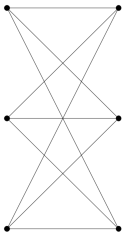
AN EFFICIENT GENUS ALGORITHM BASED ON GRAPH ROTATIONS

ALEXANDER METZGER AND AUSTIN ULRIGG

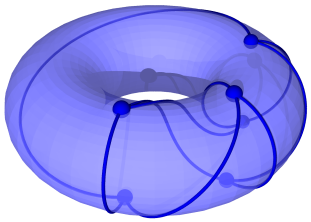
ABSTRACT. We study the problem of determining the minimal genus of a given finite connected graph. We present an algorithm which, for an arbitrary graph G with n vertices, determines the orientable genus of G in $\mathcal{O}(2^{(n^2+3n)}/n^{(n+1)})$ steps. This algorithm avoids the difficulties that many other genus algorithms have with handling bridge placements which is a well-known issue [38]. The algorithm has a number of properties that make it useful for practical use: it is simple to implement, it outputs the faces of the optimal embedding, it outputs a proof certificate for verification and it can be used to obtain 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 AND MAIN RESULT

1.1. **Introduction.** 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 [27] tells us that it is not. However, $K_{3,3}$ is *toroidal*, meaning 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

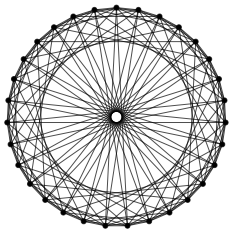
The characterizing property of a torus that allows us to embed $K_{3,3}$ is that it has a hole (unlike surfaces such as a plane or a sphere). This motivates classifying surfaces by their number of holes, that is, their genus g. In these terms, we have seen that the minimum genus surface that $K_{3,3}$ can be embedded has $g = 1$, and we say that $K_{3,3}$'s genus is 1. In general, a connected multigraph $G(V, E)$ can be embedded on an orientable surface S of genus g if G can be drawn on S without any edges crossing. We say that g is the genus of a graph G if g is the minimum genus surface on which G can be embedded. For genus zero we use the special name "planar" and for genus one we use "toroidal". Similarly, we have that the complete

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

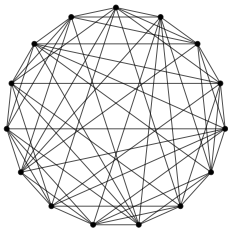
graph with 7 vertices, K_7, has genus 1 and can be embedded on a torus. However, K_8 cannot be embedded on a torus, and has genus 2. Ringel [40, 41] determined the minimum non-orientable genus for the complete graph K_n and also the orientable and non-orientable genus for the complete bipartite graph $K_{m,n}$. Further, Ringels and Youngs later determined the minimum orientable genus for K_n. [42].

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

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,



(A) Cyclotomic 31 Graph ($12 \leq g \leq 32$)



(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 electronic circuitry.

1.2. **Main Result.** We present a simple algorithm to determine graph genus, Practical_Algorithm_for_Graph_Embedding (PAGE). The algorithm runs faster than previously implemented algorithms including those presented in [2, 16, 48]. 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 that it iteratively narrows as it is processing, further enabling practical use cases.

Theorem 1 (Main Result)**.** *PAGE described in §4 determines the genus of a connected multigraph $G(V, E)$ with runtime of $\mathcal{O}(2^{(n^2+3n)}/n^{(n+1)})$.*

We emphasize that PAGE is relatively easy to implement; moreover §2 contains a number of concrete examples where the algorithm is used to determine the genus of graphs whose genus was previously unknown.

1.3. **Prior Results.** In 1963, Youngs established the fundamental principle of graph rotation theory, demonstrating that any embedding of a connected graph can be fully determined by the rotation of edges at its vertices [53]. For a vertex of degree k there are $(k-1)!$ distinct rotations of edges at that vertex and thus for a k-regular graph of n vertices there are $(k-1)!^n$ total embeddings into an orientable surface. It has been well established that the problem of determining the genus is NP-hard [50]. However, it is tractable for fixed genus [36, 38, 46]. Mohar has proven the existence of a linear time algorithm for arbitrary fixed surfaces [36]. However, the details to actually implement the algorithm have evaded the field for many decades. Similar efficient algorithms are more of theoretical interest than practical use given their large constant run-time factors, super-exponential

scaling factors when genus is not kept fixed, and immense complexity that has prevented real-world implementation. The forbidden minor approach by Robertson and Seymour [46], for instance, is theoretically cubic time because it checks if a given graph has one of the finitely many forbidden minors for the fixed surface [47]. However, the complete list of forbidden minors is only known for planar [27] and projected plane graphs [44]. 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 [25]. In practice, the finite number of graph minors scales at an impractical super-exponential rate with the genus. This makes it intractable to compute all the minors (months of super-compute has been thrown at it to no avail). As it stands, the best implemented algorithms are exponential time [38, 11, 25] and the rest are too complex/impractical for implementation and have intractable constant factors.

Recently, an algorithm called MULTI_GENUS by Gunnar Brinkmann has emerged as a particularly fast method for computing genus on graphs with relatively low genus compared to the vertex degree [4]. It outperforms many previous algorithms, including ours, for graphs where vertex degrees exceed 5, scaling more effectively with edge count and vertex degree. However, our algorithm remains advantageous for graphs with vertices of degree 5 or lower. Additionally, our approach is comparatively simpler to implement and scales more efficiently with the genus, offering an advantageous alternative in cases where the genus is large. 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.

The other current best algorithms [2, 16, 48] outside of MULTI_GENUS 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 takes many hours. K_8 and above is almost entirely out of reach. Their run-times are double exponential in the genus, or in terms of the number of vertices, $\mathcal{O}(n(n-1)!^n)$.

2. Examples

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

2.1. Circulant and Complete Multipartite Graphs.

A 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 [21]. The complete n-partite graph represents the problem of how many handshakes are possible in a room of n people and has many applications in combinatorics. It is known that $K_{2,2,\ldots,2}$ is isomorphic to the circulant graph $Ci_{1,2,\ldots,n-1}(2n)$. The genus is also known for all circulant graphs with genus 1 and 2 [5]. 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 2 (Circulants)**.** *The genus of* $C14_{1,2,3,6}$ *is 4. The genus of* $C18_{1,3,9}$ *is 4. The genus of* $C20_{1,3,5}$ *is 6. The genus of* $C20_{1,6,9}$ *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 [21]).

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

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

Significant interest has surrounded the genus of the Gray graph (it happens to be 7), which has been addressed in a dedicated study [30]. 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.

3. Progressive Refinement of Genus Bounds

Some graphs are so large it is not feasible to compute their exact genus. In practice, it is often also not needed to find an exact genus. For cases where it suffices to have an embedding within some error tolerance of the fewest holes, PAGE outputs genus bounds that narrow with increasing iterations.

Theorem 4. *For any connected multigraph $G(V,E)$ with m edges, n vertices, and unknown non-fixed genus g, PAGE computes two integer sequences g_k and G_k satisfying*

$$\frac{m-4n/3+2}{2} = g_0 \leq g_1 \leq \cdots \leq g \leq \cdots G_1 \leq G_0 = \frac{m-n+2}{2}$$
$$G_k - g_k > G_{k+1} - g_{k+1}$$

with g_k, G_k computed in runtime $\mathcal{O}(((n-1)!/k/(n-k)!)^{n/3-k})$ for $k>0$ and in constant time using the formulas for $k=0$. In other words, the sequence of lower bounds g_k and sequence of upper bounds G_k converge to the genus g after $\ell/6$ iterations with intermediate absolute error bounded by $\ell/6-k$.

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 7813 before encountering memory limitations.

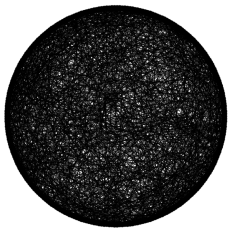
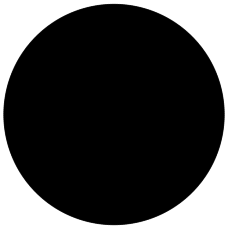
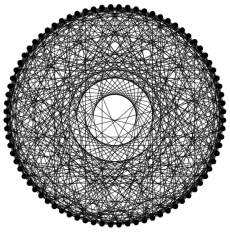


FIGURE 3. (6,12) Cage Graph $5860 \leq g \leq 7813$

To show the approach generalizes well, we applied it to large graph families from earlier in the paper, establishing these genus bounds within 15 minutes.

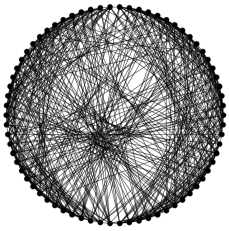


(A) Bipartite Kneser Graph (12, 3)
Genus [4401, 9021]

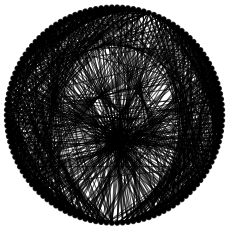


(B) DifferenceSetIncidenceGraph (40, 13, 4)
Genus [91, 221]

FIGURE 4. Bipartite Kneser and Difference Set Incidence Bounds

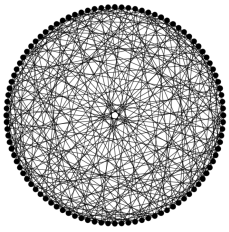


(A) Johnson Graph (8,4)
Genus [60, 246]

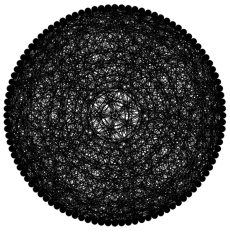


(B) Johnson Graph (9, 4)
Genus [148, 568]

FIGURE 5. Bounds for Johnson Graphs

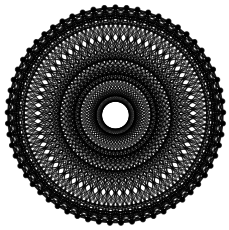


(A) Hoffman Singleton BipartDoubleGraph
Genus [68, 100]



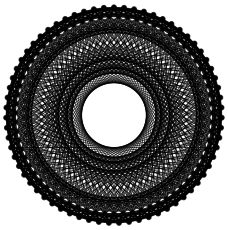
(B) Higman Sims Graph
Genus [226, 501]

FIGURE 6. Bounds for Hoffman Singleton and Higman Sims



(A) Cyclotomic Graph 61

Genus [73, 275]



(B) Cyclotomic Graph 67

Genus [90, 336]

FIGURE 7. Bounds for Cyclotomic Graphs

4. Proof of the Algorithm

We construct PAGE with formal pseudocode. The general idea is Euler's formula $n-m+F=2-2g$ which links the facial walks F with the genus g [13]. Naively, finding the minimum genus amounts to searching all combinations of cycles to find the one that corresponds to a maximal fitting of facial walks. Traditional improved exhaustive search algorithms instead search through the rotation systems since they each induce a valid facial walk whereas not all cycle combinations are valid. 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 all cycle combinations which facilitates a number of optimizations (early stopping, heuristic search) and, with our main contribution, still allows us to prune invalid rotation systems which results in an exponentially reduced search space.

Lemma 5. *Given a graph G with finite n vertices and finite m undirected edges, knowing the maximum number of facial walks F yields the minum genus g.*

Proof. This is a direct consequence of Euler's characteristic formula. [13] □

Lemma 6. *Given a graph G with finite n vertices and finite m undirected edges, the set of simple cycles C of G is finite and any valid set of facial walks of G is a subset of C.*

Proof. The finite size of C follows from the finite combinations of up to n vertices and that a simple cycle cannot repeat a vertex and must therefore contain at most n vertices. Since each facial walk in a set of valid facial walks of G is a simple cycle, the lemma is clear. □

By Lemma 6, we need to filter out a subset of the simple cycles. Finding all the simple cycles of a graph, can be done in $O((c+1)(n+m))$ where c is the number of cycles using Johnson's Algorithm [20]. Since we do not need to generate all the cycles, only the ones of a few lengths, the algorithm by Liu et al. [28] 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.

Lemma 7. *Given a graph G with finite n vertices and finite m undirected edges, choose any vertex $v \in V$ of degree d and all valid set of facial walks must include v exactly d times.*

Algorithm 1 k-Cycle Finding Algorithm

```
Input: Graph G = (V, E) and length k
Output: Sequence of length-k cycles of G
cycles ← ∅
queue ← FIFO queue with each single vertex path
while queue is not empty do
    path ← dequeue(queue)
    if len(path) = k then
        cycles ← cycles ∪ path
    else
        for each neighbor v of last vertex in path do
            if v ∉ path and v > first vertex of path then
                enqueue(queue, path)
            end if
        end for
    end if
end while
```

Proof. A valid set of facial walks must use each directed edge of G exactly once. Since each facial walk is a cycle, it must use 2 of the directed edges that touch v or none of them. There are $2 \cdot d$ directed edges that touch v and thus $2 \cdot d/2 = d$ facial walks including v. □

By Lemma 7, we can annotate our adjacency list with the number of uses of each vertex and use that in a `PotentialMaxFit` procedure to reject cycle combinations early if they try to use a vertex more than its degree number of times.

Lemma 8. *Given a graph G with finite n vertices and finite m undirected edges, any valid set of facial walks must be a set of simple cycles whose lengths add up to the number of directed edges $2 \cdot m$.*

Proof. The length of a simple cycle is the number of unique edges it uses. A valid set of facial walks must by definition use each directed edge of G exactly once. □

An important way to prune the search space is to not blindly try all combinations of cycles and instead only try valid cycle combinations that could add up to use all the directed edges. Checking all valid cycle distributions to find the working one with the most cycles would then yield the genus as formalized in Lemma 8. In order to stop early when a working distribution is found, it is desired to iterate in order of decreasing number of cycles. Finding the cycle distribution with the most cycles is the bounded integer knapsack problem in disguise (each cycle has value 1, costs its length, and there is a bounded number of cycles of each length) so it is NP-hard by reduction. Various pseudo-polynomial solutions for this problem exist. In practice, iterating in sorted order induces too much of an overhead. We instead include pseudocode for iterating in increasing order of max cycle length. This also saves the overhead of computing all cycle lengths. It is trivial to sort the produced cycle distributions in linear time (linear in the number of cycle distributions) using bucket sort.

Algorithm 2 Cycle Distribution Generation

Input: Population `[(v, c)]` of pairs of c cycles of length v and s directed edges
array $\leftarrow$ [list of empty lists]
for each (v,c) in `population` **do**
  for $j \leftarrow 1$ **to** c **do**
    for $num \leftarrow s - j \cdot v$ **downto** 0 **do**
      if array[num] is not empty **then**
        for each subset in array[num] **do**
          if v is in subset **then**
            continue
          end if
          array[$num + j \cdot v$] $\leftarrow$ array[$num + j \cdot v$] $\cup$ (tuple with v repeated j times)
          if $num + j \cdot v = s$ **then**
            Yield last element of array[$num + j \cdot v$]
          end if
        end for
      end if
    end for
  end for
end for

Lemma 9. *Given a graph G with finite n vertices and finite m undirected edges and a set of k simple cycles whose lengths add up to the number of directed edges $2 \cdot m$, the corresponding genus if the cycles are a valid facial walk is $g = \max(0, 1 - (k - m + n)/2)$ using integer division.*

Proof. This is a direct consequence of Euler's characteristic formula [13]. □

We can augment our `PotentialMaxFit` procedure to rule out cycle combinations that do not correspond to a valid rotation system. By lemma 10, this can be done simply by annotating the adjacency list with the rotation (when an edge is used, indicate at each vertex, which of its neighbors it gets assigned to) or by looping through the used cycles to check for adjacent directed edges used in both directions.

Lemma 10. *Given a graph G with finite n vertices and finite m undirected edges, a valid set of facial walks is a set of simple cycles that use each directed edge exactly once and do not contain two cycles c_1, c_2 such that c_1 contains adjacent directed edges $(a,b),(b,c)$ and c_2 contains $(c,b),(b,a)$ for any vertex b with degree greater than 2.*

Proof. A facial walk must be a simple cycle. A valid set of facial walks, is 11 a valid rotation system. A valid rotation system is a permutation of the incident edges to each vertex determining that entering through one edge requires leaving through the next. A facial walk can never contain both edges (u,v) and (v,u) so $a \neq b$. If c_1 and c_2 exist, the degree of b must be 2 for the two vertices to be next to each other when reading the permutation in either order which is a contradiction. □

Remark 11. Note that for any vertex with degree greater than 2, the conditions outlined in Lemma 10 are necessary for the existence of a valid rotation system,

but they are not sufficient by themselves. Additional criteria must be satisfied to ensure that a valid rotation system can be constructed. Specifically, for vertices with degree greater than 5, an additional verification process is required to confirm that a valid rotation system can indeed be formed from the given set of facial walks.

We have now constructed the necessary procedures to write the algorithm:

Algorithm 3 Calculate the Genus

```
while not all cycle distributions have been tried do
  for each result in GetNextCycleDistribution() do
    if not PotentialMaxFit(result) then
      continue
    end if
    search(result)
    if fit works then
      stop early and calculate the genus using Lemma 9
    end if
  end for
end while
```

Theorem 12. *PAGE yields the correct genus.*

Proof. By the lemmas, each optimization still leads to finding the maximum number of facial walks and thus the minimum genus. □

The idea of the algorithm is to go through each cycle distribution (discarding ones that are not relevant to finding the maximum cycle distribution size, and thus the genus, because a cycle distribution with more cycles has already been found to work). For each cycle distribution, the algorithm then recursively picks a cycle until either the cycle distribution is fully used (thus all directed edges are used exactly once, and the distribution works) or any cycle length does not have cycles with unused edges available (thus the distribution could never use all unused edges exactly once, and does not work). Another optimization the algorithm makes is choosing cycles from the same vertex until that vertex is satisfied (one cycle for each incident un-directed edge used) since that is a necessary condition to all edges being used exactly once and allows failing early as soon as a vertex cannot be satisfied. The recursion depth is small since the maximum number of cycles that can fit is $O(m-n)$ and in most cases the algorithm will fail out of the recursion early.

Theorem 13. *PAGE takes any connected multigraph $G(V,E)$, calculates its genus, and produces the faces for an embedding of G on a minimal genus surface S.*

Proof. By the lemmas above, our algorithms find the facial walks to deduce the minimum genus. By definition, these facial walks form the polygonal disc faces that can be glued together at shared edges and folded to connect shared vertices in order to construct 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.

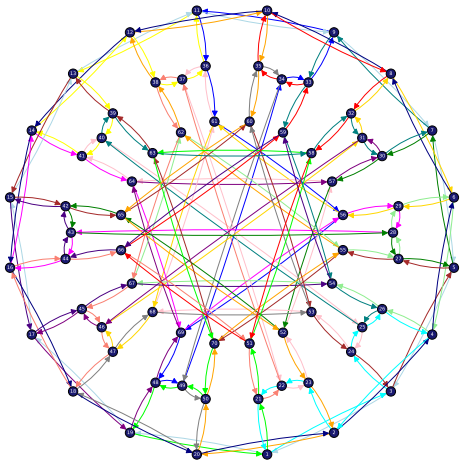


FIGURE 8. Genus 9 Embedding of the Balaban 10 Cage

Theorem 14. *PAGE is* $\mathcal{O}(2^{n^2+3}/n^{n+1})$ *for complete graphs.*

Proof. The algorithm has some optimizations that allows it to stop early not captured in the below analysis, but it still holds as an upper bound on the runtime:

(1) Finding all c elementary cycles of a graph: $\mathcal{O}(((c+1)(n+m)) = \mathcal{O}((cn+cm)$ using Johnson's algorithm.
(2) Organizing by vertex: $\mathcal{O}((cn)$ by iterating through the cycles of length at most n.
(3) Iterating through all cycles of length k: $\mathcal{O}((n!/k/(n-k)!)$ since this is the number of k-length cycles in a complete graph. This is worst case $\mathcal{O}((n!/((n/2)!)^2)$ which is roughly $\mathcal{O}((2^n)$.
(4) Find the most used vertex to explore: $\mathcal{O}((n)$ by iterating through the vertices (overall time complexity is better when write is kept $\mathcal{O}((1))$.
(5) Looking up the cycles that use a vertex: $\mathcal{O}((1)$ via hashset lookup.
(6) Checking if a cycle is used: $\mathcal{O}((1)$ via hashset lookup.
(7) Checking if the edges of a cycle are used: $\mathcal{O}((e)$ where e is the number of edges in the cycle by storing the edges used in a hashset.
(8) Checking ijk criterion of a cycle: $\mathcal{O}((e)$ by storing the current rotation with the adjacency list.
(9) Search iteration ($f =$ implied fit, $b =$ cycles by vertex $\geq u =$ unused $\geq a =$ w/ edges available $\geq d =$ ijk good): $T(f) = \mathcal{O}((n) + \mathcal{O}((1) + b\mathcal{O}((1) + u\mathcal{O}((e) + a\mathcal{O}((e) + dT(f-1); T(0) = 0 \implies \mathcal{O}((d^f \cdot (n+b+ue+ae)) = \mathcal{O}((d^f \cdot (n+b+e(u+a))) < \mathcal{O}((b^{f+2})$.
(10) All search iterations ($t =$number of start cycles to try out $< n2^n$): $\mathcal{O}((b^{f+2} \cdot t) < \mathcal{O}(((2^n/n)^{n+2} \cdot n2^n) = \mathcal{O}((2^{n^2+3n}/n^{n+1})$.

□

5. Remarks and Comments

5.1. **Extensions.** As demonstrated in various examples throughout this paper, determining the genus of a given graph has been a longstanding problem in graph

theory. Historically, the process of determining a graph's genus has often required extensive research and time, tailored specifically to the individual graph in question. PAGE offers not only a novel, practical, and efficient method for calculating the genus of an individual graph but also provides a general approach applicable to all graphs. Furthermore, for highly complex graphs, given the NP-hard nature of genus computation, PAGE can be combined with techniques that leverage the graph's automorphisms to determine its genus more effectively. PAGE is also amenable to further optimization when specific information about the graph is available. For example, integrating this method with a computer algebra system could automate optimizations based on the graph's automorphism group. 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 [39]. PAGE scales to large enough graphs to be useful for a number of applications: designing Printed Circuit Boards and microprocessors, roads and railway tracks, irrigation canals and waterways, visualizing large interconnected temporal or geographical data, chemistry, quantum physics, and more.

5.2. **Runtime comparisons.** The purpose of this short section is to quickly do a representative comparison of the runtime of PAGE with the one implemented in SAGEMath and multi_genus when computing the genus of the 3-regular cage graphs.

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	0.009	0.010	0.006
3	7	24	36	2	0.012	1.737	0.006
3	8	30	45	4	0.039	118.958	0.012
3	9	58	87	7	1.388	days	47.737
3	10	70	105	9	46.179	DNF	9354.14
3	12	126	189	17	319.63	DNF	days

Table 1. Genus and time measurements

References

1. A.T Balaban, *A trivalent graph of girth ten*, Journal of Combinatorial Theory, Series B **12** (1972), no. 1, 1–5.
2. J. B. Berndt and R. E. Scott, *A practical method for the minimum genus of a graph: Models and experiments*, Theoretical Computer Science **556** (2016), 80–91, Accessed: 2024-08-18.
3. P. Booth and G. L. Lueker, *Testing for the consecutive ones property, interval graphs, and graph planarity using pq-tree algorithms*, Journal of Computer and System Sciences **13** (1976), no. 3, 211–227, Accessed: 2024-08-18.
4. Gunnar Brinkmann, *A practical algorithm for the computation of the genus*, Ars Mathematica Contemporanea **22** (2022).
5. Marston Conder and Ricardo Grande, *On embeddings of circulant graphs*, Electronic Journal of Combinatorics **22** (2015).
6. Marston Conder and Klara Stokes, *New methods for finding minimum genus embeddings of graphs on orientable and non-orientable surfaces*, Ars Mathematica Contemporanea **17** (2019), 1–35.

7. J. B. T. Conn, *Projective planarity in linear time*, Journal of Graph Theory **5** (1979), no. 2, 159–168, Accessed: 2024-08-18.
8. S. A. Cook, *The graph genus problem is np-complete*, Computational Complexity **1** (1989), no. 2, 193–209, Accessed: 2024-08-18.
9. H. S. M. 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), 279–293.
10. M. Demoucron, *Graphes planaires: Reconnaissance et construction de representations planaires topologiques*, Accessed: 2024-08-18.
11. W. M. G. Dicks, *Hunting for torus obstructions*, Accessed: 2024-08-18.
12. Walther von Dyck, *Beiträge zur analysis situs*, Mathematische Annalen **32** (1888), 457–512.
13. Leonhard Euler, *Elements of the doctrine of solids*, Novi Commentarii academiae scientiarum Petropolitanae **4** (1758), 109–140 (Latin), Enestrom Number: 230, Fuss Index: 318.
14. M. T. Goodrich, *Depth-first search and planarity*, Accessed: 2024-08-18.
15. Ludwig Heffter, *Ueber das problem der nachbargebiete*, Mathematische Annalen **38** (1891), 477–508.
16. L. A. Hines, *Stronger ilps for the graph genus problem*, Lecture Notes in Computer Science **144** (2019), 134–148, Accessed: 2024-08-18.
17. J. Hopcroft and T. Tarjan, *An implementation of the hopcroft and tarjan planarity test and embedding algorithm*, Accessed: 2024-08-18.
18. J Hopcroft and T Tarjan, *Efficient planarity testing*, Journal of the ACM **21** (1974), no. 4, 549–568, Accessed: 2024-08-18.
19. J. K. P. John, *Depth-first search and kuratowski subgraphs*, Journal of Algorithms **10** (1989), no. 4, 560–578, Accessed: 2024-08-18.
20. Donald B. Johnson, *Finding all the elementary circuits of a directed graph*, SIAM Journal on Computing **4** (1975), no. 1, 77–84.
21. Mark Jungerman and Gerhard Ringel, *The genus of the n-octahedron: Regular cases*, Journal of Graph Theory **2** (1978), no. 1, 69–75.
22. D. M. Kahn and S. L. Young, *103 graphs that are irreducible for the projective plane*, Journal of Graph Theory **16** (1992), no. 3, 311–322, Accessed: 2024-08-18.
23. M. H. Kaplan and N. D. Smith, *A practical algorithm for embedding graphs on torus*, International Journal of Network Computing **10** (2008), no. 2, 125–136, Accessed: 2024-08-18.
24. M. E. P. Kobourov, *Non-euclidean spring embedders*, Accessed: 2024-08-18.
25. M. W. Kocay, *A large set of torus obstructions and how they were discovered*, Electronic Journal of Combinatorics **25** (2018), no. 1, Accessed: 2024-08-18.
26. T. W. Kuehn, *Simpler projective plane embedding*, Journal of Computational Geometry **20** (2005), no. 1, 78–90, Accessed: 2024-08-18.
27. C. G. Kuratowski, *Sur le problème des courbes gauches en topologie*, Fundamenta Mathematicae **50** (1962), no. 1, 5–15, Accessed: 2024-08-18.
28. 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.
29. G. L. Lueker, *A simpler linear time algorithm for embedding graphs into an arbitrary surface and the genus of graphs of bounded tree-width*, SIAM Journal on Computing **29** (2000), no. 3, 901–919, Accessed: 2024-08-18.
30. 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, Topological Graph Theory and Graph Minors, second issue.
31. R. E. McConnell and J. R. P. Stiebitz, *A kuratowski theorem for nonorientable surfaces*, Journal of Combinatorial Theory, Series B **53** (1991), no. 3, 378–390, Accessed: 2024-08-18.
32. R. J. McConnell and J. A. West, *Triangulating a surface with a prescribed graph*, Journal of Algorithms **7** (1983), no. 3, 340–348, Accessed: 2024-08-18.
33. J. P. W. Mehlhorn, *A simple test for planar graphs*, Accessed: 2024-08-18.
34. A. Mohar, *An algorithm for embedding graphs in the torus*, Journal of Graph Theory **14** (1990), no. 4, 453–464, Accessed: 2024-08-18.
35. A Mohar, *Embedding graphs in the torus in linear time*, Lecture Notes in Computer Science **456** (1990), 58–63, Accessed: 2024-08-18.

36. A. Mohar and C. Thomassen, *A linear time algorithm for embedding graphs in an arbitrary surface*, Journal of Graph Theory **15** (1991), no. 1, 1–16, Accessed: 2024-08-18.
37. Bojan Mohar and Carsten Thomassen, *Graphs on surfaces*, Johns Hopkins University Press, July 2001.
38. Wendy Myrvold and William Kocay, *Errors in graph embedding algorithms*, Journal of Computer and System Sciences **77** (2011), no. 2, 430–438, Adaptivity in Heterogeneous Environments.
39. Wendy Myrvold and Jennifer Woodcock, *A large set of torus obstructions and how they were discovered*, Electronic Journal of Combinatorics **25** (2018).
40. Gerhard Ringel, *Bestimmung der maximalzahl der nachbargebiete auf nichtorientierbaren flächen*, Mathematische Annalen **127** (1954), 181–214.
41. Gerhard Ringel, *Das geschlecht des vollständigen paaren graphen*, Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg **28** (1965), 139–150.
42. Gerhard Ringel and John W. T. Youngs, *Solution of the heawood map-coloring problem*, Proceedings of the National Academy of Sciences of the United States of America **60** (1968), 438–445.
43. N. Hartsfield G. Ringel, *Pearls in graph theory*, Academic Press Inc., 1990.
44. N. Robertson and P. D. Seymour, *A kuratowski theorem for the projective plane*, Journal of Graph Theory **10** (1986), no. 2, 259–270, Accessed: 2024-08-18.
45. N Robertson and P D Seymour, *Graph minors. viii. a kuratowski theorem for general surfaces*, Journal of Combinatorial Theory, Series B **42** (1987), no. 1, 69–83, Accessed: 2024-08-18.
46. N. Robertson and P. D. Seymour, *Graph minors xiii: The disjoint paths problem*, Journal of Combinatorial Theory, Series B **48** (1990), no. 1, 1–10, Accessed: 2024-08-18.
47. Neil Robertson and Paul Seymour, *Graph minors. xx. wagner's conjecture*, Journal of Combinatorial Theory, Series B **92** (2004), no. 2, 325–357.
48. SageMath, *Genus calculation in sagemath*, Accessed: 2024-08-18.
49. SageMath, *Sagemath generators*, https://doc.sagemath.org/html/en/reference/graphs/sage/graphs/graph_generators.html, Accessed: 2024-08-18.
50. Carsten Thomassen, *The graph genus problem is np-complete*, Journal of Algorithms **10** (1989), no. 4, 568–576.
51. Wikipedia, *Graph minor theorem*, Accessed: 2024-08-18.
52. Win.tue.nl, *Adjacency lists*, Accessed: 2024-08-18.
53. John W. T. Youngs, *Minimal imbeddings and the genus of a graph*, Journal of Mathematics and Mechanics **12** (1963), 303–315.
54. S. A. M. Zverovich, *Practical toroidality testing*, ACM Transactions on Algorithms **3** (2007), no. 1, 1–18, Accessed: 2024-08-18.

University of Washington
Email address: metzgera@uw.edu

University of Washington
Email address: austinul@uw.edu