{\text{avg}}(G) - 0.5$. Since both γ_1 and γ_2 are integers, we must have $\gamma_1 \leqslant round(\gamma_{\text{avg}}(G))$ and $\gamma_2 \geqslant round(\gamma_{\text{avg}}(G))$.

In conclusion, if $k = round(\gamma_{\text{avg}}(G))$, then with probability larger than $1 - \delta$, we will get two embeddings $\rho_1(G)$ and $\rho_2(G)$ from a random sample set of $4cm$ embeddings such that the genus of $\rho_1(G)$ is not larger than k and the genus of $\rho_2(G)$ is not smaller than k. Now a genus k embedding of the graph G can be constructed by applying Theorem 5.1. The algorithm obviously runs in time $O(m^2)$. □

With a similar argument, we can also show that constructing an upper genus embedding for a graph, i.e., an embedding of genus at least as large as its average genus, is feasible if we allow a small error probability.

Theorem 5.9. *For any real number $\delta > 0$, there is a polynomial time randomized algorithm such that given a graph G and an integer k, the algorithm either reports $k < round(\gamma_{\text{avg}}(G))$ with error probability less than δ, or constructs an embedding of genus k for the graph G.*

Proof. Similar to the analysis in Theorem 5.8, we first construct an embedding $\rho(G)$ of genus bounded by $round(\gamma_{\text{avg}}(G))$ from a random sample set of $O(m)$ embeddings. We also construct a maximum genus embedding $\rho'(G)$ for the graph G in polynomial time [17]. Now we apply Theorem 5.1. □

The error probability δ in Theorem 5.9 can actually be bounded by $\delta = O(2^{-m^d})$ for any constant d. In fact, if instead of using a random sample set of $O(m)$ embeddings, we use a random sample set of $(m + 1)^{d+1}$ embeddings, then the error probability is bounded by

$$(1 - 1/(m + 1))^{(m+1)^{d+1}} < e^{-(m+1)^d} < 2^{-m^d}.$$

6. Concluding remarks

We have developed a new data structure for graph embeddings. The data structure is superior to the existing data structures and efficiently supports all on-line operations for graph embeddings. We have shown a number of applications of the new data structure, including an $O(m \log n)$ time algorithm for constructing embeddings of graphs on surfaces of high genus, and an $O(m \log n)$ time algorithm for continuously moving from one embedding of a graph to another embedding of the same graph. We have demonstrated efficient algorithms for constructing all kinds of embeddings for planar graphs. Efficient randomized algorithms have also been developed to construct graph embeddings on surfaces of genus larger than the average genus of the graph,

[2] Without loss of generality, we assume that the number n of vertices of the graph is at least 2.

and embeddings on surfaces of genus equal to the average genus of the graph. Our computational results indicate that there is a very interesting and rich computational structure related to graph embeddings, which definitely deserves further investigation.

Our results show that construction of an embedding of a graph on high genus surfaces is in general computationally feasible. A closely related problem is the complexity of constructing low genus embeddings for a graph. It is unknown whether there is a polynomial time algorithm that approximates the minimum genus embedding of a graph to a constant ratio. A recent result by Chen et al. [10] partially hints the difficulty of approximating graph minimum genus embeddings: for any real number ε, $0 \leqslant \varepsilon < 1$, the problem of embedding a graph G of n vertices into a surface of genus $\gamma_{\min}(G) + n^{\varepsilon}$ is NP-hard.

Acknowledgements

The author would like to thank Professor Jonathan Gross and Professor Ming Li for their encouragement and comments. He is also grateful to Dr. Saroja Kanchi for her comments and constructive discussions. Finally, the author thanks an anonymous referee for comments and corrections that have improved the presentation.

References

[1] A.V. Aho, J.E. Hopcroft and J.D. Ullman, *The Design and Analysis of Computer Algorithms* (Addison-Wesley, Reading, MA, 1974).

[2] J. Chen, A linear time algorithm for isomorphism of graphs of bounded average genus, *SIAM J. Discrete Math.* **7** (1994) 614–631.

[3] J. Chen, Algorithmic graph embeddings, Lecture Notes in Computer Science, Vol. 959 (Springer, New York, 1995) 151–160.

[4] J. Chen, D. Archdeacon and J.L. Gross, Maximum genus and connectivity, *Discrete Math.* **149** (1996) 19–29.

[5] J. Chen and J.L. Gross, Limit points for average genus I. 3-connected and 2-connected simplicial graphs, *J. Combin. Theory Ser. B* **55** (1992) 83–103.

[6] J. Chen and J.L. Gross, Limit points for average genus II. 2-connected non-simplicial graphs, *J. Combin. Theory Ser. B* **56** (1992) 108–129.

[7] J. Chen and J.L. Gross, Kuratowski-type theorems for average genus, *J. Combin. Theory Ser. B* **57** (1993) 100–121.

[8] J. Chen and S.P. Kanchi, Graph embeddings and graph ear decompositions, Lecture Notes in Computer Science, Vol. 790 (Springer, New York, 1994) 376–387.

[9] J. Chen, S.P. Kanchi and J.L. Gross, A tight lower bound on the maximum genus of a simplicial graph, *Discrete Math.* **156** (1996) 83–102.

[10] J. Chen, S.P. Kanchi and A. Kanevsky, On the complexity of graph embeddings, Lecture Notes in Computer Science, Vol. 709 (Springer, New York, 1993) 234–245; Journal version "A note on approximating graph genus" to appear in Information Processing Letters.

[11] G. Di Battista, P. Eades and R. Tamassia, Algorithms for automatic graph drawing: an annotated bibliography, Tech. Report, Dept. Computer Science, Brown University, 1993.

[12] H. Djidjev and J. Reif, An efficient algorithm for the genus problem with explicit construction of forbidden subgraphs, in: *Proc. 23rd Ann. ACM Symp. on Theory of Computing* (1991) 337–347.

[13] I.S. Filotti, An algorithm for imbedding cubic graphs in the torus, *J. Comput. System Sci.* **20** (1980) 255–276.

[14] I.S. Filotti, G.L. Miller and J.H. Reif, On determining the genus of a graph in $O(v^{O(g)})$ steps, in: *Proc. 11th Ann. ACM Symp. on Theory of Computing* (1979) 27–37.
[15] G.N. Frederickson, Using cellular graph embeddings in solving all pairs shortest paths problems, *J. Algorithms* **19** (1995) 45–85.
[16] G.N. Frederickson and R. Janardan, Designing networks with compact routing tables, *Algorithmica* **3** (1988) 171–190.
[17] M.L. Furst, J.L. Gross and L.A. McGeoch, Finding a maximum-genus graph imbedding, *J. ACM* **35**(3) (1988) 523–534.
[18] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of* NP-*Completeness* (Freeman, New York, 1979).
[19] J.L. Gross and M.L. Furst, Hierarchy for imbedding-distribution invariants of a graph, *J. Graph Theory* **11** (1987) 205–220.
[20] J.L. Gross and T.W. Tucker, *Topological Graph Theory* (Wiley-Interscience, New York, 1987).
[21] J.E. Hopcroft and R.E. Tarjan, Efficient planarity testing, *J. ACM* **21** (1974) 549–568.
[22] G.F. Italiano, J.A. La Poutre and M.H. Rauch, Fully dynamic planarity testing in planar embedded graphs, Lecture Notes in Computer Science, Vol. 726 (Springer, New York, 1993) 576–590.
[23] F.T. Leighton, *Introduction to Parallel Algorithms and Architectures* (Morgan Kaufmann, Los Altos, CA, 1992).
[24] B. Mohar, Projective planarity in linear time, *J. Algorithms* **15** (1993) 482–502.
[25] F.P. Preparata and M.I. Shamos, *Computational Geometry: An Introduction* (Springer, Berlin, 1985).
[26] R. Tamassia, A dynamic data structure for planar graph embedding, Lecture Notes in Computer Science, Vol. 317 (Springer, New York, 1988) 576–590.
[27] C. Thomassen, The graph genus problem is NP-complete, *J. Algorithms* **10** (1989) 568–576.