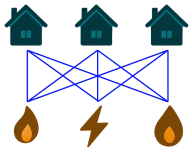
AN EFFICIENT GENUS ALGORITHM BASED ON GRAPH ROTATIONS

ALEXANDER METZGER AND AUSTIN ULRIGG

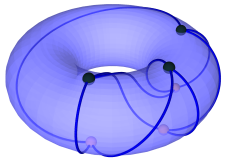
Abstract. We study the problem of determining the minimal *genus* of a simple finite connected graph. We present an algorithm which, for an arbitrary graph G with n vertices and m edges, determines the orientable genus of G in $\mathcal{O}(n(4^m/n)^{n/t})$ steps where t is the *girth* of G. This algorithm avoids difficulties that many other genus algorithms have with handling bridge placements which is a well-known issue [17]. Its implementation can be found here under an MIT license. The algorithm has a number of useful properties for practical use: it is simple to implement, it outputs the faces of an optimal embedding, and it iteratively narrows both upper and lower bounds. We illustrate the algorithm by determining the genus of the $(3, 12)$ cage (which is 17); other graphs are also considered.

1. Introduction

1.1. **Motivation and Background.** Say that you have three houses and three utilities, and you must connect each house to each utility via a wire, is there a way to do this so that none of the wires cross each other? This problem can be reframed in terms of graph theory: is $K_{3,3}$ planar? Kuratowski's theorem [12] tells us that it is not. However, $K_{3,3}$ is *toroidal*, it can be embedded on a torus without any edges crossing.



(a) The Complete Bipartite Graph $K_{3,3}$.



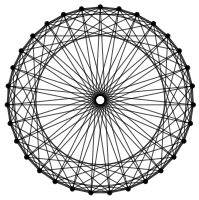
(b) $K_{3,3}$ Embedded on a Torus.

The property of a torus that allows us to embed $K_{3,3}$ is that it has a handle (unlike surfaces such as a plane or a sphere). This motivates classifying surfaces by their number of handles, that is, their genus g. In these terms, we have seen that the minimum genus surface that $K_{3,3}$ can be embedded has $g = 1$, and we say that $K_{3,3}$'s genus is 1. Although one may also consider embeddings on non-orientable surfaces, the focus of our algorithm in this paper is only on the orientable genus. In the orientable case for genus zero we use the special name "planar" and for genus one we use "toroidal". Similarly, it is known that the complete graph with 7 vertices, K_7, has genus 1 and can be embedded on a torus. However, K_8 cannot be embedded on a torus, and has genus 2. In fact, Ringel [19, 20], determined the minimum non-orientable genus for the complete graph K_n and also the orientable and non-orientable genus for the complete bipartite graph $K_{m,n}$. Further, Ringels and Youngs later determined the minimum orientable genus for K_n [21]:

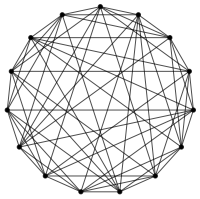
$$g(K_n) = \left\lceil \frac{(n-3)(n-4)}{12} \right\rceil \qquad g(K_{m,n}) = \left\lceil \frac{(n-2)(m-2)}{4} \right\rceil$$

However, it is not always so simple to determine the genus of an arbitrary graph. For example, the following are examples of graphs with unknown genus.

2020 *Mathematics Subject Classification.* Primary 05C85; Secondary 05C10.
Key words and phrases. Genus of graph, rotation system, genus algorithm.



(a) Cyclotomic 31 Graph ($12 \leq g \leq 26$).



(b) Johnson (6,2) Graph ($4 \leq g \leq 5$).

The challenge of determining the orientable genus of graphs and constructing their embeddings is a fundamental problem in graph theory, with applications in map colouring, very large scale integration, topology, and network science.

2. Main Result

Throughout this paper G denotes a finite connected simple graph, with n vertices, m edges, and girth t. We present a simple algorithm to determine the genus of a graph: Practical_Algorithm_for_Graph_Embedding (PAGE). The algorithm runs faster than previously implemented algorithms including those presented in [2, 5, 8]. PAGE can easily handle graphs like K_7 and K_8 in a few seconds and scales well to graphs with over a hundred edges, which it can run in a few minutes. The algorithm also provides upper and lower bounds which it iteratively narrows as it runs, allowing for practical applications.

Theorem 1 (Main Result)**.** *PAGE described in §6 determines the genus of G with runtime of $\mathcal{O}(n(4^m/n)^{n/t})$.*

We emphasize that PAGE is relatively easy to implement; moreover §4 contains a number of concrete examples where the algorithm is used to determine the genus of graphs whose genus was previously unknown.

3. Preliminaries

For the convenience of the reader, following along with *Graphs on Surfaces* [16] by Mohar and Thomassen, we provide the following definitions and relevant theorems.

3.1. **Graph Embeddings and Surfaces.** Intuitively, one can think of an embedding of a graph on a surface as a drawing of the graph on that surface without any edges crossing. We will now make this precise. A *curve* in a topological space X is the image of a continuous map $f : [0, 1] \to X$. It is called *simple* if f is injective, and a simple arc with endpoints $f(0)$ and $f(1)$ is said to *connect* these endpoints [16, Sec. 2.1]. A graph G is *embedded* in a topological space X if the vertices of G are distinct points in X and every edge of G is represented by a simple arc connecting in X the vertices it joins in G, such that the interior of each arc is disjoint from other edges and vertices [16, Sec. 2.1]. When we later discuss rotation systems and facial walks, we use the associated directed edges: an undirected edge joining vertices u and v has two directed versions, (u, v) and (v, u), corresponding to the two possible directions of traversal along the same arc. An *embedding* of G into X is thus an isomorphism of G with a graph embedded in X. If such an embedding exists, we say that G *can be embedded in* X [16, Sec. 2.1]. An embedding of G in a surface S is said to be *cellular* (or a *2-cell embedding*) if every face of G is homeomorphic to an open disc in $\mathbb{R}^2$. Every cellular embedding is a 2-cell embedding, and vice versa [16, Thm. 3.2.4].

3.2. **Rotation Systems.** Cellular embeddings can be encoded combinatorially by listing the local rotations at each vertex. Given a cellular embedding of G in a surface S, let $\Pi = \{\pi_v \mid v \in V(G)\}$, where each π_v is a cyclic permutation of the edges incident with vertex v, such that $\pi_v(e)$ is the successor of e in the clockwise ordering around v. Each permutation π_v is called the *local rotation* at vertex v, and the set Π itself is the *rotation system* of the embedding of G in S [16, p. 90].

Rotation systems determine the embedding up to homeomorphism, as shown below.

Theorem 2 (Heffter-Edmonds-Ringel)**.** [16, p. 90] *Suppose that G is a connected multigraph with at least one edge that is cellularly embedded in an orientable surface S. Let Π be the rotation system of this embedding, and let S' be the surface of the corresponding 2-cell embedding of G. Then there is a homeomorphism of S onto S' taking G in S onto G in S' (in such a way that we induce the identity map from G onto its copy in S'). In particular, every cellular embedding of a graph G in an orientable surface is uniquely determined, up to homeomorphism, by its rotation system.*

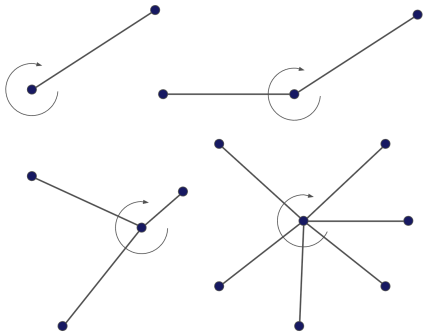


Figure 3. Local Rotations at a Vertex.

3.3. **Orientable Genus.** The orientable genus of a graph G quantifies exactly how many handles a surface needs for G to be embedded in it. The *orientable genus* g of a graph G is the smallest integer h for which G embeds in the orientable surface S_h, the connected sum of h tori [16, p. 94].

3.4. **Facial Walks of a Rotation System.** Once $\Pi = \{\pi_v\}$ is fixed, we build each face by starting at a vertex edge pair and traversing as follows:

$$(v, e) \xrightarrow{\pi_v} (v,\, \pi_v(e)) \xrightarrow{\text{traverse}} (v',\, \pi_v(e)) \xrightarrow{\pi_{v'}} \cdots$$

Because G is finite this returns to the start, tracing out a closed walk in G. Such a closed walk is called a *facial walk* of Π. The amount of distinct facial walks of a rotation system Π is denoted F (we do not distinguish a facial walk from any of its cyclic shifts). An embedding attaining the minimal genus is a *minimum genus embedding* [16, p. 95]. Every minimum genus embedding of a connected graph is cellular [16, Prop. 3.4.1]. Hence, genus computations may focus exclusively on cellular embeddings, and in particular, by Theorem 2 above, only on rotation systems. In fact,

Theorem 3 (Every Rotation System Corresponds to an Embedding)**.** [16, p. 87] *Every connected multigraph with at least one edge admits a 2-cell embedding in some orientable surface. The embedding is constructed by forming a polygon for each facial walk defined by the rotation system, and identifying the sides of these polygons along their shared edges.*

Thus, when enumerating possible embeddings of G, one may safely iterate over all $\prod_{v \in V}(\deg(v) - 1)!$ rotation systems, since each corresponds to an embedding on an orientable surface of some genus. The genus g of the surface can then be computed using Euler's formula,

$$n - m + F = 2 - 2g, \implies g = \frac{-n + m - F + 2}{2}$$

Thus, the smallest g occurs when F is maximal, i.e, when we find a rotation system with the most amount of induced facial walks.

3.5. **Definitions.** Throughout this paper, a *non-backtracking closed directed trail* is a walk that starts and ends at the same vertex, uses no repeated directed edges, and does not immediately follow any directed edge by its reverse. In this paper, a *closed trail* always refers to a non-backtracking closed directed trail unless otherwise stated. Additionally, we say that a collection $\mathcal{F}$ of *closed trails* is *realizable* as a rotation system of G if there exists a rotation system Π of G whose facial walks exactly match the closed trails in $\mathcal{F}$.

3.6. **Prior Work.** It has been well established that the problem of determining the genus is NP-hard [24]. However, it is tractable for fixed genus [15, 17, 22]. Mohar has proven the existence of a linear time algorithm for arbitrary fixed surfaces [15]. Unfortunately, these theoretical results are non-constructive and so far no one has been able to design an algorithm that achieves such performance [17]. For instance, Robertson and Seymour showed that checking if a graph contains a given minor can be done in cubic time [22] and that, for each fixed surface, there are only finitely many forbidden minors that determine whether a graph embeds in that surface [23]. However, the complete list of forbidden minors is only known for planar [12] and projective plane graphs [1]. Even small toroidal graphs (genus 1) cannot be solved with this approach since there are at least 17.5K toroidal minors and likely many more [18]. In practice, the finite number of

graph minors scales at an impractical exponential rate with the genus. This makes it intractable to compute all the minors [17]. As it stands, the best implemented algorithms are exponential time, and the rest are too complex/impractical for implementation and have intractable constant factors [4, 17, 18].

3.7. **Existing Algorithms and Limitations.** An algorithm called MULTI_GENUS by Gunnar Brinkmann is a particularly fast method for computing genus of graphs with relatively low genus compared to the vertex degree [3]. Although a formal runtime analysis of MULTI_GENUS has not been presented, experimental results suggest it handles graphs with higher vertex degrees efficiently. However, PAGE seems to remain advantageous for graphs with vertices only of degree 5 or lower. Additionally, our approach is comparatively simpler to implement. As it stands, MULTI_GENUS represents the fastest known approach for high-degree graphs, and low relative genus, whereas PAGE provides an effective alternative, and is particularly advantageous for graphs with bounded vertex degree or high girth.

Several other genus algorithms outside of MULTI_GENUS have been developed, improving upon earlier methods through more optimized implementations and by using existing optimized solvers such as those for integer linear programming [2, 5, 8]. They can easily compute the genus of graphs the size of K_6 in less than a second but, even just adding another vertex, K_7 could take hundreds of hours [2]. K_8 and above is almost entirely out of reach. They omit formal runtime bounds, which are likely super-exponential for general graphs. For example, the best available open-source implementation in SageMath exhibits a worst-case complexity of $\mathcal{O}(n(n-1)!^n)$ [8].

4. Results on Specific Graphs

The purpose of this section is to describe various results that we obtained when using PAGE to determine the genus of certain graphs.

4.1. **Cages.** A graph family of special interest is the (r,t)-cage graphs. They are the smallest r-regular graphs with girth t. The genus of $(3,t)$ cage graphs is known up to $t=10$, and PAGE extends these results by determining the genus of the $(3,12)$ cage. Although the structure of $(3,t)$ cages is not fully known for $t>12$, our approach is likely applicable to higher values of t as these cages are discovered. We outperform all existing algorithms, including MULTI_GENUS, for $t>8$, and have the only tractable algorithm for $t\geq 12$.

Theorem 4. *The genus of the unique* $(3,12)$ *cage graph is 17.*

An example of a huge graph that is impractical to embed optimally is the (6, 12) cage graph. It consists of 7812 vertices, 23436 edges, and an automorphism group of nearly 6 billion elements. Nonetheless, PAGE can still progressively narrow down the genus range. We established bounds for the genus of the (6,12) cage graph between 5860 and 7810 before encountering memory limitations.

4.2. **Circulant and Complete Multipartite Graphs.** Another graph family that is of special interest is the complete n-partite graph $K_{2,2,\ldots,2}$ (n copies of 2), also known as the cocktail party graph of order n. It is conjectured to have genus $\lceil (n-1)(n-3)/3 \rceil$ for all n, proven for all n not a multiple of 3 [11]. The complete n-partite graph represents the problem of how many handshakes are possible in a room of n couples if no one shakes their own partner's hand and has many applications in combinatorics. It is known that $K_{2,2,\ldots,2}$ is isomorphic to the circulant graph $Ci_{2n}(1,2,\ldots,n-1)$, defined as the graph with vertices labeled $0,1,\ldots,2n-1$ where each vertex i is adjacent to vertices $(i+j) \bmod 2n$ and $(i-j) \bmod 2n$ for every $j \in \{1,2,\ldots,n-1\}$. The genus is also known for all circulant graphs with genus 1 and 2 [6]. However, not all circulant graphs are complete n-partite graphs or of small genus. In the vast majority of cases, the genus of arbitrary circulant graphs is unknown. Using PAGE, we were able to determine the genus for several circulant graphs where the values were previously unknown in less than a second:

Theorem 5 (Circulants)**.** *The genus of* $Ci_{14}(1,2,3,6)$ *is 4. The genus of* $Ci_{18}(1,3,9)$ *is 4. The genus of* $Ci_{20}(1,3,5)$ *is 6. The genus of* $Ci_{20}(1,6,9)$ *is 6.*

Moreover, our approach also verified the genus of certain circulant graphs that correspond to well-known structures, such as the complete n-partite graphs: the genera of certain complete multipartite graphs are well-established: the genus of $K_{2,2}$ is 0. The genus of $K_{2,2,2}$ is 0. The genus of $K_{2,2,2,2}$ is 1. The genus of $K_{2,2,2,2,2}$ is 3. These values were verified with PAGE, consistent with the known results in the literature (see [11]).

4.3. **The Gray Graph.** Additionally, significant interest has surrounded the genus of the Gray graph (it happens to be 7), which has been addressed in a dedicated study [14]. Most existing algorithms, except MULTI_GENUS, require over 42 hours to compute the genus of similar graphs, whereas PAGE achieves the same result in just a few minutes.

4.4. **Progressive Refinement of Genus Bounds.** For large graphs, an exact genus is often infeasible and unnecessary in practice. For cases where it suffices to have an embedding within some error tolerance of the fewest handles, PAGE outputs genus bounds that narrow with increasing iterations.

Theorem 6. *For any graph G with genus g, PAGE computes two monotone sequences of integers*

$$g_0 \leq g_1 \leq \cdots \leq g_r = g, \qquad G_0 \geq G_1 \geq \cdots \geq G_s = g,$$

that converge to g and,

(1) PAGE halts when $g_i = G_j$, at which point $g_i = g = G_j$.
(2) The length of both sequences is finite, so PAGE always terminates.

The proof of Theorem 6 is included later in §8.1. To demonstrate the usefulness of Theorem 5, we applied PAGE to large graphs, establishing the genus bounds in Table 1 within 15 minutes.

Graph Name	Reference	Genus Lower Bound	Genus Upper Bound
Bipartite Kneser Graph (12, 3)	[27]	4401	8979
DifferenceSetIncidence(40, 13, 4)	*	91	214
Johnson Graph (8,4)	[28]	60	238
Johnson Graph (9, 4)	[28]	148	558
HoffmanSingletonBipartDoubleGraph	*	68	100
Higman Sims Graph	[25]	226	490
Cyclotomic Graph 61	[26]	73	265
Cyclotomic Graph 67	[26]	91	325

Table 1. Genus bounds for large graphs established by PAGE within 15 minutes. Graph names follow the built-in naming used by *Mathematica*. *These graphs are defined internally by `GraphData`, but lack publicly available documentation.

5. Theoretical Justification of PAGE

5.1. **Pre-Processing.** We will limit our analysis to connected graphs with vertices of degree > 2 since all graphs simplify to this case per Lemma 1.

Lemma 1. *We can simplify any graph to remove vertices of degree ≤ 2 without changing its genus. The genus of a disconnected graph can be calculated by summing the genera of its connected components.*

Proof. Degree 0 vertices are isolated and can be drawn anywhere on the surface without causing edge crossings since they are not connected to any edges. Degree 1 vertices likewise can always be added back into an embedding. Degree 2 vertices can simply be removed and replaced with an edge connecting its neighbors. The additivity of the minimum orientable genus over connected components has been well established [16]. □

5.2. **Rotation Systems and Euler's Formula.** The general idea of PAGE is Euler's formula $n-m+F = 2-2g$ which links the number of facial walks F with the genus g [9]. Naively, finding the minimum genus amounts to searching all combinations of closed trails to find the one that corresponds to a maximal number of facial walks as seen in Lemma 3. Traditional improved exhaustive search algorithms instead search through the rotation systems since they each induce an embedding [8, 10]. Searching through rotation systems however does not easily allow further pruning of the search space nor inform a heuristic search through rotation systems in an order that most quickly narrows down the genus. Our algorithm instead searches through closed trail combinations which facilitates a number of optimizations (early stopping, heuristic search) and, with our main contribution, still allows us to prune collections of closed trails that cannot be realized as a rotation system of G, which results in an exponentially reduced search space.

5.3. **Necessary Conditions for Realizing Closed Trails.** The following lemmas detail necessary conditions for realizing collections of closed trails as rotation systems, allowing PAGE to reduce the search space of all combinations of closed trails.

Lemma 2. *Given a graph G, any set of facial walks induced by a rotation system must use each directed edge exactly once.*

Proof. This is clear by the definition of a rotation system. □

Lemma 3. *Given a graph G, the set of closed trails C of G is finite and any set of facial walks induced by a single rotation system of G is a subset of C.*

Proof. The finite size of C follows from the finite combinations of up to $2m$ directed edges and that a closed trail cannot repeat a directed edge and must therefore contain at most $2m$ directed edges. By Lemma 2, any facial walk in a set of facial walks induced by a rotation system of G is a closed trail. □

Lemma 4. *Given a graph G choose any vertex $v \in V$ of degree d. Then in any set of facial walks induced by a rotation system of G, the vertex v occurs exactly d times as part of a closed trail.*

Proof. By Lemma 2, any set of facial walks induced by a rotation system of G must use each directed edge of G exactly once. Since each facial walk is a closed trail by Lemma 3, it must use $2k$ of the directed edges that touch v for some non-negative integer k representing the number of occurences of v in that facial walk. There are $2d$ directed edges that touch v and thus $2d/2 = d$ occurrences of v. □

Lemma 5. *Given a graph G, any set of facial walks induced by a rotation system must be a set of closed trails whose lengths add up to the number of directed edges $2m$.*

Proof. By Lemma 3, the set of facial walks is a set of closed trails. The length of each trail is the number of directed edges it uses. A set of facial walks induced by a rotation system of G uses each directed edge of G exactly once by Lemma 2. There are $2m$ directed edges in G. □

Proposition 1. *Let G be a graph. Then any collection of facial walks $\mathcal{F}$ induced by a rotation system of G is a set of closed trails that satisfies the following:*

(1) Every directed edge appears in exactly one closed trail.
(2) There do not exist two distinct closed trails c_1 and c_2 such that c_1 contains the sequence of directed edges $(a, b), (b, c)$ and c_2 contains the reversed sequence $(c, b), (b, a)$ for any vertex b with $\deg(b) > 2$.

Proof. Each facial walk is constructed by following directed edges according to the cyclic order at each vertex defined by the rotation system. Since the walk turns at each step without reversing the incoming edge, it is a closed trail. The rotation system specifies a unique next edge for every incoming edge, so the facial walks partition the $2m$ directed edges of G, with each used exactly once. Now suppose for contradiction that two facial walks c_1 and c_2 contain subpaths $(a, b), (b, c)$ and $(c, b), (b, a)$ respectively. Then b must simultaneously follow both cyclic orders (a, b, c) and (c, b, a), which is impossible unless $\deg(b) = 2$, in which case the two orders are equivalent. □

5.4. **Sufficiency and Limitations.** In the following, we discuss why the above necessary conditions are not sufficient for ensuring the correctness of PAGE. The following is a partial converse of Proposition 1.

Proposition 2. *Let G be a graph and $\mathcal{F}$ be a collection of closed trails satisfying the conditions of Proposition 1. If every vertex v of G has $\deg(v) \leq 5$ then $\mathcal{F}$ is realizable as a set of facial walks of some rotation system of G. Additionally, if some vertex v of G has $\deg(v) > 5$ the conditions in Proposition 1 are not sufficient.*

Proof. Fix a vertex v of degree d and label its incident edges $e_1, \ldots, e_d$. Traverse the closed trails in $\mathcal{F}$. Each time a closed trail enters v along a directed edge $e_i = (u, v)$ and leaves along $e_j = (v, w)$, we say that the ordered pair (e_i, e_j) *occurs* at v. This defines a map

$$\pi_v : \{e_1, \ldots, e_d\} \to \{e_1, \ldots, e_d\}, \qquad \pi_v(e_i) = e_j \quad \text{if } (e_i, e_j) \text{ occurs at } v.$$

We now show that π_v is a cyclic permutation of the incident edges of v. Let $e_j = \{v, w\}$. Since each directed edge (v, w) occurs in exactly one closed trail by Proposition 1, there exists some trail entering v along (u, v) and exiting along (v, w). Let $e_i = \{u, v\}$. Then (e_i, e_j) occurs at v, and so $\pi_v(e_i) = e_j$. Thus, π_v is surjective. As the set $\{e_1, \ldots, e_d\}$ is finite π_v is also injective, so it is a bijective map from $\{e_1, \ldots, e_d\}$ to itself and hence is a permutation of $\{e_1, \ldots, e_d\}$. To show that π_v has no fixed points, suppose for contradiction that

$\pi_v(e_i) = e_i$. Then (e_i, e_i) occurs at v, meaning some trail passes through v along (u, v) and immediately backtracks along (v, u), contradicting the non-backtracking condition. To show that π_v has no 2-cycles, suppose $\pi_v(e_i) = e_j$ and $\pi_v(e_j) = e_i$ for $e_i \neq e_j$. Let $e_i = \{u, v\}$ and $e_j = \{w, v\}$. Then there exist two trails c_1 and c_2 such that c_1 contains $(u, v), (v, w)$ and c_2 contains $(w, v), (v, u)$, violating condition (2) of Proposition 1. Suppose that π_v decomposes into $k > 1$ disjoint cycles $\pi_v = C_1 \circ C_2 \circ \cdots \circ C_k$. Since there are no fixed points or transpositions, each C_i has length at least 3, so

$$d \geq \sum_{i=1}^{k} \text{Length}(C_i) \geq 3k \quad \Rightarrow \quad k \leq \left\lfloor \frac{d}{3} \right\rfloor.$$

But $d \leq 5$, so $k = 1$. Therefore, π_v is a cyclic permutation. Defining π_v at each vertex $v \in V$ gives a rotation system for G, and by construction, the facial walks of this rotation system are precisely the closed trails in $\mathcal{F}$. In Figure 4 the conditions of Prop. 1 are met, yet the closed trails cannot be realized by a rotation system. □

6. Construction of the Algorithm

6.1. **Reducing the Search Space.** We reduce the search space by pruning all closed trail combinations that don't satisfy the necessary conditions to be realized by a rotation system of G. We consider collections that use each directed edge exactly once (Lem. 2), pass through each vertex according to its degree (Lem. 4), use exactly $2m$ directed edges (Lem. 5), and are realizable by a rotation system for vertex degrees less than 6 (Prop. 1 and 2). The `PotentialMaxFit` procedure applies these lemmas to efficiently reject unrealizable closed trails early. Lemma 3 indicates the need to filter out a subset of the closed trails. First, `PotentialMaxFit` discards sets with duplicate directed edges (Lem. 2). It tracks vertex usage (Lem. 4) and rejects overused vertices. Next, `PotentialMaxFit` ensures the sum of trail lengths is at most $2m$ (Lem. 5) and rejects collections with conflicting subpaths (Prop. 1 and 2). These conditions allow us to prune unrealizable candidates early.

6.2. **Ensuring Sufficiency of the Conditions.** The final step ensures we only consider collections of closed trails realizable by a rotation system. We extend `PotentialMaxFit` to reject if the local rotation at any vertex splits into disjoint cycles (Prop. 2). This can be done efficiently by updating the partial cyclic orderings at each vertex each time we consider a new closed trail in our candidate set. This suffices, as cyclic orderings at each vertex define a rotation system.

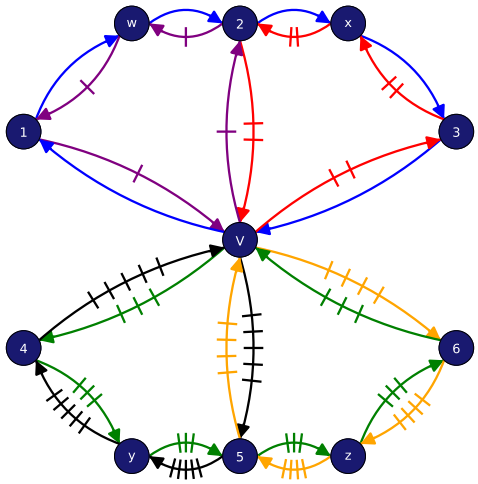


Figure 4. Degree 6 example showing Proposition 1 is not sufficient.

6.3. **Optimization via Backtracking.** At this point, we could finish a naive implementation of the algorithm by iterating through all closed trail combinations and rejecting any that don't satisfy `PotentialMaxFit`. Closed trails can be enumerated using Algorithm 2 by going through each length $t, t+1, \ldots, 2m$. Yet, before implementing this we can still make some easy optimizations. We know that if (a, b) occurs in some closed trail c_1 in a realizable set, then (b, a) occurs in some other closed trail c_2 in the same set. We can thus avoid searching all closed trail combinations including c_1 if we show that `PotentialMaxFit` rejects all possible c_2 trails when c_1 is part of the set for any (a, b).

We do this using a recursive backtracking algorithm that picks an edge e at each step that it knows must be part of a realizable set. It breaks ties by choosing in the way that minimizes the number of closed trails to search. When the candidate set is empty, this is the edge contained in the minimum number of closed trails so the search only has to consider a small number of starting trails which significantly reduces the branching factor at the top of the recursion tree and eliminates redundant exploration. When the candidate set is nonempty, this is the reverse edge of some edge in some closed trail in the set as long as the reverse edge is not in any closed trail in the set. We loop through all such reverse edges to choose the one contained in the minimum number of closed trails that have not been ruled out. We define `RequiredEdge` to be the function that computes this edge e given a realizable set.

The backtracking algorithm backtracks when it has tried all the closed trails that contain the edge chosen at the current step and all of them are rejected by `PotentialMaxFit` because then the candidate set cannot be completed to a realizable set. The backtracking algorithm keeps adding closed trails to the candidate set until it reaches a full one with $2m$ edges by Lemma 5. By construction of `PotentialMaxFit`, this is guaranteed to be realizable using a rotation system. In order to stop searching when we first find such a realizable candidate set, we iterate over trail combinations in order of decreasing number of trails. Since genus is inversely related to the number of faces in an embedding, and each face corresponds to a closed trail, ordering by decreasing number of trails naturally prioritizes embeddings of lower genus. This allows the algorithm to halt as soon as a minimal-genus realization is found, while still guaranteeing the search finds the minimum genus over all valid rotation systems without having to search all of them. To facilitate iterating over trail combinations, we define `AllClosedTrailsWithEdge` to enumerate all the closed trails containing a given edge. All this being said, we can finally formalize these optimizations using `PotentialMaxFit`, `RequiredEdge`, and `AllClosedTrailsWithEdge` in Algorithm 1 which completes PAGE.

7. Formal Algorithms

Algorithm 1 PAGE - Calculate the Genus

```
1:  procedure SEARCH(G, candidate_set)
2:      required_edge ← REQUIREDEDGE(G, candidate_set)
3:      trails_to_check ← ALLCLOSEDTRAILSWITHEDGE(G, required_edge)
4:      for each trail in trails_to_check do
5:          candidate_set ← candidate_set ∪ trail
6:          if PotentialMaxFit(candidate_set) rejects then
7:              candidate_set ← candidate_set \ trail
8:              continue
9:          end if
10:         if candidate_set contains 2m directed edges then
11:             return (V − E + |candidate_set| − 2)/(−2)
12:         end if
13:         recurse ← SEARCH(G, candidate_set)
14:         if recurse ≠ −1 then
15:             return recurse
16:         else
17:             candidate_set ← candidate_set \ trail
18:             continue
19:         end if
20:     end for
21:     return -1
22: end procedure
23: SEARCH(G, ∅)
```

7.1. **Formal Pseudocode of PAGE.** This is the PAGE algorithm as detailed in §6. On line 6, we reject all unrealizable candidate sets. On line 10, we check if we have completed a realizable candidate set and if we do, we calculate the genus using Euler's formula and bubble up the result through the search tree on line 11. On line 13, we keep adding more closed trails to the candidate set recursively. On line 14 and 17, we reject candidate sets that become unrealizable for all possible trails we could add further down the search tree. On line 15, we bubble up realizable candidate sets from further down the search tree if found. On line 21, we've checked all closed trails and all of them reject so we backtrack.

Algorithm 2 k-trail Finding Algorithm

```
1: Input: Graph G = (V, E) and length k
2: Output: Sequence of length-k closed trails of G
3: closed_trails ← ∅
4: queue ← FIFO queue with each single vertex path
5: while queue is not empty do
6:     path ← dequeue(queue)
7:     if len(path) = k then
8:         if first vertex of path is a neighbor of last vertex then
9:             if last vertex is not the same as second vertex then
10:                if directed edge from last to first vertex is not a repeat then
11:                    closed_trails ← closed_trails ∪ path
12:                end if
13:            end if
14:        end if
15:    else
16:        for each neighbor v of last vertex in path do
17:            if len(path) > 2  & v is the second to last vertex of path then
18:                continue
19:            end if
20:            if directed edge from last vertex to v is a repeat then
21:                continue
22:            end if
23:            if v ≥ first vertex of path then
24:                enqueue(queue, path ∪ {v})
25:            end if
26:        end for
27:    end if
28: end while
```

7.2. **Formal Pseudocode of k-trail Finding Algorithm.** Since a realizable rotation system corresponds to a collection of closed trails whose lengths sum to $2m$ (Lemma 5), we need only generate closed trails of lengths that could feasibly appear in such a partition of the directed edge set. Moreover, if t is the girth of G, then every non-backtracking closed directed trail has length at least t, since every such trail contains a cycle. Thus Algorithm 2 is only called for relevant lengths k with $t \leq k \leq 2m$, and in practice one may further restrict to lengths that can occur in a partition of $2m$. We adapt the algorithm by Liu et al. [13] for enumerating simple cycles. It is advantageous for its simplicity, easily parallelized form, and ability to generate all cycles of a given length k efficiently without keeping other cycles in memory or doing extensive computation on the graph beforehand. It is easily extended to closed trails as seen in the pseudocode. Line 8 ensures that the path forms a closed walk by checking that the last vertex is adjacent to the first. Line 9 prevents immediate reversals (backtracking) by ensuring that the last edge added does not reverse the previous one. Line 10 enforces that no directed edge is repeated within a trail. The condition in Line 23, requiring $v \geq$ the first vertex of the path, ensures that each cycle is only enumerated once up to rotation. We define the ordering on the vertices arbitrarily as long as its fixed for the given graph.

8. Correctness and Output

Theorem 7. *PAGE takes any graph G, calculates its genus, and produces the faces for an embedding of G on a minimal genus surface S.*

Proof. As shown in §5 and §6, PAGE discards sets of closed trails using necessary and sufficient criteria until it ends up with a realizable set of closed trails of maximum size. These closed trails are the facial walks which form the polygonal disc faces that when glued along shared edges and connected at shared vertices form S. □

This allows a "proof certificate" to verify that the genus outputted by PAGE is no less than the minimum genus and is how we produced Figure 1b and the below color-coded faces of the minimum genus embedding of the Balaban (3, 10)-cage.

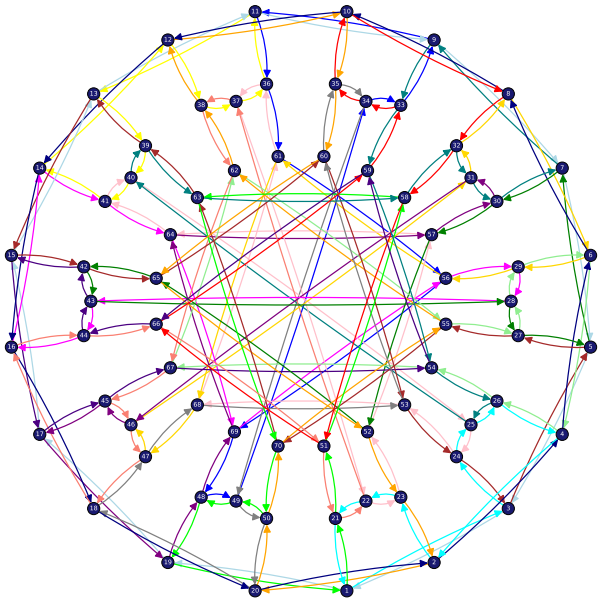


Figure 5. Genus 9 Embedding of the Balaban 10 Cage.

Theorem 8. *PAGE runs in $\mathcal{O}(n(4^m/n)^{n/t})$ time for graphs of girth t.*

Proof. The algorithm has some optimizations that allow it to stop early not captured in the below analysis, but it still holds as an upper bound on the runtime: the number of directed trails of any length is $\mathcal{O}(4^m)$ because it is the sum of the values in the $2m$-th row of Pascal's triangle. This can also be used to upper bound the number of closed trails which is what Algorithm 2 enumerates. Let c be the number of trails enumerated by our trail finding algorithm. Organizing the c trails by vertex is $\mathcal{O}(2mc) < \mathcal{O}(2m4^m)$ since each trail has at most $2m$ edges. Choosing the required edge at each step that minimizes the branching factor can be done in $\mathcal{O}(n)$ by iterating through the vertices to find the most used ones. Looking up the trails that use the edge can be done in $\mathcal{O}(1)$ time via hashset lookup by the endpoints of the edge. Checking if a trail is used can be done in $\mathcal{O}(1)$ via hashset lookup. Checking if the edges of a trail are used is $\mathcal{O}(m)$ by storing the edges used in a hashset. Checking if adding a trail makes the rotation system unrealizable can be done in $\mathcal{O}(m)$ by storing the current rotation and using `PotentialMaxFit`. Let f be the number of closed trails needed to have $2m$ edges in the candidate set, b the number of trails by vertex, $u \leq b$ the number of unused trails by vertex, $a \leq u$ the number of unused trails with edges available, and $d \leq a$ the ones where the candidate set is realizable. Then each `Search` iteration satisfies $T(f) = \mathcal{O}(n) + \mathcal{O}(1) + b\mathcal{O}(1) + u\mathcal{O}(m) + a\mathcal{O}(m) + dT(f-1)$ and $T(0) = 0$. This implies $T(f) = \mathcal{O}(d^f \cdot (n + b + um + am)) = \mathcal{O}(d^f \cdot (n + b + m(u + a))) < \mathcal{O}(b^{f+2})$. Let $h < 4^m$ be the number of start trails to try out. Then the total time of all `Search` iterations is $\mathcal{O}(b^{f+2} \cdot h) < \mathcal{O}((4^m/n)^{n/t} \cdot 4^m) = \mathcal{O}(n(4^m/n)^{n/t})$. □

8.1. **Proof of Theorem 6.** When feasible, we search in decreasing order on the number of closed trails, equivalently increasing genus, so we can stop as soon as a realizable candidate set is achieved. Searching in this order may require too much computation before reaching a realizable candidate set, especially if there are too many closed trails to iterate through to identify the greatest-cardinality candidate sets as in §4.4.

In this case, we can instead iterate in increasing order of maximum closed-trail length. More precisely, for $\ell = t, t+1, \ldots, 2m$, we search using only closed trails of length at most ℓ. If no realizable candidate set exists using only trails of length at most $\ell - 1$, then any remaining realizable candidate set must contain at least one closed trail of length at least ℓ. Since every closed trail has length at least t, such a candidate set has cardinality

$$F \leq 1 + \left\lfloor \frac{2m - \ell}{t} \right\rfloor.$$

By Euler's formula, this gives the lower bound

$$g \geq \left\lceil \frac{m - n + 2 - \left(1 + \left\lfloor \frac{2m-\ell}{t} \right\rfloor\right)}{2} \right\rceil.$$

This produces a non-decreasing lower-bound sequence for the genus which eventually converges to g in finitely many iterations. We now prove Theorem 6.

Proof. Proof. By Euler's formula [9], if F is the number of facial walks, then

$$g = \frac{m - n + 2 - F}{2}.$$

Since every facial walk has length at least the girth t and the facial walks together use all $2m$ directed edges, we have $tF \leq 2m$. Hence

$$g \geq \frac{m - n + 2 - \frac{2m}{t}}{2} = \frac{2t - nt + m(t-2)}{2t}.$$

Thus we may take the initial lower bound to be

$$g_0 = \max\left\{0, \left\lceil \frac{2t - nt + m(t-2)}{2t} \right\rceil\right\}.$$

For the initial upper bound, the maximum possible orientable genus is bounded above by the one-face Euler characteristic bound, so we may take

$$G_0 = \left\lfloor \frac{m - n + 1}{2} \right\rfloor.$$

Thus $g_0 \leq g \leq G_0$.

While PAGE searches it keeps the current bounds $g_{\text{current}} \leq g \leq G_{\text{current}}$. As PAGE iterates it may perform one of two updates,

- (*New Upper Bound*) When a candidate set of closed trails is formed containing all $2m$ directed edges, the previous call to PotentialMaxFit has already verified that it is realizable as a rotation system, so that each closed trail can be regarded as a facial walk of some rotation system. Then PAGE calculates the genus of the surface this rotation system embeds G on, yielding $G_{\text{new}} = \frac{1}{2}(m - n + 2 - F)$ where F is the amount of facial walks. If $G_{\text{new}} < G_{\text{current}}$ then we have a new upper bound and we append it to the upper-bound sequence $G_{s+1} := G_{\text{new}}$.
- (*New Lower Bound*) $g_{\text{current}} = g_i$ follows directly from the iteration i as previously defined.

Each update preserves $g_{\text{current}} \leq g \leq G_{\text{current}}$ and we have already shown that g_{current} converges to g in a finite number of steps. The same is true for G_{current}, since in $\mathcal{O}(m)$ iterations we will also have explored all closed trail combinations and by the correctness of PAGE have found a realizable candidate set with genus g. Thus PAGE halts with $g_i = g = G_j$ in a finite number of steps. □

9. Remarks and Comments

9.1. **Extensions.** As shown in various examples throughout this paper, determining the genus of a given graph is a longstanding problem in graph theory. Historically, the process of determining a graph's genus has often required extensive research and time, specifically focused on the individual graph in question. PAGE offers a new, practical, and fast method for calculating the genus of any graph. PAGE is also easily amenable to further optimization when specific information about a graph is known. For example, for large graphs, PAGE could be integrated with a computer algebra system to automate optimizations based on the graph's automorphism group. For instance, a graph like the Tutte-Coxeter (3,8) cage has the property that any path of up to 5 edges is equivalent to any other path up to automorphism [7]. PAGE could leverage properties like this to select larger initial segments of closed trails while searching and even further reduce the search

space if needed. Additionally, PAGE has the potential to answer many open conjectures in graph theory and advance the problem of completing the list of forbidden toroidal minors and indeed the sets of forbidden minors for surfaces of higher genus [18]. PAGE runs on large enough graphs to be useful for a number of applications such as designing circuit boards and microprocessors, roads and railway tracks, irrigation canals and waterways, quantum physics, and more.

9.2. **Runtime comparisons.** The purpose of this section is to do an experimental comparison of the runtime of PAGE with SAGEMATH and MULTI_GENUS when computing the genus of the 3-regular Cage graphs (Sec. 4.1) and the complete and complete-bipartite graphs (Sec. 3). We can see that PAGE takes advantage of the high girth of the Cage graphs to scale exceptionally well. PAGE is also competitive on the complete and complete-bipartite graphs even though they have low fixed girths of 3 and 4 respectively.

k	g	v	e	genus	PAGE (s)	SAGEMATH (s)	MULTI_GENUS (s)
3	3	4	6	0	0.008	0.004	0.006
3	4	6	9	1	0.008	0.039	0.006
3	5	10	15	1	0.008	0.027	0.006
3	6	14	21	1	0.008	0.010	0.006
3	7	24	36	2	0.010	1.737	0.006
3	8	30	45	4	0.032	118.958	0.012
3	9	58	87	7	1.625	DNF	45.099
3	10	70	105	9	39.211	DNF	9863.72
3	12	126	189	17	254.45	DNF	DNF

Table 2. Genus and time measurements (in seconds) for Cage graphs.

k	g	v	e	genus	PAGE (s)	SAGEMATH (s)	MULTI_GENUS (s)
2	3	3	3	0	0.005	0.004	0.008
3	3	4	6	0	0.005	0.003	0.008
4	3	5	10	1	0.005	0.005	0.008
5	3	6	15	1	0.005	0.023	0.008
6	3	7	21	1	0.005	DNF	0.009
7	3	8	28	2	2.219	DNF	0.008

Table 3. Genus and time measurements (in seconds) for complete graphs.

k	g	v	e	genus	PAGE (s)	SAGEMATH (s)	MULTI_GENUS (s)
3	4	6	9	1	0.005	0.047	0.006
4	4	8	16	1	0.005	0.010	0.010
5	4	10	25	3	0.014	DNF	0.008
6	4	12	36	4	0.011	DNF	0.009

Table 4. Genus and time measurements (in seconds) for complete bipartite graphs.

REFERENCES

1. Dan Archdeacon, *A kuratowski theorem for the projective plane*, Journal of Graph Theory **5** (1981), no. 3, 243–246.
2. Stephan Beyer, Markus Chimani, Ivo Hedtke, and Michal Kotrbčík, *A practical method for the minimum genus of a graph: Models and experiments*, Experimental Algorithms (Cham) (Andrew V. Goldberg and Alexander S. Kulikov, eds.), Springer International Publishing, 2016, pp. 75–88.
3. Gunnar Brinkmann, *A practical algorithm for the computation of the genus*, Ars Mathematica Contemporanea **22** (2022), no. 4.
4. John Chambers, *Hunting for torus obstructions*, 2002.
5. Markus Chimani and Tilo Wiedera, *Stronger ILPs for the Graph Genus Problem*, 27th Annual European Symposium on Algorithms (ESA 2019) (Dagstuhl, Germany), vol. 144, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2019, pp. 30:1–30:15.
6. Marston Conder and Ricardo Grande, *On embeddings of circulant graphs*, Electronic Journal of Combinatorics **22** (2015), no. 2.
7. Harold Scott MacDonald Coxeter, *Twelve points in pg(5, 3) with 95040 self-transformations*, Proceedings of the Royal Society of London. Series A, Mathematical and Physical Sciences **247** (1958), no. 1250, 279–293.

8. SageMath Developers, *Genus calculation in sagemath*, 2024.
9. Leonhard Euler, *Elementa doctrinae solidorum*, *Euler Archive - All Works*, no. 230, 1758.
10. Lothar Heffter, *Ueber das problem der nachbargebiete*, Math. Ann. **38** (1891), 477–508.
11. Mark Jungerman and Gerhard Ringel, *The genus of the n-octahedron: Regular cases*, Journal of Graph Theory **2** (1978), no. 1, 69–75.
12. Casimir Kuratowski, *Sur le problème des courbes gauches en topologie*, Fundamenta Mathematicae **15** (1930), no. 1, 271–283.
13. Hongbo Liu and Jiaxin Wang, *A new way to enumerate cycles in graph*, Advanced Int'l Conference on Telecommunications and Int'l Conference on Internet and Web Applications and Services (AICT-ICIW'06), 2006, pp. 57–57.
14. Dragan Marušič, Tomaž Pisanski, and Steve Wilson, *The genus of the gray graph is 7*, European Journal of Combinatorics **26** (2005), no. 3, 377–385.
15. Bojan Mohar, *A linear time algorithm for embedding graphs in an arbitrary surface*, SIAM Journal on Discrete Mathematics **12** (1999), no. 1, 6–26.
16. Bojan Mohar and Carsten Thomassen, *Graphs on surfaces*, Johns Hopkins University Press, 2001.
17. Wendy Myrvold and William Kocay, *Errors in graph embedding algorithms*, Journal of Computer and System Sciences **77** (2011), no. 2, 430–438.
18. Wendy Myrvold and Jennifer Woodcock, *A large set of torus obstructions and how they were discovered*, Electronic Journal of Combinatorics **25** (2018), no. 1.
19. Gerhard Ringel, *Bestimmung der maximalzahl der nachbargebiete auf nichtorientierbaren flächen*, Math. Ann. **127** (1954), 181–214.
20. Gerhard Ringel, *Das geschlecht des vollständigen paaren graphen*, Abh.Math.Semin.Univ.Hambg. **28** (1965), 139–150.
21. Gerhard Ringel and John William Theodore Youngs, *Solution of the heawood map-coloring problem**, Proceedings of the National Academy of Sciences **60** (1968), no. 2, 438–445.
22. Neil Robertson and Paul Seymour, *Graph minors .xiii. the disjoint paths problem*, Journal of Combinatorial Theory, Series B **63** (1995), no. 1, 65–110.
23. Neil Robertson and Paul Seymour, *Graph minors. xx. wagner's conjecture*, Journal of Combinatorial Theory, Series B **92** (2004), no. 2, 325–357.
24. Carsten Thomassen, *The graph genus problem is np-complete*, Journal of Algorithms **10** (1989), no. 4, 568–576.
25. Eric Weisstein, *Higman–sims graph*, https://mathworld.wolfram.com/Higman-SimsGraph.html, 2024a, From MathWorld – A Wolfram Web Resource.
26. Eric Weisstein, *Cyclotomic graph*, https://mathworld.wolfram.com/CyclotomicGraph.html, 2024b, From MathWorld – A Wolfram Web Resource.
27. Eric Weisstein, *Bipartite kneser graph*, https://mathworld.wolfram.com/BipartiteKneserGraph.html, 2024c, From MathWorld – A Wolfram Web Resource.
28. Eric Weisstein, *Johnson graph*, https://mathworld.wolfram.com/JohnsonGraph.html, 2024d, From MathWorld – A Wolfram Web Resource.

University of Washington
Email address: metzgera@uw.edu

University of Washington
Email address: austinul@uw.edu