International Journal of Networking and Computing – www.ijnc.org
ISSN 2185-2839 (print) ISSN 2185-2847 (online)
Volume 6, Number 2, pages 149–166, July 2016

A Practical Algorithm for Embedding Graphs on Torus[1]

Jiahua Yu
School of Computing Science, Simon Fraser University
Burnaby, BC, V5A 1S6, Canada

Qian-Ping Gu
School of Computing Science, Simon Fraser University
Burnaby, BC, V5A 1S6, Canada

Received: February 15, 2016
Accepted: April 4, 2016
Communicated by Akihiro Fujiwara

Abstract

Embedding graphs on the torus is a problem with both theoretical and practical importance. It is required to embed a graph on the torus for solving many application problems such as VLSI design, graph drawing and so on. Polynomial time and exponential time algorithms for embedding graphs on the torus are known. However, the polynomial time algorithms are very complex and their implementation has been a challenge for a long time. On the other hand, the implementations of some exponential time algorithms are known but they are not efficient for large graphs in practice. To develop an efficient practical tool for embedding graphs on the torus, we propose a new exponential time algorithm for embedding graphs on the torus. Compared with a well used previous exponential time algorithm, our algorithm has a better practical running time.

Keywords: Graph algorithms, embedding graphs on surfaces, torus, computational study

1 Introduction

Embedding a graph G on a surface is to draw each vertex of G as a point and each edge between two vertices as a curve connecting the points of the two vertices on the surface. Graph G is embeddable on a surface if G can be drawn on the surface in such a way that no two edges cross. For example, a graph embeddable on a sphere (plane) is called a planar graph. Embedding graphs on surfaces is an important field in graph theory and has numerous applications such as VLSI design, graph drawing, visualizing relations and structural properties among data, and so on. Algorithms for embedding graphs on surfaces have been extensively studied. The simplest surface in the study of graph embedding is the sphere which has orientable genus zero. Linear time algorithms have been developed and implemented for embedding graphs on the sphere [5, 7, 12]. These algorithms and their implementations are widely used in many applications such as the libraries for planar

[1] A preliminary version of this paper appeared in the Proc. of the 3rd International Symposium on Computing and Networking (CANDAR2015) [28].

graphs [1, 2]. Much effort has been made for embedding graphs on surfaces of genus greater than zero. Linear time algorithms for embedding graphs on surfaces of bounded genus have been known [15, 17, 18]. These algorithms, however, are too complex to be implemented and are mainly of theoretical interest. The problem of whether a graph is embeddable on a surface of genus k or not is NP-complete when k is part of the input [25, 26].

Developing efficient tools for embedding graphs on surfaces is of great interest in practice. A projective plane is the surface of non-orientable genus one. A linear time algorithm for embedding graphs on the projective plane is known [16]. This algorithm is simplified with a trade off that increases the running time to $O(n^2)$ [20]. The simplified algorithm has been implemented and seems the first polynomial time algorithm implemented for embedding graphs on a surface other than the sphere.

A torus is the surface of orientable genus one. Embedding graphs on the torus is a more prevailing problem for which much work is in progress and an efficient implementation is promising. A linear time algorithm for embedding graphs on the torus is proposed [13] and a simplified version with $O(n^3)$ running time is introduced [14]. Although an implementation of the $O(n^3)$ algorithm was announced [3], it does not seem working [19, 24]. To our best knowledge, a working implementation of polynomial time algorithm for embedding graphs on the torus is not known and available tools for embedding graphs on the torus are the implementations of exponential time algorithms [22, 27].

All previous algorithms for embedding graphs on the torus follow a common approach called *embedding extension* [18]: Given a graph G, the planarity of G is first checked. If G is planar, a planar embedding of G is used as an embedding of G on the torus. For a nonplanar G, a subgraph called *frame* of G is embedded on the torus such that the embedding of the frame contains a non-contractible cycle on the torus (if a nonplanar G is embeddable on the torus then any embedding of G on the torus has a non-contractible cycle), and then the remaining parts of G are attached to the embedding of the frame.

Neufeld and Myrvold introduce an algorithm (NM Algorithm) which chooses a cycle in a nonplanar G as the frame [22]. Graph G is nonplanar if it has a subgraph *homeomorphic* to K_5 (complete graph of five vertices) or $K_{3,3}$ (complete bipartite graph of six vertices, three of which connect to each of the other three) which are called *obstructions* for the sphere (definitions of homeomorphic and obstructions for surfaces are given in the next section). Subgraphs homeomorphic to K_5 or $K_{3,3}$ can be used as frames for embedding graphs on the torus as well [18]. Based on this, Woodcock gives an algorithm which uses such a subgraph as the frame and finds all *non-equivalent embeddings* (definition in the next section) of the frame on the torus [27]. For each embedding of the frames in NM Algorithm and Woodcock's algorithm, there are exponentially many attachments (the ways to attach the remaining parts of G) to the embedding of the frame.

It is observed that a more complex frame places more restrictions on the embedding extension, resulting in a smaller number of attachments to the frame. A trade off is that a more complex frame has more non-equivalent embeddings on the torus to be checked, resulting in a more complex implementation. Juvan and Mohar propose an algorithm (JM Algorithm) which uses more complex frames [14]. For a nonplanar graph G, JM Algorithm further checks if G is embeddable on the projective plane. For G embeddable on the projective plane, JM Algorithm finds an embedding of G on the torus or concludes G not embeddable on the torus. For G not embeddable on the projective plane, JM Algorithm finds a subgraph K homeomorphic to an obstruction for the projective plane as the frame, finds all non-equivalent embeddings $\Pi(K)$ of K on the torus and for each $\Pi(K)$, tries all attachments to find an embedding of G or concludes G not embeddable on the torus. JM Algorithm further reduces the number of attachments to each $\Pi(K)$ from exponential many to polynomial many by complex techniques which are challenging to implement.

In this paper, we propose a new exponential time algorithm for embedding graphs on the torus. Similar to JM Algorithm, our algorithm follows the embedding extension approach and uses a subgraph homeomorphic to an obstruction for the projective plane as the frame. However, we avoid the complex techniques for reducing the number of attachments and use a simple approach for the attachment part. Our approach for the attachment may run in exponential time in the worst case but is efficient in practice and reduces the implementation complexity significantly. By the above, we have a balanced point between the running time and the implementation complexity for our algorithm.

A major challenge to use a subgraph K homeomorphic to an obstruction for the projective plane as the frame is to find all non-equivalent embeddings of K on the torus because there can be hundreds such embeddings for K homeomorphic to one obstruction and there are 103 obstructions. We clear this hurdle by a simple algorithm for computing the embeddings. New ingredients in our algorithm also include pre-processing steps to find some graphs whose embeddability on the torus can be computed without using the attachment part, and prune schemes to reduce the running time for the attachment part. Computational results show that our algorithm has a better practical running time than a well used previous exponential time algorithm. Our implementation can also be used as a base to develop an implementation for JM Algorithm.

The rest of this paper is organized as follows. Next section gives the preliminaries of the paper. In Section 3, we review the techniques for embedding graphs on the projective plane and torus on which our algorithm builds. Our new algorithm is described in Section 4. Computational results are reported in Section 5 and the final section concludes the paper.

2 Preliminaries

A graph G in this paper is undirected and consists of a set $V(G)$ of vertices and a set $E(G)$ of edges. For a set $A \subseteq E(G)$ of edges, let $V(A)$ be the set of vertices incident to an edge of A. We denote by $\deg_G(v)$ the *node degree* of v. A graph H is a subgraph of G if $V(H) \subseteq V(G)$ and $E(H) \subseteq E(G)$. For $A \subseteq E(G)$ and $W \subseteq V(G)$, we denote by $G[A]$ and $G[W]$ the subgraphs of G induced by the subset A of edges and subset W of vertices, respectively. Further, for a subgraph H of G, we denote by $G \setminus H$ the subgraph of G induced by the edge set $E(G) \setminus E(H)$.

A *walk* of a graph G is a sequence $e_1, ..., e_k$ of edges, where $e_i = \{v_{i-1}, v_i\} \in E(G)$ for $1 \leq i \leq k$. A walk is closed if $v_0 = v_k$. A walk is called a *path* if no vertex is repeated in the sequence. A closed walk is called a *cycle* if no vertex is repeated in the sequence except $v_0 = v_k$. Graph G is connected if there is a path between any two vertices in G. Graph G is k-connected if it remains connected after removing any $k-1$ vertices from G. A *subdivision* of edge $e = \{u, v\}$ in G is to replace e with a new vertex w and two edges $\{u, w\}$ and $\{w, v\}$. A graph H is a *subdivision* of G if it can be obtained from G with a series of edge subdivisions. Two graphs G and G' are *homeomorphic* if they have a common subdivision H.

The main reference for graphs on surfaces in this paper is the monograph by Mohar and Thomassen [21]. A surface (without boundary) is a connected Hausdorff space in which every point has an open neighborhood homeomorphic to the open subset $\{(x, y)|0 < x, y < 1\}$ of the Euclidean plane $\mathbb{E}^2$. A surface is *orientable* if a consistent clockwise rotation can be defined around every point of the surface, otherwise *non-orientable*. The sphere is an orientable surface. An open disc in a surface is an open subset of the surface homeomorphic to the open subset $\{(x, y)|0 < x, y < 1\}$ of $\mathbb{E}^2$. For an open subset r of a surface, we denote by $\overline{r}$ the closure of r and $\mathrm{bd}(r) = \overline{r} \setminus r$ the boundary of r. To add a *handle* to a sphere, we remove two disjoint open discs D_1 and D_2 from the sphere, and identify $\mathrm{bd}(D_1)$ with $\mathrm{bd}(D_2)$ such that the clockwise orientations around $\mathrm{bd}(D_1)$ and $\mathrm{bd}(D_2)$ disagree. The *genus* of a sphere is defined zero. An orientable surface of genus h is obtained by adding h handles to the sphere. The torus (see Figure 1 (a)) is obtained by adding one handle to the sphere. To add a *crosscap* to a sphere, we remove one open disc D homeomorphic to $\{(x, y)|0 < x, y < 1\}$ from the sphere, identify every point $(x, 0)$ of $\mathrm{bd}(D)$ with the point $(1 - x, 1)$ of $\mathrm{bd}(D)$, $0 \leq x \leq 1$, and identify every point $(0, y)$ of $\mathrm{bd}(D)$ with the point $(1, 1 - y)$ of $\mathrm{bd}(D)$, $0 \leq y \leq 1$. A non-orientable surface of non-orientable genus l is obtained by adding l crosscaps to the sphere. The projective plane, denoted by $\mathbb{N}_1$, is made by adding one crosscap to the sphere.

A graph G is embedded on a surface Σ if vertices of $V(G)$ are mapped to distinct points in Σ and every edge $e = \{u, v\}$ is mapped to a curve in Σ with end points u and v such that distinct edges do not intersect except at the shared end points. The collection of such points and curves is called an embedding, denoted by $\Pi(G)$, of G. We may use $v \in V(G)$ both for vertex v of G and for the point in $\Pi(G)$ corresponding to v, and use $e = \{u, v\} \in E(G)$ both for edge e of G and for the curve in $\Pi(G)$ corresponding to e.

Graph G is *embeddable* on a surface if there is an embedding of G on the surface, otherwise

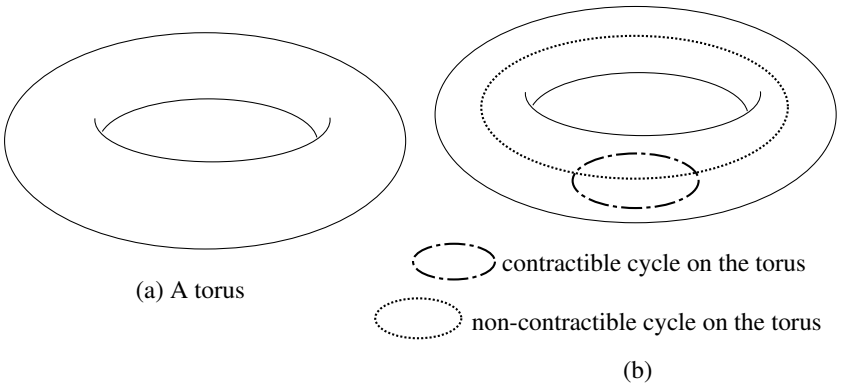

Figure 1: A torus and cycles on the torus.

non-embeddable. The *genus* $g(G)$ and *non-orientable genus* $\tilde{g}(G)$ of a graph G are the minimum h and l such that G is embeddable on the orientable surface of genus h and non-orientable genus l, respectively. A graph G is a (topological) *obstruction* for Σ if $\deg_G(v) > 2$ for every $v \in V(G)$, G is not embeddable on Σ and $G \setminus \{e\}$ is embeddable on Σ for every $e \in E(G)$. A graph is non-embeddable on Σ if and only if it has a subgraph homeomorphic to an obstruction for Σ. A graph embeddable on the torus is called *toroidal* otherwise *non-toroidal*.

For an embedding $\Pi(G)$ of G on Σ, each connected component of $\Sigma \setminus \Pi(G)$ is called a *face* of $\Pi(G)$ or a face of G. Each face r is an open region (subset) of Σ. We denote by $R(\Pi(G))$ the set of faces of $\Pi(G)$. We say that a vertex v (resp. edge $e = \{u, v\}$) is incident to a face r if $v \in \overline{r}$ (resp. $e \cap \overline{r}$ contains a point x with $x \neq u$ and $x \neq v$). We denote by $V(r)$ and $E(r)$ the sets of vertices and edges incident to face r, respectively. The vertices and edges incident to face r form a closed walk in $\Pi(G)$ that is the boundary $\mathrm{bd}(r)$ of r.

A cycle (closed curve) in $\Pi(G)$ is *contractible* if it bounds an area homeomorphic to an open disc, otherwise *non-contractible* (see Figure 1 (b) for contractible and non-contractible cycles on the torus). Given an embedding $\Pi(G)$ of G on a surface Σ, a *G-noose* is a closed curve on Σ that does not intersect any edge $e = \{u, v\}$ of G except its end vertices u, v. The *length* of a G-noose is the number of vertices of G it intersects. The minimum length of a non-contractible G-noose is known as the *facewidth* of G, denoted by $\mathrm{fw}(G)$.

For a graph G embedded on a surface, the *rotation* of a vertex v of G is a cyclic ordering (clockwise or counter-clockwise) of the edges incident to it. For a vertex v with $\deg_G(v) > 1$, there are $(\deg_G(v)-1)!$ different rotations of v, where $(\deg_G(v)-1)!$ is the factorial of $(\deg_G(v)-1)$. The *rotation system* of G is the collection of the rotations of all vertices and a sign +1 or -1 on every edge of G, where an edge has the sign +1 if the rotations of its two end vertices have the same direction, and has sign -1 otherwise. A graph embedded on an orientable surface has a rotation system with the sign +1 on every edge, while a rotation system for a graph embedded on a non-orientable surface may have both signs +1 and -1 on edges. Given a graph G, each embedding of G on an orientable surface corresponds to a rotation system of G. Two embeddings of G on an orientable surface are (combinatorial) *equivalent* if they have the same rotation system, otherwise *non-equivalent*. A graph G has $\prod_{v \in V(G), \deg_G(v) > 2} (\deg_G(v) - 1)!$ different rotation systems on an orientable surface.

3 Review of basic techniques

We briefly review some basic techniques for embedding graphs on the projective plane and torus. Our algorithm for embedding graphs on the torus builds on these techniques. We first introduce some notions in these works.

Let K be a subgraph of a connected graph G. A *K-bridge* [18] in G is a subgraph B of G that is either an edge $e = \{u, v\} \in E(G)$ with $u, v \in V(K)$ but $e \notin E(K)$ or a subgraph $C \cup X$ of G, where

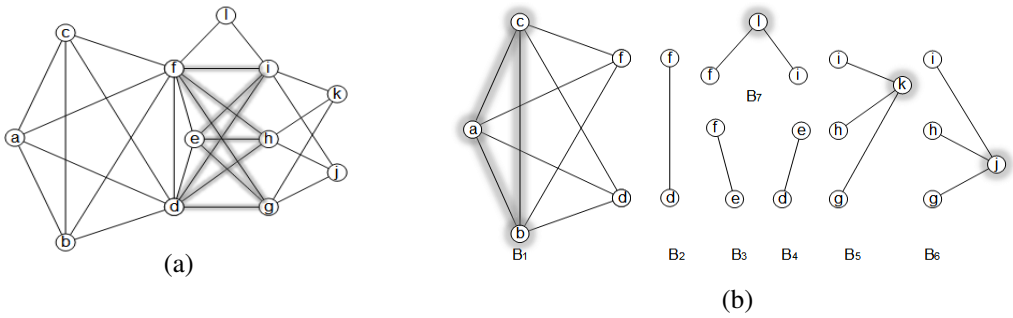

Figure 2: (a) A graph G and a highlighted subgraph K. (b) K-bridges in G.

C is a connected component of $G[V(G) \setminus V(K)]$ and $X = \{\{u, v\} \in E(G), u \in V(C), v \in V(K)\}$. Figure 2 gives examples of K-bridges. Given K, the set of K-bridges $\mathcal{B}$ can be calculated with a modified version of the depth first search (DFS) on the edges of G. For a K-bridge B, $V(B) \cap V(K)$ is called the *attachment vertex set* of B, denoted by $\mathrm{AttV}(B)$.

3.1 Embedding graphs on the projective plane

For embedding graphs on the projective plane $\mathbb{N}_1$, a linear time algorithm is known [16]. A simplified version of the algorithm, called MR Algorithm, with an $O(n^2)$ running time and its implementation are reported [20]. We review MR Algorithm which will be used as a subroutine in our algorithm for embedding graphs on the torus. For an input graph G, MR Algorithm first checks if G is planar. If the answer is yes, then a planar embedding of G is computed as an embedding of G on $\mathbb{N}_1$. Otherwise, a subgraph K of G homeomorphic to K_5 or $K_{3,3}$ is computed. Graph G can be decomposed into K and a set $\mathcal{B}$ of K-bridges. MR Algorithm finds all non-equivalent embeddings $\Pi(K)$ of K on $\mathbb{N}_1$ (27 and 6 $\Pi(K)$ for K homeomorphic to K_5 and $K_{3,3}$, respectively). For each embedding $\Pi(K)$, every bridge of $\mathcal{B}$ is assigned to a face r of $\Pi(K)$ and embedded on r.

For each face r of $\Pi(K)$ in $\mathbb{N}_1$, a vertex v in G appears in the boundary $\mathrm{bd}(r)$ of r at most once (r has 0-singularity). A face r is called a *candidate* for bridge B if $\mathrm{AttV}(B) \subseteq V(r)$. To embed B in a candidate r, each vertex v of $\mathrm{AttV}(B)$ is identified by the vertex v in $\mathrm{bd}(r)$ and draw the rest of B on r. This is called an *attachment* of B to r. A bridge has a unique attachment to a 0-singularity candidate. An attachment of B to r is *legal* if a planar embedding of B exists on r. A candidate r is *admissible* to a bridge B if B has a legal attachment to r. Two bridges B_1, B_2 and a face r are *compatible* if r is admissible to B_1 and B_2 and the union of legal attachments of B_1 and B_2 to r has a planar embedding on r, otherwise B_1, B_2, r are *incompatible*.

If each bridge of $\mathcal{B}$ has an admissible candidate (otherwise G is not embeddable on $\mathbb{N}_1$), MR Algorithm converts the problem of assigning bridges to faces into the 2-SAT problem. Assume that each bridge of $\mathcal{B}$ has at most two admissible faces. The 2-SAT instance for the bridge assignment problem is constructed as follows:

- For every bridge B and every face $r \in R(\Pi(K))$, a Boolean variable (B, r) is created such that if B is assigned to r then $(B, r) = \text{true}$, otherwise $(B, r) = \text{false}$.
- For each bridge B which has only one admissible face r, a clause $((B, r) \vee (B, r))$ is created.
- For each bridge B which has two admissible faces r_1 and r_2, two clauses $((B, r_1) \vee (B, r_2))$ and $(\overline{(B, r_1)} \vee \overline{(B, r_2)})$ are created to guarantee that B is assigned to only one face.
- For every incompatible triple B_1, B_2, r, a clause $(\overline{(B_1, r)} \vee \overline{(B_2, r)})$ is created to guarantee that B_1 and B_2 will not be both assigned to r.

It is easy to check that a true assignment for the 2-SAT instance gives an assignment of bridges to faces such that a planar embedding of the bridges can be found in the face assigned to the bridges.

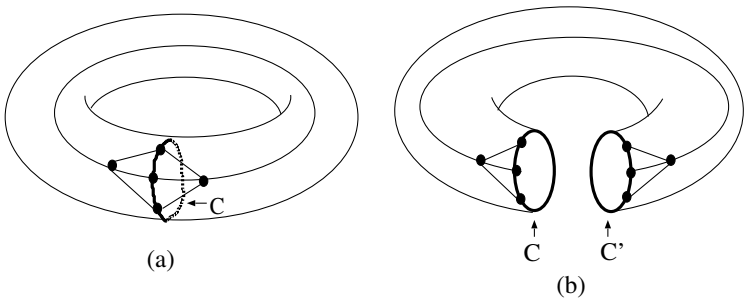

Figure 3: (a) A cycle C of K_5 is embedded as a non-contractible cycle on the torus. (b) The torus is cut along C into a cylinder.

Given that there are $O(n)$ bridges, $O(1)$ faces and $O(n^2)$ incompatible 3-tuples, the 2-SAT formula has $O(n)$ variables and $O(n^2)$ clauses and can be solved in $O(n^2)$ time.

If every bridge of $\mathcal{B}$ has at most two admissible faces, then whether G can be embedded on $\mathbb{N}_1$ can be decided by checking if the 2-SAT instance is satisfiable or not. However, not every bridge in $\mathcal{B}$ has at most two admissible faces. A good news is that only a constant number of bridges have three admissible faces, and all other bridges have at most two. For each bridge B with three admissible faces, the assignment of B to each admissible face can be enumerated. A combination of an enumeration of every such face is called an *arrangement*. For each arrangement, a 2-SAT instance is created to solve the bridge assignment problem for other bridges. The number of arrangements is a constant for K_5 and $K_{3,3}$.

3.2 Algorithms for embedding graphs on the torus

Neufeld and Myrvold introduce a simple algorithm (NM Algorithm) for embedding graphs on the torus [22]. The algorithm is based on the following observation: Every embedding of a nonplanar toroidal graph G on the torus has a non-contractible cycle C. Figure 3 (a) gives an example of C in an embedding of K_5. Given a nonplanar toroidal graph G, intuitively, NM Algorithm finds a cycle C of G as the frame and embeds C as a non-contractible cycle on the torus, cuts the torus along C to get a cylinder, makes a duplication of C on each end of the cylinder, and tries to get a planar embedding for the rest part of G on the cylinder (see Figure 3 (b) for an example). It is shown [22] that for a nonplanar toroidal graph G, at least two cycles of a cycle basis of G are non-contractible in any embedding $\Pi(G)$ of G on the torus; and at least two cycles in the cycle basis of a subgraph K of G homeomorphic to K_5 or $K_{3,3}$ are non-contractible in any embedding $\Pi(G)$ on the torus. Skoda and Mohar [24] further show that at least one cycle in the cycle basis of K homeomorphic to any subgraph K_4 of K_5 or any subgraph $K_{3,2}$ of $K_{3,3}$ is non-contractible in any embedding $\Pi(G)$ on the torus. By this result, an embedding of G on the torus can be found by trying only distinct cycles of the cycle basis of K homeomorphic to K_4 or $K_{3,2}$.

An improved version of NM Algorithm works as follows [24]: Given a graph G, if G is planar then a planar embedding of G is used as an embedding of G on the torus. For a nonplanar G, the algorithm finds a subgraph K of G homeomorphic to K_5 or $K_{3,3}$ and a subgraph K' of K homeomorphic to K_4 or $K_{3,2}$. For each cycle C in the cycle basis of K', the algorithm embeds C as a non-contractible cycle $\Pi(C)$ on the torus, cuts the torus along C to get a cylinder, puts C on one end of the cylinder, makes a duplication C' of C on the other end of the cylinder, and for every edge e in $G \setminus C$ incident to a vertex of C, attaches e to either C or C' to see if such an attachment gives a planar embedding of $G \setminus C$ on the cylinder. Let E_C be the set of edges in $G \setminus C$ incident to a vertex of C. Since each edge of E_C can be attached to C or C', NM Algorithm checks $2^{|E_C|} = 2^{O(n)}$ different attachments. NM Algorithm can be modified to an efficient tool for detecting some non-toroidal graphs [24]. This will be used in our algorithm as a pre-processing step.

The approach for embedding graphs on $\mathbb{N}_1$ in Section 3.1 can be naturally applied to get an

algorithm for embedding graph G on the torus [18]. Based on this, Woodcock [27] gives an algorithm which uses a subgraph K of G homeomorphic to K_5 or $K_{3,3}$ as the frame, finds all non-equivalent embeddings $\Pi(K)$ of K on the torus (231 and 20 $\Pi(K)$ for K homeomorphic to K_5 and $K_{3,3}$, respectively), and for each $\Pi(K)$, assigns the K-bridges to candidates r of $\Pi(K)$ by a recursive approach.

Unlike in the embedding of G on $\mathbb{N}_1$, for a face r of $\Pi(K)$ on the torus, a vertex v in G may appear in the boundary $\mathrm{bd}(r)$ of r twice (r has 1-singularity if a vertex appears in $\mathrm{bd}(r)$ twice). For a 1-singularity r, each vertex v appearing in $\mathrm{bd}(r)$ $1 \leq j \leq 2$ times is replaced by j distinct vertices $v^1, ..., v^j$ to convert the closed walk of $\mathrm{bd}(r)$ to a cycle $C(r)$. Given a 1-singularity face r, an *unfolded face* r^u is a region homeomorphic to an open disc with $C(r)$ as its boundary (i.e., $\mathrm{bd}(r^u) = C(r)$). For each vertex $v \in V(r)$, we denote by $V_r(v)$ the set of the vertices replacing v to form $C(r)$.

To embed B in a candidate r of 0-singularity, each vertex v of $\mathrm{AttV}(B)$ is identified by the vertex v in $\mathrm{bd}(r)$ and draw the rest of B on r. For a 1-singularity face r, each v is identified by some vertex v^i in $V_r(v)$ and draw the rest of B on the unfolded face r^u of r. Each of the above is called an *attachment* of B to r. Notice that for a 1-singularity candidate r, v may appear in $\mathrm{bd}(r)$ twice and if so, there are two attachments of B to r w.r.t. v. If B has k vertices appearing in $\mathrm{bd}(r)$ twice, there are 2^k different attachments to r. This is a major difference of embedding G on the torus from that on $\mathbb{N}_1$. For each bridge to face assignment, the embedding of B in r^u for every attachment of B to r is checked. There are $2^{O(|\mathrm{AttV}(B)|)}$ different attachments of B to r. The total number of checks for all bridges and all faces is $2^{O(n)}$.

Both NM Algorithm and Woodcock's algorithm have $2^{O(n)}$ attachments to check. The constant behind the Big-Oh in the exponent of $2^{O(n)}$ in NM Algorithm is larger than that in Woodcock's algorithm because $|E_C|$ is larger than the number of vertices in $\mathrm{AttV}(B)$ appearing in $\mathrm{bd}(r)$ twice (K_5 and $K_{3,3}$ place more restrictions for the extension than a cycle).

A linear time algorithm for embedding graphs on the torus is introduced in [13]. Juvan and Mohar give a simpler but $O(n^3)$ time algorithm (JM Algorithm) [14]. Both algorithms use the embedding extension approach. JM Algorithm uses a subgraph K of G homeomorphic to an obstruction for $\mathbb{N}_1$ as the frame. For any embedding $\Pi(K)$ of K on the torus, each face of $\Pi(K)$ has 0-singularity or 1-singularity. JM Algorithm eliminates each 1-singularity face r by separating r into "sub-faces" with selected paths such that duplications of a vertex v in $\mathrm{bd}(r)$ are in different "sub-faces". A further step is taken to arrange at most two candidate faces for each bridge and a constant number of arrangements need to be considered. Within each arrangement, the bridges are assigned to faces by the 2-SAT approach. While these algorithms have polynomial running time, the constants behind the Big-Oh can get large, which reduces their practical efficiency. Furthermore, the algorithms are complex and their implementations become impractical. An implementation of JM Algorithm was announced [3]. However, the implementation does not seem working [19, 24]. For these reasons, the polynomial running time algorithms have remained for theoretical interest so far.

4 A new algorithm for embedding graphs on the torus

In this section, we introduce a new algorithm for embedding graphs on the torus. We start from our observation on the differences of existing algorithms and a guideline for the new algorithm. All algorithms in the previous section follow the embedding extension approach as shown in Table 1. A general trend is that simpler frames make an algorithm easier to implement but place less restrictions for the embedding extension and thus resulting in a higher time complexity. Previous algorithms are at the two extremes of the trade off, it is desirable to find a balance in the middle where efficiency is improved but the implementation does not become a burden. To accomplish this, we try to combine the clear outline of exponential time algorithms and effective but less sophisticated techniques from polynomial time ones. Next we introduce a general scheme of the new algorithm followed by more detailed explanations on the key issues in Section 4.1. Some techniques for improving the practical running time of the new algorithm are discussed in Sections 4.2-4.5.

Table 1: Comparison among algorithms for embedding graphs on the torus.

Algorithm	Frames	Extension unit	Notes
NM	cycle	Remaining graph	
Woodcock	K homeomorphic to K_5, $K_{3,3}$	K-bridges	
JM	K homeomorphic to Obstructions for $\mathbb{N}_1$	K-bridges	Avoid 1-singularity faces

4.1 New algorithm

We propose a new algorithm for embedding graphs on the torus. Given a graph G, our algorithm uses a subgraph of G homeomorphic to an obstruction for $\mathbb{N}_1$ as the frame and try to extend an embedding of the frame to get an embedding of G on the torus. A subgraph homeomorphic to an obstruction for $\mathbb{N}_1$ has a larger size and more complex structure than a subgraph homeomorphic to K_5 or $K_{3,3}$ and places more restrictions for the embedding extension. Below are the major steps of the algorithm:

- For input graph G, use MR Algorithm in Section 3.1 to test whether G can be embedded on $\mathbb{N}_1$. If so, compute an embedding $\Pi_N(G)$ on $\mathbb{N}_1$. Otherwise, compute a subgraph K of G homeomorphic to an obstruction for $\mathbb{N}_1$ and the set $\mathcal{B}$ of K-bridges.
- Given $\Pi_N(G)$, if the facewidth of $\Pi_N(G)$ is at least four ($\text{fw}(\Pi_N(G)) \geq 4$), output G is non-toroidal and terminate. Otherwise compute an embedding $\Pi(G)$ of G on the torus from $\Pi_N(G)$ (this can be done efficiently [8]) and terminate.
- Given K, compute all non-equivalent embeddings $\Pi(K)$ of K on the torus.
- For every embedding $\Pi(K)$, try to extend $\Pi(K)$ to an embedding $\Pi(G)$ of G on the torus: for every bridge-to-face assignment (B, r), where B is a K-bridge of $\mathcal{B}$ and r is a candidate (face) of $\Pi(K)$ for B, for every attachment of B to r, if the attachment extends $\Pi(K)$ to an embedding $\Pi(G)$ then output $\Pi(G)$.

  If none of the extensions above succeeds then output G non-toroidal.

In the new algorithm, we use the $O(n^2)$ time MR Algorithm to test if G is embeddable on $\mathbb{N}_1$. If an embedding $\Pi_N(G)$ of G on $\mathbb{N}_1$ is found, the facewidth of $\Pi_N(G)$ can be calculated in linear time. If $\Pi_N(G)$ has facewidth at most three then $\Pi_N(G)$ can be converted to an embedding $\Pi(G)$ of G on the torus in linear time [8]. If G is not embeddable on $\mathbb{N}_1$, a subgraph K of G homeomorphic to an obstruction for $\mathbb{N}_1$ can be found in $O(n^3)$ time using edge elimination and MR Algorithm.

Given an embedding $\Pi(K)$ on the torus, let $\mathcal{B}$ be the set of K-bridges in G. We try to attach each bridge B of $\mathcal{B}$ to every candidate r for B. Since a face r of $\Pi(K)$ may have 1-singularity, for each bridge B to candidate r assignment, we check every possible attachment of B to r to see if this attachment gives an embedding of G on the torus. There are $2^{O(|\text{AttV}(B)|)}$ different attachments of B to r. For each bridge B of $\mathcal{B}$, there are $O(1)$ candidates of $\Pi(K)$ for B. The total number of checks for all bridges and all candidates is at most

$$\prod_{B \in \mathcal{B}} O(2^{O(|\text{AttV}(B)|)}) = 2^{O(|\mathcal{B}|)} \times 2^{O(n)} = 2^{O(n)}.$$

Since a larger and more complex frame is used in the new algorithm, the constant behind the Big-Oh in the exponent of $2^{O(n)}$ is smaller than that in the previous algorithms.

A major challenge in the new algorithm is to find all non-equivalent embeddings $\Pi(K)$ of a subgraph K homeomorphic to an obstruction for $\mathbb{N}_1$ on the torus. In the previous algorithms, all non-equivalent embeddings of K homeomorphic to K_5 and $K_{3,3}$ are manually created. This approach is impractical for the obstructions for $\mathbb{N}_1$ because an obstruction can have a large number (a few

Algorithm Torus-Embedding
Input: A connected graph G of n vertices and $m \leq 3n$ edges.
Output: An embedding $\Pi(G)$ of G on the torus or G is non-toroidal.
begin
Pre-processing Step I: Test whether G is non-toroidal by Procedure Non-Toroidal (Section 4.3).
if G is planar **then** Output a planar embedding $\Pi(G)$ and **return**.
if G is non-toroidal **then** Output G non-toroidal and **return**.
Pre-Processing Step II: Find a core subgraph H of G by Procedure Core-Graph (Section 4.4).
if G is non-toroidal **then** output G non-toroidal and **return**.
Test whether H is embeddable on $\mathbb{N}_1$ by MR Algorithm (Section 3.1).
if YES **then** Let $\Pi_N(H)$ be an embedding of H on $\mathbb{N}_1$.
if $\text{fw}(\Pi_N(H)) \leq 3$ **then** Find an embedding $\Pi(H)$ from $\Pi_N(G)$ by the algorithm in [8],
attach $G \setminus H$ to $\Pi(H)$ to get $\Pi(G)$ and **return**.
else Output G non-toroidal and **return**.
else
Find a subgraph K of H homeomorphic to an obstruction for $\mathbb{N}_1$
by MR Algorithm (Section 3.1) and find the set $\mathcal{B}$ of K-bridges of H.
Find all non-equivalent embeddings $\Pi(K)$ of K on the torus
by Algorithm All-Embeddings (Section 4.2).
for every $\Pi(K)$ **do**
$\Pi = \Pi(K)$, $\mathcal{B}' = \mathcal{B}$ and test whether Π can be extended to an embedding $\Pi(H)$
on the torus by Procedure Recursive-Extension (Section 4.5).
if YES **then** attach $G \setminus H$ to $\Pi(H)$ to get $\Pi(G)$ and **return**.
end for
end if
Output G non-toroidal.
end.

Figure 4: New algorithm.

hundreds) of non-equivalent embeddings on the torus and there are 103 different obstructions. We clear this hurdle by a simple algorithm (Algorithm All-Embeddings in Figure 5) which finds all non-equivalent embeddings of K on the torus by enumerating all rotation systems for K. Algorithm All-Embeddings has running time $O(\prod_{v \in V(K), \deg_K(v) > 2} (\deg_K(v) - 1)!)$. Since the number of vertices and node degree of each vertex in K are small constants, the algorithm is efficient in practice. More details of Algorithm All-Embeddings are given in Section 4.2.

Some non-toroidal graphs can be detected efficiently by a modified NM Algorithm [24]. Based on this, we use a simple and efficient procedure to find some non-toroidal graphs. We also test if a graph is planar by this procedure which is included in the new algorithm as Pre-processing Step I and explained in Section 4.3.

For the graphs which are not detected planar or non-toroidal in Pre-processing Step I, we try to reduce the input instance size for the new algorithm. Given such a graph G, we compute a subgraph H (called *core graph*) of G and try to find the embedding of H instead of G. This reduces the problem size for the algorithm, especially for the bridge-to-face attachment part which may run in exponential time. The computation of core graphs is included in the new algorithm as Pre-processing Step II and introduced in Section 4.4.

We further apply a recursive approach to check the bridge-to-face attachments. Bridges are assigned and attached one by one and a next bridge is processed only if the current bridge does not lead to any violation to the embeddability. In this way, the algorithm would trace back or stop at points where current structure is already not embeddable and thus save the time on checking the remaining attachments of bridges to faces. More details are given in Section 4.5.

The new algorithm with the improvements above is given in Figure 4. Notice that if a graph G

Algorithm All-Embeddings
Input: A graph G of n vertices and $m \leq 3n$ edges.
Output: All non-equivalent embeddings $\Pi(G)$ of G on the torus.
begin
Let $U(G) = \{v | v \in V(G) \wedge \deg_G(v) > 2\}$ and $W(G) = V(G) \setminus U(G)$.
Fix an arbitrary rotation for each vertex in $W(G)$.
if $U(G) = \emptyset$ **then** Find the embedding $\Pi(G)$ corresponding to the current rotation system, output $\Pi(G)$ and **return**.
for each vertex $u_i \in U(G)$ **do** Fix an edge e_i^1 as the first edge in its rotation.
Let $\mathcal{S}$ be an empty set of embeddings.
Select one vertex u_1 from $U(G)$ as the reference vertex.
Select two different edges e_1^2 and e_1^3 incident to u_1 s.t. $e_1^1 \notin \{e_1^2, e_1^3\}$.
$d = \deg_G(u_1)$.
if d is odd **then** Require that e_1^2 be in position range $[2, \frac{d+1}{2}]$ in the rotation of u_1.
for each rotation system defined by the orderings of the remaining edges **do**
if the rotation system gives an embedding $\Pi(G)$ on the torus **then** $\mathcal{S} = \mathcal{S} \cup \{\Pi(G)\}$
end for
else Require that e_1^2 be in position range $[2, \frac{d}{2}]$ in the rotation of u_1.
for each rotation system defined by the orderings of the remaining edges **do**
if the rotation system gives an embedding $\Pi(G)$ on the torus **then** $\mathcal{S} = \mathcal{S} \cup \{\Pi(G)\}$
end for
Require that e_1^2 be at position $\frac{d}{2} + 1$ in the rotation of u_1.
Require that e_1^3 be in position range $[2, \frac{d}{2}]$ in the rotation of u_1.
for each rotation system defined by the orderings of the remaining edges **do**
if the rotation system gives an embedding $\Pi(G)$ on the torus **then** $\mathcal{S} = \mathcal{S} \cup \{\Pi(G)\}$
end for
end if
Output $\mathcal{S}$ and **return**.
end.

Figure 5: Finding all non-equivalent emdeddings of G by enumerating rotation systems of G.

of n vertices has more than $3n$ edges then G is non-toroidal.

4.2 Finding all non-equivalent embeddings

All non-equivalent embeddings of G on a surface can be computed by enumerating all rotation systems of G. Figure 5 gives an algorithm for finding all such embeddings of G on the torus by enumerating the rotation systems. For each vertex v in G, there are $(\deg_G(v) - 1)!$ rotations of v. So the algorithm in Figure 5 has running time $O(\prod_{v \in V(G), \deg_G(v) > 2} (\deg_G(v) - 1)!)$ which may not scale well for graphs of large size or large node degree. For our purpose, the algorithm is only applied to a frame K homeomorphic to an obstruction for $\mathbb{N}_1$. Because the number of vertices v with $\deg_K(v) > 2$ and $\deg_K(v)$ are small constants, the algorithm performs well. Furthermore, the algorithm needs to be run on each obstruction only once since the results can be stored for later use. Given a rotation system of G, the corresponding embedding $\Pi(G)$ on the torus can be constructed efficiently [27, 20].

4.3 Pre-processing Step I: non-toroidality check

While a larger and more complex frame contributes more information to the final structure of possible embeddings of G on the torus, a smaller frame can be used to find non-toroidal graphs more efficiently. For a frame K of G, let $\mathcal{B}$ be the set of K-bridges in G. If a bridge B in $\mathcal{B}$ can not be embedded on any face (plane) of $\Pi(K)$ then we can conclude that G is non-toroidal. We call such a

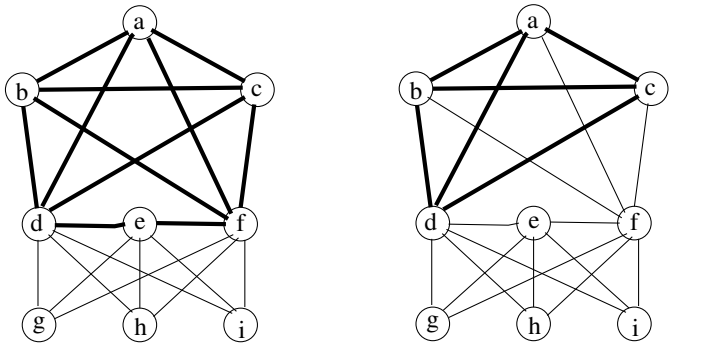

Figure 6: Frames of different sizes and their K-bridges.

bridge a *forbidden subgraph*. Given a non-toroidal G, a larger K may give a set $\mathcal{B}$ of bridges, each of them is compatible with some face of $\Pi(K)$ but the union of some bridges constitutes a forbidden subgraph G'. To conclude G non-toroidal in this case, we have to enumerate all embeddings for the bridges in $\mathcal{B}$. On the other hand, the forbidden subgraph G' may become a K-bridge for a smaller frame K and the non-toroidal conclusion can be made more efficiently (finding G' not embeddable on a plane). Figure 6 gives such an example.

As shown in Figure 6 (a), if the subgraph homeomorphic to K_5 is selected as the frame K then $\mathcal{B}$ has three bridges $\{\{g,d\},\{g,e\},\{g,f\}\}$, $\{\{h,d\},\{h,e\},\{h,f\}\}$ and $\{\{i,d\},\{i.e\},\{i,f\}\}$. Every bridge is compatible with some face of $\Pi(K)$. However, the three bridges are admissible to two faces of $\Pi(K)$ for every embedding of K, implying that two bridges need to be embedded in a same face, and any two bridges are not compatible with any face of $\Pi(K)$. That is, the union of the three bridges form a forbidden subgraph $K_{3,3}$ and G is non-toroidal. To draw this conclusion, we have to enumerate all embeddings for every combination of two bridges.

As shown in Figure 6 (b), for any cycle C of K_4 as the frame K, the forbidden subgraph $K_{3,3}$ is contained in a K-bridge B. If we embed C as a non-contractible cycle on the torus, then B can not be embedded on any face of $\Pi(K)$. If this is true for every cycle of K_4 then the graph is non-toroidal [24]. This conclusion can be made in linear time.

NM Algorithm can be modified to find some non-toroidal graphs as shown above [24]. Based on this, we include an efficient procedure for detecting some non-toroidal graphs as Pre-processing Step I in our algorithm. In this test, we use cycles in the cycle basis of a subgraph G' of G homeomorphic to a subgraph K_4 of K_5 or a subgraph $K_{3,2}$ of $K_{3,3}$ as frame K and check if $\mathcal{B}$ has a forbidden subgraph. If so, we conclude G non-toroidal, otherwise we proceed to the next step. By this test, we can find non-toroidal graphs more efficiently and thus improve the overall running time of the new algorithm. The procedure is given in Figure 7. Since there are constant number of cycles in the cycle basis of G', and the set of bridges and planarity test can be computed in linear time, the procedure has linear running time.

4.4 Pre-processing Step II: Finding core graph

To further improve the practical running time of our algorithm, we try to reduce the size of input graphs for the algorithm which may run in exponential time. For a large graph G to be embeddable, only a few small subgraphs of G are hard to embed and the remaining of G can be attached easily. These small subgraphs constitute a core graph of G. We try find a core graph H of G, find an embedding $\Pi(H)$ of H on the torus first and then attach $G \setminus H$ to $\Pi(H)$. This reduces the size of graphs for the algorithm.

We use the bi-connected components and tri-connected components [11] to find a core graph. It is proved in [4] that the genus of a graph is equal to the sum of the genera of its bi-connected

Procedure Non-Toroidal
Input: A graph G of n vertices and $m \leq 3n$ edges.
Output: G is planar or G is non-toroidal or G maybe toroidal.
begin
  if G is planar **then** Output a planar embedding $\Pi(G)$ of G and **return**.
  Find a subgraph G' of G homeomorphic to a subgraph K_4 of K_5 or
    a subgraph $K_{3,2}$ of $K_{3,3}$ and the set $\mathcal{C}$ of all cycles in the cycle basis of G'.
  for each cycle $C \in \mathcal{C}$ **do**
    Take C as frame K and compute the set $\mathcal{B}$ of K-bridges in G.
    if all bridges in $\mathcal{B}$ are planar **then** Output G maybe toroidal and **return**.
  end for
  Output G is non-toroidal.
end

Figure 7: Pre-processing Step I: test if G is non-toroidal.

components. By this result, a toroidal G has at most one nonplanar bi-connected component BC and each component C of $G \setminus BC$ is planar. Further, BC has at least one and at most two nonplanar tri-connected components. We take the union of the nonplanar tri-connected components in BC as a core graph H of G. Each connected component C of $G \setminus H$ is planar and $V(C) \cap V(H)$ has at most two vertices. Given an embedding $\Pi(H)$ of H on the torus and a planar embedding $\Pi(C)$ of C, for each vertex $v \in V(C) \cap V(H)$, we denote by v_C the copy of v in $\Pi(C)$ and by v_H the copy of v in $\Pi(H)$. An embedding of $C \cup H$ on the torus can be obtained by identifying v_C and v_H as one vertex for every $v \in V(C) \cap V(H)$. The size of core graph H is smaller than that of G. This improves the running time for the algorithm.

It it worth mentioning that tri-connected component calculation is non-trivial to implement. One implementation publicly available can be found in *Open Graph Drawing Framework(OGDF)* [6, 10]. The procedure for computing a core graph of G is given in Figure 8. The procedure has linear running time [6, 10].

Procedure Find-Core
Input: A nonplanar graph G of n vertices and $m \leq 3n$ edges.
Output: G is non-toroidal or a core subgraph H of G.
begin
  Find set $\mathcal{BC}$ of bi-connected nonplanar components of G. /* $\mathcal{BC} \neq \emptyset$ */
  if $|\mathcal{BC}| \geq 2$ **then** Output G is non-toroidal and **return**.
  Let BC be the bi-connected nonplanar component of $\mathcal{BC}$.
  Find the set $\mathcal{TC}$ of tri-connected nonplanar components of BC.
  if $|\mathcal{TC}| > 2$ **then** Output G non-toroidal and **return**.
  Output the union of the components of $\mathcal{TC}$ as core graph H and **return**.
end

Figure 8: Pre-processing Step II: Finding a core graph H of G.

4.5 Recursive embedding

Our new algorithm takes a recursive approach to enumerate all possible embeddings of the core graph H on the torus. By the recursive approach, we could prune some search branches which do not result in an embedding of H on the torus and thus improving the running time. Given a core graph H, a frame K homeomorphic to an obstruction for $\mathbb{N}_1$, the set $\mathcal{B}$ of K-bridges and all embeddings $\Pi(K)$ of K on the torus are computed. For each embedding $\Pi(K)$, we try to attach

Procedure Recursive-Extension
Input: A nonplanar graph G of n vertices and $m \leq 3n$ edges, a frame K of G,
a subset $\mathcal{B}'$ of the K-bridges and an embedding Π of a subgraph of G.
Output: G is non-toroidal or an embedding $\Pi(G)$ of G on the torus.
begin
if $\mathcal{B}' = \emptyset$ **then** Output Π as $\Pi(G)$ and **terminate**.
for every bridge $B \in \mathcal{B}'$ and every candidate r of Π for B **do**.
for every legal attachment of B to r **do**
Include the embedding of B in Π and remove B from $\mathcal{B}'$.
Test whether Π can be extended to an embedding $\Pi(G)$ of G
by Procedure Recursive-Extension.
end for
end for
Output G is non-toroidal and **return**.
end

Figure 9: Extend $\Pi(K)$ to $\Pi(H)$ recursively.

each bridge in $\mathcal{B}$ to every candidate of $\Pi(K)$ for the bridge recursively. The procedure is given in Figure 9.

5 Computational results

In this section, we show experimental results for the new algorithm. We implemented an improved version of NM Algorithm by Skoda and Mohar [24] and our new algorithm in C++. Both implementations are based on the OGDF [6] library, which includes many graph data structures and useful algorithms. Besides efficient implementation of planar embedding algorithms, one major advantage of this library over others is that it contains an implementation of SPQR-Tree [10], which is used for calculating 3-connected components of a graph and is complex to build from scratch. In the implementation of the new algorithm, we also include a graph matching library VFLib [9] for graph homeomorphism.

We run our implementations on a laptop with Intel(R) Core(TM) i3-2350M CPU 2.3GHz, 8GB physical memory and operating system Windows 7. While the processor has multiple cores, only one of them is used for the experiments. This comes both for simplicity and for consistency with computational results in others' research.

The algorithms are tested with three categories of graphs. Category 1 (C1) instances are based on Delaunay triangulations of point set taken from TSPLIB [23]. These original graphs were widely used in research on planar graphs. While these graphs are planar, they can make good inputs for torus embedding by adding some additional edges. Category 2 (C2) instances are randomly generated toroidal graphs. These graphs have edge number between $2n$ and $3n$ so that the generated graphs are not trivially toroidal. They are also guaranteed to be simple, connected and nonplanar. Category 3 (C3) instances are randomly generated graphs with edge number between n and $3n$ where toroidality is unknown. We compare the performance of our algorithm with that of NM Algorithm. The running time of the algorithms are given in Tables 2 and 3 for C1 graphs, and in Tables 4 and 5 for C2 graphs and C3 graphs, respectively. In the tables, $n = |V(G)|$ and $m = |E(G)|$. We test 100 graphs for each graph size and show the average running time (Avg), worst case running time (Max) and standard deviation (Std). The running time is in milliseconds and a time less than 1 millisecond is indicated by 0. NA indicates that the algorithm did not stop in 24 hours.

Graphs of Category C1 are based on Delaunay triangulations of point set. These triangulations have the property that each face except for the outer one is bounded by three edges. For one such triangulation H, adding one edge e that does not lie on any face r ($e = \{u, v, \} \not\subseteq V(r), r \in R(H)$) to H breaks its planarity but the resulting graph is still toroidal. Further additional edges have a

large chance to break its toroidality. In our experiments, we construct C1 graphs by adding one or two random edges to these triangulations. We divide C1 graphs further into two subcategories: C1A graphs and C1B graphs, obtained by adding one random edge and two random edges to the triangulations, respectively. Therefore, C1A graphs are all toroidal and C1B are mostly non-toroidal. With this construction, C1A and C1B represent two boundary cases of input: toroidal graphs that are close to be non-toroidal and non-toroidal graphs that are close to be toroidal. C1A graphs are harder to embed than other toroidal graphs considering its larger number of edges. Toroidality of C1B graphs are harder to detect because less edges could lead to trivial non-toroidality detection by pre-processing steps. As a result, they can properly represent the worst case inputs for the algorithms.

The running time of NM Algorithm and our algorithm for C1A graphs is given in Table 2. The average time of the algorithms is also shown in Figure 10.

Table 2: Running time for C1A graphs.

n	m	NM Algorithm			New Algorithm		
		Avg	Max	Std	Avg	Max	Std
51	141	191	8,732	900	175	558	132
130	378	3,541	70,820	11,756	671	2,455	488
225	623	72,985	949,600	234,513	3,216	19,400	4,366
280	789	NA	NA	NA	2,046	192,400	19,313

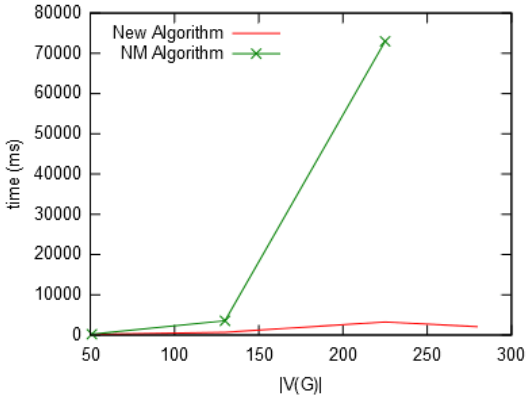

Figure 10: Average running time for C1A graphs.

The running time of NM Algorithm and our algorithm for C1B graphs is given in Table 3. The average time of the algorithms is also shown in Figure 11.

Table 3: Running time for C1B graphs.

n	m	NM Algorithm			New Algorithm		
		Avg	Max	Std	Avg	Max	Std
51	142	234	5,830	795	183	386	77
130	379	4,692	70,820	22,458	654	1,878	345
225	624	25,575	188,500	109,473	2,318	7,328	1,777
280	790	NA	NA	NA	600	4,811	1,052

Category 2 instances are randomly generated toroidal graphs with edge number between $2n$ and $3n$. By bounding the number of edges in graphs, it avoids dense graphs that are guaranteed to be

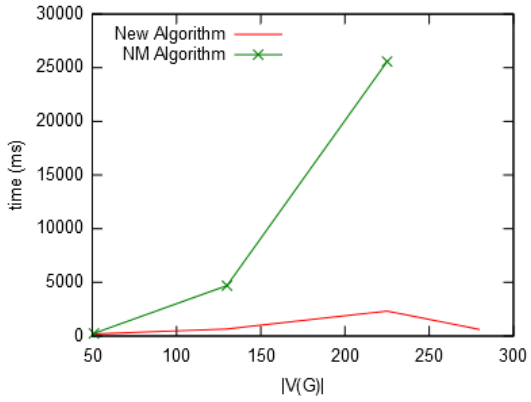

Figure 11: Average running time for C1B graphs.

non-toroidal and too sparse graphs whose embedding becomes trivial. Unlike the extreme cases in C1A instances, graphs in this category represent more general cases where embedding can be either hard or easy. More importantly, it avoids fixed structures from the base graphs, making the results more general and graph-independent. Results for this category are shown in Table 4.

Table 4: Results with C2 graphs as input

n	m	NM Algorithm			New Algorithm		
		Avg	Max	Std	Avg	Max	Std
20	[40, 60]	163	4,961	636	35	112	14
30	[60, 90]	31	630	82	46	114	16
40	[80, 120]	72	4,789	477	64	142	25
50	[100, 150]	557	53,820	5,354	77	191	29
60	[120, 180]	53	2,363	246	93	197	36
70	[140, 210]	131	11,890	1,182	115	321	51
80	[160, 240]	17	412	59	128	362	55
90	[180, 270]	307	12,700	1,733	151	387	67
100	[200, 300]	836	35,520	4,960	178	519	90
110	[220, 330]	1,675	112,300	13,226	187	571	83

Category 3 graphs are also randomly generated but their toroidality are not known. These randomly generated graphs have big chance to be non-toroidal, this data set puts more focus on the performance on non-toroidal graphs. The results are shown in Table 5.

In Tables 2 and 3, NM Algorithm shows a dramatic growth in running time as the input graph size increases. When the input graph has size 280, NM Algorithm fails to yield a result in a reasonable time. In contrast, the new algorithm handles this case well by finishing within a few seconds in average and a few minutes at most. Despite that the new algorithm also has exponential running time in theory, it has an efficient practical performance for the toroidal (C1A) graphs which are hard to embed and the non-toroidal (C1B) graphs which are hard to detect.

Table 4 shows that the average running time for the new algorithm increases closely to linear and its average and maximum running time is more efficient than NM Algorithm. This trend matches the results for C1A (toroidal) graphs as well.

Table 5 shows a different aspect of the algorithms' performance. Graphs in C3 consists of more non-toroidal cases than the previous two categories. In the generated graphs, the proportion of toroidal graphs decreases exponentially in the input graph size. The running time of NM Algorithm indicates that NM Algorithm is more efficient to detect the random non-toroidal graphs. For the

Table 5: Results with C3 graphs as input

n	m	NM Algorithm			New Algorithm		
		Avg	Max	Std	Avg	Max	Std
20	[20, 60]	10	175	31	20	71	17
40	[40, 120]	16	1,315	131	28	74	16
60	[60, 180]	8	286	33	30	76	16
80	[80, 240]	3	103	13	40	85	15
100	[100, 300]	9	706	70	40	78	15
120	[120, 360]	7	272	37	44	87	18
140	[140, 420]	3	124	12	47	138	20
160	[160, 480]	2	3	1	48	134	22
180	[180, 540]	2	4	1	53	100	20
200	[200, 600]	2	5	1	59	116	27

new algorithm, its running time is still stable in the input size regardless of the toroidal proportion. Therefore, its running time is less dependent on the toroidality of input graphs. Although its average running time exceeds that of NM Algorithm for the random non-toroidal graphs, this time is still far less than a second and stays within an acceptable range. It is worth to mention that NM Algorithm is efficient for detecting the random non-toroidal graphs but is far less efficient than the new algorithm for the non-toroidal (C1B) graphs which are hard to detect.

A computational study on NM Algorithm (the original version [22]) and Woodcock's algorithm (using a subgraph homeomorphic to K_5 and $K_{3,3}$ as the frame) is reported [27]. We do not have an access to the code, the graph instances and the computer used in the study. It is difficult to have a direct comparison between the results in this paper and these in [27] because of different computing platforms and data sets. The method for generating the random toroidal graphs in [27] is similar to that for C2 graphs. We quote some results from [27] for the random toroidal graphs in Table 6. The running time in the table is in milliseconds by a computer with a CPU of 3.6GHz. The average time (Avg) is on 100 instances of a same size. A rough comparison between the results for C2 graphs (Table 4) and these quoted from [27] for the random toroidal graphs (Table 6) is as follows: The CPU used for Table 4 has 2.3GHz and the CPU used in Table 6 has 3.6GHz [27]; the running time of the original NM Algorithm increases faster in the graph size than that of the improved NM Algorithm; and similarly, the running time of Woodcock's algorithm increases faster than the new algorithm in this paper.

Table 6: Results quoted from Table 5.2 in [27] for the random toroidal graphs. NM Algorithm used in this table is the original version.

n	NM Algorithm		Woodcock's Algorithm	
	Avg	Max	Avg	Max
20	4	40	1	10
30	22	150	3	20
40	90	1,340	10	70
50	248	2,410	19	230
60	449	2,360	31	410
70	703	4,840	69	2,010
80	1,656	26,480	92	3,480
90	3,679	64,720	923	77,810
100	3,529	57,050	145	1,390
110	8,579	276,180	351	11,650

From the analysis above, we conclude that the new algorithm is more efficient and more preferable

in general. As long as toroidality of input graphs are unknown, we can safely assume that both toroidal and non-toroidal graphs exist and worst cases will happen.

6 Concluding remarks

In this paper, we proposed a new algorithm for embedding graphs on the torus. The computational results show that the running time of the new algorithm does not increase rapidly in the graph size for both toroidal and non-toroidal graphs. Especially, the new algorithm is much faster than the previous algorithms for the toroidal graphs and thus gives an efficient tool for the applications which require the embeddings of graphs on the torus. The tool is available on request. The implementation of our algorithm can be considered as one big step towards a full implementation of the $O(n^3)$ time JM Algorithm. The remaining work for implementing JM Algorithm is to realize the functions to avoid 1-singularity faces. This work may be complex and challenge but worth to explore. It is also interesting to find the complete set of obstructions for the torus. It is proved that there is a constant number of obstructions for the torus but the complete set of obstructions is not known yet (at least 239451 obstructions have been identified [27]). Our algorithm can be applied to find the set.

Acknowledgment

The authors thank Skoda for his improved version of NM Algorithm and the code (in JAVA) of the algorithm, and thank Mohar and Skoda for their discussions on the problem of embedding graphs on the torus.

References

[1] Library of Efficient Data Types and Algorithms, Version 5.2, 2008. http://www.algorithmic-solutions.com/enleda.htm.

[2] Public Implementation of a Graph Algorithm Library and Editor, 2007. http://pigale.sourceforge.net/.

[3] Algorithms for embedding graphs in surfaces, 2014. http://www.fmf.uni-lj.si/ mohar/.

[4] Battle, J.; Harary, F.; Kodama, Y. Additivity of the genus of a graph. *Bulletin of the American Mathematical Society* **1962**, *68*, 565–568.

[5] Boyer, J.M.; Myrvold, W. Simplified O(n) planarity algorithms **2001**.

[6] Chimani, M.; Gutwenger, C.; Jünger, M.; Klau, G.W.; Klein, K.; Mutzel, P. The open graph drawing framework (OGDF). *Handbook of Graph Drawing and Visualization* **2011**, pp. 543–569.

[7] Demoucron, G.; Malgrange, Y.; Pertuiset, R. Graphes planaires: reconnaissance et construction de représentations planaires topologiques. *Revue Française de Recherche Opérationelle* **1964**, *8*, 14.

[8] Fiedler, J.; Huneke, J.P.; Richter, R.B.; Robertson, N. Computing the orientable genus of projective graphs. *Journal of Graph Theory* **1995**, *20*, 297–308.

[9] Foggia, P. The vflib graph matching library, version 2.0, 2001.

[10] Gutwenger, C.; Mutzel, P. A linear time implementation of SPQR-trees. Proceedings of 2000 Graph Drawing, GD00, LNCS 1984, 2001, pp. 77–90.

[11] Hopcroft, J.E.; Tarjan, R.E. Dividing a graph into triconnected components. *SIAM Journal on Computing* **1973**, *2*, 135–158.

[12] Hopcroft, J.; Tarjan, R. Efficient planarity testing. *Journal of the ACM (JACM)* **1974**, *21*, 549–568.

[13] Juvan, M.; Marincek, J.; Mohar, B. Embedding a graph into the torus in linear time. *preprint* **1994**, (abstract in the Proceedings of the 1995 International Conference on Integer Programming and Combinatorial Optimization, IPCO1995, LNCS 920, pp. 360-363).

[14] Juvan, M.; Mohar, B. An algorithm for embedding graphs in the torus. *preprint* **1998**.

[15] Kawarabayashi, K.; Mohar, B.; Reed, B. A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width. Proceedings of the 49th IEEE Symposium on Foundations of Computer Science. 2008, pp. 771–780.

[16] Mohar, B. Projective planarity in linear time. *Journal of Algorithms* **1993**, *15*, 482–502.

[17] Mohar, B. Embedding graphs in an arbitrary surface in linear time. Proceedings of the twenty-eighth annual ACM symposium on Theory of computing. ACM, 1996, pp. 392–397.

[18] Mohar, B. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics* **1999**, *12*, 6–26.

[19] Myrvold, W.; Kocay, W. Errors in graph embedding algorithms. *Journal of Computer and System Sciences* **2011**, *77*, 430–438.

[20] Myrvold, W.; Roth, J. Simpler projective plane embedding. *Electronic Notes in Discrete Mathematics* **2000**, *5*, 243–246.

[21] Mohar, B.; Thomassen, C. *Graphs on surfaces*; Vol. 10, JHU Press, 2001.

[22] Neufeld, E.; Myrvold, W. Practical toroidality testing. Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms. Society for Industrial and Applied Mathematics, 1997, pp. 574–580.

[23] Reinelt, G. TSPLIB-A traveling salesman problem library. *ORSA journal on computing* **1991**, *3*, 376–384.

[24] P.Skoda, B.; Mohar, B. personal communication, 2012.

[25] Thomassen, C. The graph genus problem is NP-complete. *Journal of Algorithms* **1989**, *10*, 568–576.

[26] Thomassen, C. Triangulating a surface with a prescribed graph. *Journal of Combinatorial Theory, Series B* **1993**, *57*, 196–206.

[27] Woodcock, J.R. A faster algorithm for torus embedding. Master's thesis, University of Victoria, 2006.

[28] Yu, J.; Gu, Q. A practical algorithm for embedding graphs on torus. Proceedings of the third International Symposium on Computing and Networking, 2015, pp. 50-57.