QUADRANGULATIONS AND NONORIENTABLE GENERA: A THESIS ON THEORY AND RESULTS IN TOPOLOGICAL GRAPH THEORY

A thesis presented to the faculty of
San Francisco State University
In partial fulfilment of
The Requirements for
The Degree

Master of Science
In
Computer Science

by

Warren Singh

San Francisco, California

December 2023

CERTIFICATION OF APPROVAL

I certify that I have read *QUADRANGULATIONS AND NONORIENTABLE GENERA: A THESIS ON THEORY AND RESULTS IN TOPOLOGICAL GRAPH THEORY* by Warren Singh and that in my opinion this work meets the criteria for approving a thesis submitted in partial fulfillment of the requirements for the degree: Master of Science in Computer Science at San Francisco State University.

Timothy Sun
Assistant Professor of Computer Science

Aakash Gautam
Assistant Professor of Computer Science

QUADRANGULATIONS AND NONORIENTABLE GENERA: A THESIS ON THEORY AND RESULTS IN TOPOLOGICAL GRAPH THEORY

Warren Singh
San Francisco State University
2023

Certain results in graph theory remain incomplete or unverified. By computationally defining and manipulating certain types of graphs, then applying well-defined operations and processes in a procedural, computer-mediated way, it is shown herein that certain theoretical results may be empirically supported. Strengthening of a previous result on minimal quadrangulations of surfaces and establishing the nonorientable genera of two families of graphs are the two major results of this work.

I certify that the Abstract is a correct representation of the content of this thesis.

_____

Chair, Thesis Committee Date

ACKNOWLEDGMENTS

With thanks to TS, for unfailing kindness, gracious and accomodating support, mathematical (and other) legwork, and camaraderie in the research and exploration process.

Utmost gratitude for E, my loving partner, without whom none of this would have been possible.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Chapter 1

Introduction

Graph theory may be described as the branch of mathematics concerned with networks of points connected by lines. The history of the study of graphs is usually traced back to 1735, when the mathematician Leonhard Euler described the problem now known as the **Seven Bridges of Königsberg**.

Since that time, the field has expanded to include many aspects of mathematics which were not conceived of when Euler mused on the nature of his walks in current-day Kaliningrad. These newer additions to the field include considerations of graphs on surfaces other than the plane.

This work provides an overview of work on various problems done with respect to face-simple minimal quadrangulations, and the nonorientable genus of nearly bipartite graphs on particular kinds of surfaces. A primer is first provided in the Preliminaries section on terminology and theorems, as well as previous results. Then, a description of the computational implementation of a number of the functions nec-

98 essary, as well as the tests used for verification follows. A discussion of the results,
99 then concluding remarks and a bibliography follow.

Chapter 2

Preliminaries

2.1 General Fundamentals

There are many particular pieces of terminology used as foundational pieces for this work; the following primer does not cover all possible terms in graph theory, merely aiming to explain those strictly necessary for the work discussed.

A graph may be understood as a collection of points, with lines serving as connections between points; to be specific, an *edge* connects two points (interchangeably, a line or edge may also be called a *link*). The points themselves can also referred to as *nodes* or *vertices*. When there exists an edge between two nodes, we say the two nodes are *connected*, or *adjacent*.

A *simple graph* (usually denoted as a *graph*) is one in which no node is connected to any other node by more than one edge (no "multi-edges"), and also where no node is connected to itself (there are no "self-edges" or "loops").

Edges may sometimes be described with directionality, but are not in the work presented: edges here are undirected for basically all intents and purposes.

Two distinct edges are called *independent* if they do not share any endpoints in common.

A *connected* graph is one in which it is possible to trace a path via edges and nodes from one to the other of any distinct pair of nodes.

A *complete* graph is a graph where each distinct pair of nodes is connected by an edge. A complete graph, therefore, contains all possible edges. One can say that a node in a complete graph is *saturated*, where saturated means that a node is incident with as many edges as is allowed in that context (for instance, since a simple graph does not allow multiedges, a node in a complete graph is saturated since it is adjacent to every other node in the graph). We can then define *unsaturated* immediately as a quality that a node has if it is **not** incident with as many allowable edges as possible.

A *nearly complete* graph is a graph which may be derived from a complete graph by deleting a set of independent edges, up to a *perfect matching* such that no more independent edges may be deleted from the complete graph.

A *bipartite* graph is one in which the set of all nodes may be partitioned into two non-overlapping sets X, Y such that no two nodes in X are connected by an edge, and no two nodes in Y are connected by an edge.

Thus, a complete bipartite graph is one in which each node in one set (without

loss of generality, the "left" set of nodes) is connected to every node in the other set (the "right" set of nodes).

The following graph operations are defined below:

- *Graph complement:* also called an edge complement, this is a graph G' derived from the parent graph G which has the same node set but whose edges consist of all the edges *not* present in G (often denoted as an overline, such as $\overline{H}$, the edge complement of some graph H)

- *(Graph) join:* the result G_3 of $G_1 + G_2$, where G_3 consists of all the edges which connect the nodes of G_1 and G_2, along with the original G_1, G_2 graphs.

- *Disjoint union:* the union of two graphs where identical elements from each graph are nonetheless treated as distinct (distinguished by labels as to their origin) (denoted below using $\cup$)

In mathematics, a *surface* can be thought of in the same intuitive way as the colloquial usage of the term, and is sometimes described as a generalization of the plane.

Of course, surfaces have more formal ways of definition. Generally, surfaces are two-dimensional spaces, geometrical space which resembles a deformed plane. In topology, a surface is a *topological space* where every point has an *open neighborhood* which is *homeomorphic* to an open subset of E^2, the Euclidean plane (most often introduced as part of the Cartesian plane in formal schooling).

Surfaces may be either what is known as *orientable*, or *nonorientable*. Orientable surfaces may be described in a number of ways. One intuitive analogy is to consider a surface as a collection of points, then to imagine a small human walking on the surface without stepping over any edges in order to get "around" the surface. An orientable surface is one in which there is no way for this person to end up upside down while having their feet touching the same point as before (consider the outermost shell of a sphere or a donut). This is in contrast to nonorientable surfaces, which enable this person to be upside down and right side up while "standing" on the same point as before. Some more common examples of nonorientable surfaces are the Möbius strip and the Klein bottle.

Surfaces are important in this context due to the notion of a *graph embedding* (sometimes also referred to as an *imbedding*). On an intuitive level, a graph embedding is the "drawing" of a graph onto a surface such that edges intersect at endpoints, and nodes don't intersect with any other component, ever.

More formally, an embedding of a graph G in a surface S is a continuous, one-to-one function from a topological representation of the graph into the surface [4] (3.1.4).

This definition immediately establishes some things:

- The points of the graph G are associated with nodes of the embedding
- the edges of G are associated with arcs such that:
  - the endpoints of an arc which is associated with an edge e are the points

associated with the nodes of e (i.e., the nodes of e)

- there exists no arc on the embedding which includes points associated with other nodes (nodes which do not form the endpoints of the associated edge)
- on the embedding, no two arcs intersect (since this would mean the associated edges intersect, which they do not on the graphs we are concerned with)

A *face* of a graph may be thought of as a region taking the form of a $k-$sided polygon. More formally, the face of a graph is any region included in the complement of its image that is homeomorphic to an open disk [4].

We can say that a face is circumscribed by edges, and call a face *quadrilateral* if it is circumscribed by exactly four edges. In general, the *length* of a face is the number of edges (not necessarily *distinct* edges, a relevant point later when discussing *face simplicity*) which defines its boundary. The notion of a face is also useful when considering one of the more well known types of embeddings, a *cellular* embedding, in which each face is informally analogous in all the important ways (formally, *homeomorphic*) to an open disk (i.e., a filled in circular region without a defined edge boundary). Embeddings discussed in this work are cellular, but are referred to without this qualifier in general.

A graph embedding is called *quadrangular* if all of its faces are quadrilateral.

Of a graph embedding, we call it *face simple* if no two faces share more than one

edge, and no edge is used in a face more than once.

An embedding is described with the term *minimal* if it has the smallest number of nodes given the context. For instance, one of the results found in this work deals with minimal quadrangulations of a surface, which is a simple graph which has the smallest number of nodes out of all simple graphs that quadrangulate that surface. (i.e., the graph can embed onto the surface such that all faces of the graph are quadrilateral)

Surfaces may also have what is called *genus*, which may be thought about using the heuristic of how many holes there are in the shape. For example, the sphere has a genus of zero, the torus (or "donut") has a genus of one, two tori glued together at one point have a genus of two, and so on.

To be specific, closed, orientable surface genera are denoted $S_0, S_1, ...$, where S_g denotes the torus with g holes. Nonorientable surface genera are denoted $N_1, N_2, ...$ where N_h denotes the sphere with h *crosscaps*, where a crosscap refers what is more generally known as a Möbius strip.

A *rotation* of a node u is a cyclic permutation of the edges which contain u as an endpoint. In an undirected graph, edges do not have direction, but for the purposes of this work, it is sometimes useful to consider that each edge can be thought of as two directed "half-edges", each pointing the opposite direction to the other. In other words, a rotation of a node is described by an ordered listing of the edges which "emanate" out from it.

With this in mind, a *(general) rotation system* for a graph G is a list of rotations for each node along with some indicator of the "straightness" of the edges in each rotation. In particular, edges may be thought of as "straight" or "twisted": straight as in a strip of paper, or twisted as in the same strip of paper given a "half-twist" (or 180 degree flip of one end), making it somewhat akin to a Möbius strip (albeit one which does not have its ends fastened together). In some of the literature referenced, these are known as type 0 and type 1 edges, repectively.

Taking a step back and considering a graph as a whole, it becomes clear that many different rotation systems are possible for a graph (since one can "shuffle" the cyclic order of edges at any node), though possibly fewer for a graph embedded in a *particular* surface (as it can be thought of as putting topological constraints onto the situation), since we recall that in a valid embedding, no edge crosses any other edge. Specifically, only some rotation systems induce embeddings on a fixed surface, but every rotation system corresponds to an embedding in *some* surface.

It is also of note to observe that different rotation systems can refer to the same embedding, since one can "flip" a node, which reverses the cyclic rotation at that node and switches each of its edges from one type to the other ($0 \longleftrightarrow 1$). In general, a rotation system with multiple type 1 edges could still be orientable, the exceptional condition being if there exists a closed walk that contains an odd number of type 1 edges (intuitively, one can see this by virtue of the fact that Möbius strip is a nonorientable surface).

Explicitly, a *node flip* is an operation on a particular node of a graph. Conceiving of the two "sides" of a node, and marking one of the two orientations as facing *up* or *down*, a node flip changes a node from facing up to facing down, and vice versa. Flipping a node in this manner reverses the cyclic rotation at that node, and changes the type ($0 \longleftrightarrow 1$) of all of its edges, including (and this is important to note for implementation purposes) how all of this nodes' neighbors "understand" those edges (as either type 0 or type 1).

When attempting to find results dealing with orientable or nonorientable objects, it becomes necessary to check the orientability of a graph embedding. We use the following algorithm as the approach for this orientability check.

A breadth-first search is performed on a graph embedding in order to create a spanning tree with only type 0 edges (edges with no half twist). In order to ensure this with the tree as it is under construction, nodes are flipped until the spanning tree contains only type 0 edges. Then, after completion, all edges *not* contained in the spanning tree should be type 0 edges as well. If both of these are true, then the embedding is orientable (regardless of its initial presentation, which may have contained one or more type 1 edges) [4].

In a 1750 letter to Goldbach, Euler described an equation for what is now called the *Euler characteristic* of a polyhedron. The equation is

$$V - E + F = \chi$$

where V is the number of nodes, E is the number of edges, and F is the number of faces. The Euler characteristic (χ) of a sphere, for instance, is 2 [4]. This equation also applies to graphs, and more specifically for the aim of this work, to graph embeddings.

Usefully, the Euler characteristic χ can be substituted for $2 - 2g$ where g is the genus of a (orientable) surface, and thus the genus of a graph embedding can be found or verified given the other information about a graph. (for a nonorientable surface, $\chi = 2 - h$ where h is the genus of the nonorientable surface)

The *dual* of a graph embedding is one graph embedding derived from another. Given some graph embedding G, the dual of G is another graph embedding D, which has a node for each face of G, and edges between nodes where the faces in G are separated by an edge. There are some further details for self-loops, but they are not used in this work, since this work concerns simple graphs.

The dual of a graph embedding becomes particularly relevant when considering the quality of *face simplicity*, defined again thusly: a graph embedding is *face simple* if its dual is simple. Face simplicity can also be thought of, as before, in terms of the original embedding: no two distinct faces share more than one edge.

Andre Bouchet[3] defined the notion of a *diamond sum*, which is a particular way of combining one graph embedding with another, and which is useful for building complete graph embeddings. This procedure forms the foundation of the approach for building embeddings of complete bipartite graphs out of smaller embeddings. Here, the diamond sum construction is used to iteratively build up larger and larger graph embeddings (larger in the sense of involving ever-greater numbers of nodes and edges) while preserving properties relevant to the inquiry (such as face-simplicity, or the precise nature of an embedding, i.e. quadrilateral) [13].

By way of example, suppose that there are two embeddings which we wish to diamond sum together. Then, we choose one node to remove from each embedding, call them u, v. After, we carefully list (we note than the term *list* is used to describe a collection of objects in which order both matters and is preserved) out all neighbors of each in two separate lists (i.e. one for u, one for v). Assuming (without loss of generality) that each list from first item to last item is associated with a clockwise direction of rotation around the "origin" node (the node from each embedding which is being removed) we pair the lists off, such that each neighbor of u is "merged" with one (and only one) neighbor of v. It is also important to make explicit here that this means the diamond sum operation is valid only on nodes which have the same degree (i.e., the same number of neighbors)

It is also important to note that by flipping nodes before performing the diamond sum on two graphs, we can ensure that none of the nodes involved in the diamond

sum have any type 1 edges, since the removal of these edges from the graph would affect the result adversely (the orientability or non-orientability quality may change in an undesired and undefined way).

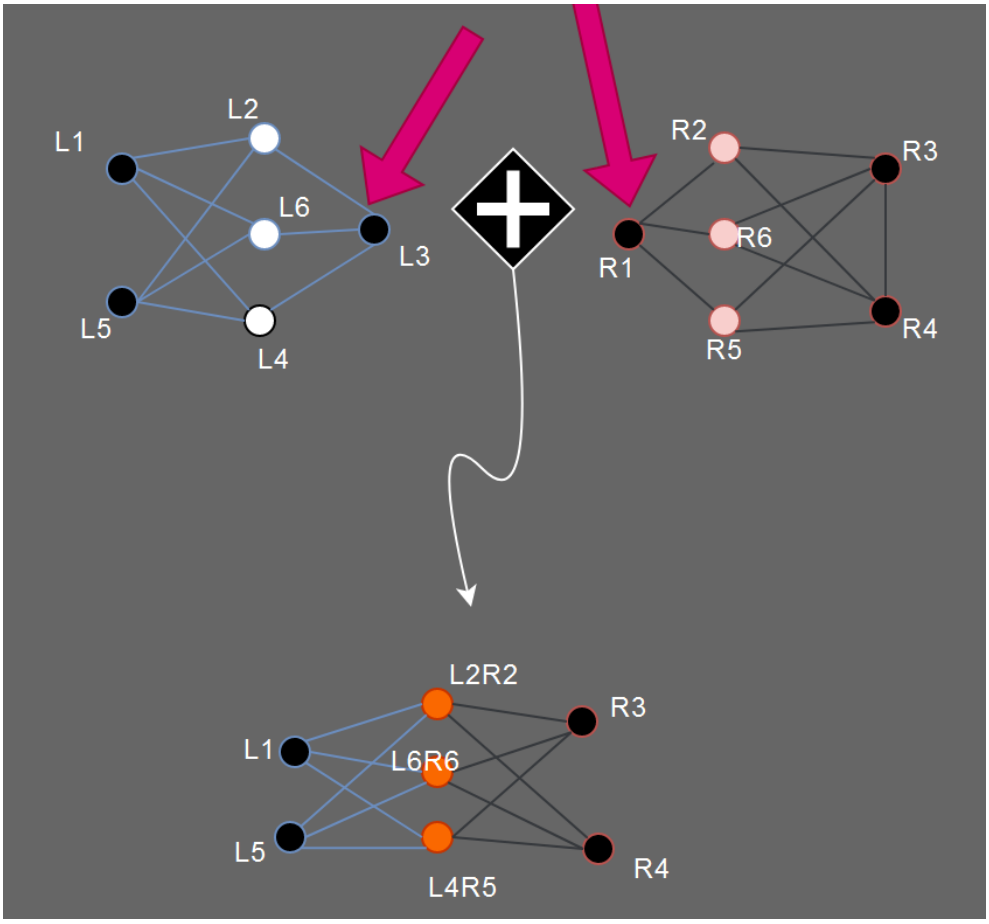

Figure 2.1: An example of a diamond sum of two graphs, focused on the nodes L3 and R1

On a pictoral level, each of the neighbors of u has merged with one of the

neighbors of v, and each of the merged nodes now has all of the neighbors of each of their parent nodes. In the implementation, the existence of the "neighbor lists" becomes much more important, and the each merged node does not simply have the two parent lists concatenated - instead, one of the parent lists is reversed, and *then* the two lists are concatenated. This is to preserve the "clockwise" rotation of the neighboring nodes around the resulting merged node.

It is useful to note that the diamond sum is a *connected sum* (a term originating from topology referring to a particular operation on surfaces). For us, this means that the result of a diamond sum of two graph embeddings has a genus equal to the sum of the two "input" embeddings (though the term connected sum referred originally to surfaces, it applies to graph embeddings for reasons discussed in the section on graph embeddings).

One can see this in the following way: the diamond sum excises a node which has no type 1 edges (edges which contain a half twist) and then merges all of the neighbors with the other input (details as previously discussed). Genus is determined by the number of "handles" or "crosscaps" affixed to the surfaces, and because of the stipulation of "all type 0 edges at the excised node", none of these handles or crosscaps are affected. Therefore it follows that the result has a genus equal to the sum of the genera of the inputs.

It is sometimes the case that we will be interested in which nodes in a resulting graph originate from which parent graph (or, by extension how an end result graph

might be built up iteratively over multiple successive diamond sum operations): thus we might wish to carefully label (and relabel after diamond sum operations) which nodes come from which parent graph. This idea is more fully discussed below in the implementation section.

Current graphs are a particular kind of weighted directed graph (the edges have both weights and directions), and have the property that the sum of the weights entering a node (going "into" the node) is equal to the sum of the weights leaving the node (the idea is the same as in idealized current graphs from the study of circuits, hence the name). Some of the results are achieved with current graphs, further information on which (in particular, the constructions of current graphs) may be found in sections 4.4 and 6.1.3 of Gross and Tucker's monograph [4].

Some of the verifications/constructions performed rely on having access to the exact rotations at all nodes (recall this means a specific, persistent clockwise order of all of a nodes neighbors). Ringel [11, 12], in his first proofs of orientable and nonorientable genus formulas for complete bipartite graphs actually provides a specific way to generate a complete rotation system for bipartite graphs with a specified number of nodes, so for the relevant portions Ringel's rotations systems (generated via the method he described in the previously cited papers) are used.

2.2 Previous Work

2.2.1 Quadrangulations (minimal and otherwise)

Hartsfield and Ringel [6, 5] first formulated the general problem of finding a minimal quadrangulation for a closed surface a little over thirty years ago at the time of this writing. For bipartite graphs, Ringel showed that $K_{m,n}$ has an orientable quadrangular embedding for $m \equiv 2 \pmod 4$ when $n \geq 2$.

We recall from previous sections that a minimal quadrangulation of a surface is the embedding of a graph into a surface such that the embedding has only quadrilateral faces and that the graph has the smallest possible number of nodes.

The problem of finding a minimal quadrangulation can be rephrased through examining certain numbers of edges and nodes. This is insightfully referred to by some as a "graph-centric" approach (as opposed to a "surface-centric" approach), since it deals with nodes and edges instead of focusing on characteristics of a surface. An embedding is called an *(n,t)-quadrangulation* if it is a quadrangular embedding of a simple graph with n nodes and with $\binom{n}{2} - t$ edges.

We consider again the formula for the Euler characteristic of a polyhedron:

$$V - E + F = \chi$$

Supposing that the graph embedding G_e is quadrangular, and since each edge

has two sides, we can say that $2E = 4F$. Substituting for F, we have

$$V - E + \frac{1}{2}E = \chi$$

which gives

$$V - \frac{1}{2}E = \chi$$

and solving for E,

$$E = 2V - 2\chi$$

and since the number of edges in a simple graph in general is $\binom{v}{2}$ (and here for $(n,t)-$quadrangulations it is $\binom{v}{2} - t$), substituting for E and solving for t, we now have

$$t = \binom{v}{2} - 2V + 2\chi$$

expanding the binomial term and walking through a bit of algebraic manipulation, we get after a bit

$$t = \frac{v(v-5)}{2} + 2\chi$$

We recall that $\chi = 2 - 2g$ for orientable situations (where g is the orientable genus) and $\chi = 2 - h$ for nonorientables ones (where h is the nonorientable genus). Supposing that we take the modulo of this equation in each of the respective bases for the orientable and nonorientable cases (4 and 2, respectively) in order to eliminate

extraneous terms (integer and genus), we get

$$t \equiv \frac{v(v-5)}{2} \pmod 4$$

and

$$t \equiv \frac{v(v-5)}{2} \pmod 2$$

for the orientable and nonorientable cases, respectively.

Thus we now have a necessary condition on t for G_e to be a quadrangular embedding. This derivation owes to Liu, Ellingham, and Ye (LEY) [8] as a specific reference, though it is well established generally in the field of topological graph theory [4].

Supposing that we have a minimal quadrangulation, we also get the condition that $0 \leq t \leq n-4$, since if t were allowed to be $n-3$ or greater, there could arise the situation that enough of the missing edges were on the same node to make it of degree 2. Then, this node could simply be deleted, resulting in a smaller quadrangulation, a contradiction.

LEY [8] also establishes the existence of $(n,t)-$quadrangulations of this form using a technique which this work does not follow (described in the "Preliminaries" section and further discussed in the "Results" section, this work relies on Bouchet's diamond sum technique, as opposed to Hartfield's "diagonal technique" as LEY uses).

2.2.2 Nonorientable Genus and (less than complete) Bipartite Graphs

Our work in this area primarily builds on the work of Mohar et al. [10], in particular their work on the nonorientable genus of nearly complete bipartite graphs, though an earlier work on orientable genera of the same yields insight into details and methods which overlap [2].

Since this work deals with nearly complete bipartite graphs, there exist slightly different realities for the number of nodes, where

$$V = m + n$$

where m, n are the two (nonoverlapping) sets of nodes belonging to (without loss of generality) the *white* (or left) and *black* (or right) nodes. The notation $G(m, n, k)$ is used to refer to a graph obtained from a complete bipartite graph $K_{m,n}$ and which has k independent edges (i.e., no two distinct edges share any endpoints) deleted.

Similarly to before, they start with the equation for the Euler characteristic of a polyhedron (referred to by them as Euler's formula) in the nonorientable case:

$$V - E + F = 2 - h$$

with a bipartite graph, the minimum length of a face is 4, and thus $4F \leq 2E$, meaning

$$V - E + \frac{1}{2}E = 2 - h$$

and substituting for V,

$$(m+n-k)-\frac{1}{2}E=2-h$$

since K, the original graph was complete, the original number of edges is mn, of which we have removed k, thus

$$(m+n-k)-\frac{1}{2}mn+\frac{k}{2}=2-h$$

and with some algebraic manipulation, keeping in mind genera are defined as positive integers,

$$h \geq \left\lceil \frac{(m-2)(n-2)-k}{2} \right\rceil \tag{2.1}$$

with this as a lower bound since a higher genus surface can always "house" a lower genus embedding.

With parity considerations, and a similar derivation as the end of the derivation for t in the previous section on quadrangulations, it becomes clear that $mn \equiv k \pmod 2$ for nonorientable genera, and (mod 4) for the orientable case (though that is not addressed in detail in our work).

We can also note that finding the nonorientable genus for $G(m,n,k)$ using 2.1 gives the answer for $G(m,n,k+1)$ as well just due to how 2.1 computes.

Their work on the nonorientable genus of the nearly complete bipartite graphs, primarily showing that 2.1 holds in general, leaves only $G(k,k,k)$ and $G(2k+1,2k,2k)$ as exceptions to the general result of 2.1. This gap is exactly what our

work on this subject fills in.

They reduce the problem to finding graphs which have quadrangular embeddings, though as a general note the diamond sum techniques applied successfully to the orientable cases don't work for the extra cases needed for the nonorientable ones. The approach actually taken for these results is detailed below in the Results section.

Chapter 3

Implementation

The implementation of the codebase, encompassing necessary operations as well as various quality of life functions was done in Java, and the full repository may be accessed by emailing the author of this report. The codebase was managed with Git and is currently stored on GitHub.

A primer on the important parts of the implementation is provided below; such necessary but prosaic things as accessors and mutators go unelaborated on but may be found within the code itself.

For certain graphs (discussed below in the Results section), we use generated rotation systems from Ringel [12]. In these cases built for $K_{n+1,2n}$ for both even and odd n, we hardcoded the rotation systems according to the following: For even n:

- Label the $n+1$ left nodes as $a, b, c, d, ...$ and the right nodes as $1, 2, ..., 2n$
- Each rotation at the right nodes will alternate between $1, 2, ..., 2n$ and $2n, ..., 2, 1$, beginning with the ascending sequence of the two

- Nodes $c, d, ...$ on the left will also alternate $2n, ..., 2, 1$ and $1, 2, ..., 2n$, beginning with the descending sequence of the two

- Node a is given the rotation:

$$n+1 \quad n+2 \quad n \quad n-1 \quad n+3 \quad n+4 \quad n-2 \quad n-3 \quad ... \quad 2n-1 \quad 2n \quad 2 \quad 1$$

- Node b is given the rotation:

$$1 \quad 2n \quad 2 \quad 3 \quad 2n-1 \quad 2n-2 \quad 4 \quad 5 \quad ... \quad n+3 \quad n+2 \quad n \quad n+1$$

- type 1 edges are $(a, 1), (a, 2), ..., (a, n), (b, 1), (b, n+2), (b, n+3), ..., (b, 2n)$

To generate type 1 edges in order to achieve the desired quadrilateral and nonorientable qualities, the discovery of the patterns used as described simply proceeded by human directed exploration. Then, as testing examples produced desired results, this procedure for the "half-twist" edges was used, and the base cases also checked for the desired qualities.

The method for generation is reiterated briefly and the use to which the generated Ringel embeddings are put discussed further in the Results section, in the proof for **Theorem 4.1**.

We also made use of a separate generating method for two other families of graphs in the following way:

For each rotation systems, the vertices are labelled as elements of the group $\mathbb{Z}_{2n}$ for a given n. Two "generating rows" a, b are described. Then, the rotation of any even-labelled node i is found by adding i to every element in row a, and the rotation of any odd-labelled node j is found by adding j to every element in row b.

For the family of graphs described by $K_{n,n}$ minus n independent edges, where $n \equiv 1 \pmod 4$, and $n \geq 9$, row a is

-3 1 **5** **-7** **9** **-11** **...** **n-2** 3 -1 **-5** **7** **-9** **11** **...** **-(n-2)**

and row b is

-1 3 **n-2** **...** **11** **9** **7** **5** 1 -3 **-(n-2)** **...** **-11** **-9** **-7** **-5**

where for each row the **bolded** terms are sequences of consecutive odd integers. The sign is either consistent, or alternating. The type 1 edges here take the form $(i, i+1)$ for all $i \in \mathbb{Z}_{2n}$.

For the family of graphs described by $K_{n,n}$ minus n independent edges, where $n \equiv 3 \pmod 4$, and $n \geq 7$, row a is

-5 1 **7** **-9** **11** **-13** **...** **-(n-2)** 5 -1 3 -3 **-7** **9** **-11** **13** **...** **n-2**

and row b is

$-1 \quad 5 \quad$ **n-2 ... 13** $\quad$ **11** $\quad$ **9** $\quad$ **7** $\quad 1 \quad 3 \quad -3 \quad -5 \quad$ **-(n-2) ... -13** $\quad$ **-11** $\quad$ **-9** $\quad$ **-7**

where for each row the **bolded** terms are sequences of consecutive odd integers. The sign is either consistent, or alternating. The type 1 edges here take the form $(i, i+1)$ for all even $i \in \mathbb{Z}_{2n}$ and $(j, j+3)$ for all j.

These graphs are part of a type of graphs known as current graphs, as discussed in the General Fundamentals section above.

Both types of generated rotations discussed above (Ringel and current) were produced by hand since there were relatively few base cases that were then used, though a more involved effort to automate the generation of these systems could have been undertaken if needed.

One important file inside of the codebase concerns base cases. Because the results involve graphs of arbitrary number of nodes, and the proofs are approached inductively, a group of graphs (many of consecutive number of nodes) needed to be constructed with the appropriate qualities. This base case file contains a number of graph with necessary qualities (orientable, nonorientable, of varying number of nodes, etc.) which have been verified or constructed to function as building blocks for the purposes of the desired results.

Beginning with base cases which have the needed qualities (which come from papers discussed in the Previous Results section above – some generated by hand, but

all with the verified and needed qualities for iterative constructions), then diamond summing them in a specific manner (which preserves the desired qualities) in an iterative fashion allows for the building up of the number of nodes of the end result, which is guaranteed to still have the desired qualities (though additional code was written and used to double check those qualities at the end). The end results are a "tranche" of objects which were verified to have the desired qualities; these objects are "sequential" in that the integer values of their number of nodes is sequential (or similarly) such that by induction, since this tranche fits the desired results, the rest of the objects with $n, ...$ number of nodes (as in, the rest of the graphs in this "family") are also proved by induction to hold the properties desired.

The class archetype Embedding was used for holding most of the actual mechanics and functionality, and most of the foundational implementation in terms of data structures and operations takes place as a result of the Embedding's methods and data.

One unexpected complication was the difference between shallow copies and deep copies, as well as the separate but related issue of how Java passes by reference instead of by value. All of this means that in order to perform various checks and operations it often becomes necessary (in order to avoid undesired program behaviour and outcomes) to explicitly make a deep copy of whatever object/structure is in question. Taking the easier approach (by relying on references) results in crucial data structures being mutated and downstream processes not finishing correctly as

a result.

This was particularly important when verifying the orientability (or non-orientability) of any particular graphs, since this involved constructing a spanning tree of the graph using Breadth-First Search (BFS), then flipping the orientation of various nodes in sequence. Early results from testing produced errors until this oversight was rectified, and the BFS construction was preceded by making a deep copy of the graph being examined.

Orientability testing could then proceed correctly once deep copy implementation was taken care of. This was implemented naively in Java in accordance with the "orientability testing algorithm" described above in the General Fundamentals section.

Node flips were implemented by defining functional operations on internal data structures, meaning holding sets of type 1 edges (any edge not included is assumed to be type 0) to which edges were added or removed during the course of the node flipping operation. No information was explicitly stored about a particular node being "face up" or "face down". While this might be of interest in providing further detail about a particular embedding, it was not seen as important, and the differentiation between different embeddings were implicitly present due to explicit information present and recorded (i.e., which edges were type 1).

The graph embeddings were represented by the Embedding Class. In particular, each instantiation of the Embedding Class represents a rotation system, using two

key data structures to represent the important parts which define a rotation system. The first is a HashMap data structure in Java (similar to an abstract "map" in general) which holds key-value pairs. In this implementation, the HashMaps use a String datatype as a node name/label as a key, and a Java ArrayList datatype holding Strings as the value (this is the list of a node's neighbors, or its *adjacency list*). The edges are thus implicitly represented by each node/neighbor pair and can be constructed explicitly later by using the information present. We recall also that the ArrayList has a persistent ordering of elements, which is crucial since rotation systems are defined by the cyclic permutation of neighbors around each node. The ArrayList was chosen due to its mutability (in particular, its built-in ability to change size/capacity, which was important for some of the automated diamond summing processes needed), but otherwise behaves usefully like an array in that it has indexed values and an accessible length property.

The second most important data structure is a HashSet in Java, which is used to keep track of the type 1 edges, if any, in a particular instance of the Embedding class (we recall that a rotation system is defined by both its node/edges collection, and by its type 1 edges). The HashSet is used since it allows for set functionality inside of this particular data structure: to be explicit, that only one of each edge may be stored within. This also makes necessary the writing of an equivalence definition for edges. One might easily imagine that an edge going from node A to node B (designated "AB") could be falsely interpreted as a different edge than one

going from B to A (designated "BA"), even though they are the same edge. This is expecially important when it comes to type 1 edges, since they are used in their totality (as opposed to half edges with respect to faces). So defining equality (more precisely, an equality determining function for comparing type 1 edges) for edges to be "order agnostic", i.e. $AB = BA$ is a crucial middle step.

Once this is done, the HashSet datastructure which is used to store references to (keep track of) type 1 edges has some built in functionality in Java which make it useful (such as a fast way of checking if the set contains an object already).

Perhaps the most crucial bit of code for the entire project was the implementation of the diamond sum operation. Implementing it successfully took perhaps the longest time of any function written in the code; this is because of its foundational importance, the complexity of the actual operations, as well as its rigidity (although the operation is well defined, explicitly coding the process results in something which, if any one line or variable reference is incorrect, results in errors being thrown; the process cannot be considered robust or fault tolerant as a result).

The diamond sum function is a member function of the Embedding class, meaning that an instance of an Embedding (which may be thought of as a particular graph) calls this function on another Embedding, and returns the resulting "combined" (diamond summed) embedding. The arguments for this function are references to the other graph, each of the two string labels for the two nodes which are to be excised from each (one from the left graph, one from the right graph), and

two string labels for neighbor nodes, one from the left node to be excised, one from the right node to be excised.

Because it is of potential interest to preserve the information about where resulting nodes come from, all nodes from the "left" calling graph have their node's string label names prefixed with a "0", and similarly for the "right" other graph with "1". Note that this also applies not just for the keys in the HashMap representation of the graph, but also for all of the neighbor list (ArrayList values) as well.

After the removal of the excised nodes (along with the removal of all mention of these nodes from any inclusion in neighboring node's lists of neighbors), the merging of each of these nodes neighbors in a pairwise fashion has to begin. Beginning with the two nodes whose labels were provided in the function call, new nodes are created. The new node labels come from concatenating the two names of the parent nodes, and the new node neighbor lists come from filling a newly created ArrayList with the combined two lists of the parent nodes's neighbors. We note again that in order to preserve a "clockwise" orientation of neighbors, the "right" graph's nodes for merging have their neighbor lists reversed before concatenation with the neighbor list from the left graph's nodes, meaning the resulting merged node has its list of neighbors in clockwise order around it by increasing index in the neighbor list.

As the process runs, it is important to relabel all relevant labels across all the graph, meaning that in addition to the merged nodes' new names, each of the neighboring nodes to the new merged nodes have to have the new updated label for

the merged node in their adjacency list of neighbors.

Type 0 edges (no half twist) are implicitly included here through the existence of the central node with each of the entries in its adjacency list. As the full description of a rotation system can also include type 1 edges (an edge with a "half twist" simiar to a möbius strip with its ends unfastened), each instance of the Embedding class also contains a separate list for holding information on its type 1 edges. As such, the type 1 edge list would also have to have its entries relabelled in the case of merged or deleted nodes. This problem is circumvented by the "cleaning" of nodes prior to the diamond sum operation taking place; the cleaning process is discussed more below.

In the process of building up a graph embedding of arbitrary size, it sometimes becomes desirable or necessary to check the genus of a particular embedding. In order to do so, we use Euler's previously mentioned equation (modified slightly):

$$V + E - F = 2 - 2g$$

where g is the (orientable) genus of the graph/embedding/surface in which the graph is embedded. To be specific, we solve for g, rearranging the terms in a relatively straightforward algebraic manner, and then use available values for V (the number of nodes in the graph), E (the number of edges in the graph), and F (the number of faces in the graph) to find g. This is for orientable surfaces, noting that for nonorientable surfaces, $2 - 2g$ (the Euler characteristic for orientable surfaces) is

instead $2 - h$ (where h is the nonorientable genus).

It is at this point that one might ask how one obtains F. It seems clear from the way that the graph embeddings are represented using the data structures as described that there are ways to find V and E relatively easily (either by linear enumeration over the data structures, or by persistent storage of the information and accessing/mutating it as needed). But there doesn't appear, *prima facie*, to be a straightforward or simple way to easily obtain the number of faces in a graph.

Instead, a computational-first approach is taken. For any given graph embedding, the first neighbor of the first node (in an arbitrary ordering) starts the process. Then, that first neighbor is considered. In that first neighbor's list of neighbors, the origin node is located, then the next neighbor by index is considered. This continues in a similar fashion until the original origin node is reached again. Then, the chain of nodes examined in this way (along with the edges which connect them) is a face. We are guaranteed, because of the way that the embeddings are constructed, that each of the edges that we traverse to get to the nodes in this face tracing process can be thought of as two directional "half edges". When dealing with face-simple graphs, we note that undertaking a "face-tracing" procedure, no directional half-edge is repeated – any single directional half-edge occurs only once throughout any face tracing process. Thus detection of any repeated directional half edge in this face tracing process means either a duplicated face (a face that has already been traced and recorded), or indicates the current face being traced is complete.

Then, the next neighbor of the first node in that arbitrary ordering is examined, and so on through all of the neighbors of all of the nodes of the graph embedding. Storage of each of the faces traversed so far allows for early cutoff when examining duplicate faces.

At the end of this process, all the faces of a graph embedding have been examined, and the exact number of faces in a graph embedding can be accessed and used for analytical purposes.

More formally, this process is generally known as Heffter-Edmonds face tracing, and is useful for computing the genus of a rotation system of a simple graph.

Because type 1 edges have a half twist in them, face tracing along edges leads to a topological curiosity. Following a type 1 edge from one node to the other leaves the "spotlight" on the "underside" of the destination node. And technically speaking, this face being traced actually includes this particular underside of the node. So how does one keep track of faces in this instance?

While it could be the case that there is some way of keeping track of whether the current "trace" is on the "upside" or the "downside", one slightly easier solution (more proceduraly straightforward) is to generate a *double cover*, which is derived from the Embedding's HashMap. In a double cover, each node is turned into two nodes, which correspond to the upside and the downside. Type 0 edges map inside of each of these two groupings, while type 1 edges map between these two groupings. Then, face tracing can proceed accordingly on the double cover data structure (which

is similarly represented using a Java Hashmap). It may be noted that an orientable embedding (i.e. one which may have its nodes flipped in such a way that it contains no type 1 edges), when having its faces traced on its double cover, the tracing will never go from one "side" to the other, since none of the upside nodes will have edges connecting to downside nodes (given the absence of type 1 edges). Thus tracing on the double cover is really only necessary to use as a process when considering nonorientable embeddings.

Regardless, this functionality which output the double cover of a given embedding was built in to the Embedding class, since this allowed for the face tracing code (which was also relatively complex to implement, test, and debug) to remain unchanged.

When diamond summing two graphs together, it is necessary that none of the edges connected to the excised node have a half twist (i.e., are type 1 edges), and thus some functionality was built in to address this. One function was written in order to check whether an excised node is involved in any type 1 edges. Then, a separate function flips the orientation of any neighboring nodes which form a type 1 edge with the excised node, until there are no more type 1 edges emanate from the excised node. Failing to do this piece of due diligence can cause errors downstream in an iterative graph embedding construction process, since type 1 edges may be erased from a result, leading possibly to undesired results (e.g., an end result is orientable when it should be nonorientable).

Because of the arbitrary nature of the number of nodes of the constructions (as in, the number of nodes chosen for an embedding in the end result could be as high as dictated), it became very useful to develop scripts which could automatically iteratively build up more complex graphs to use in constructing the desired end result.

While the original diamond sum function required as input the explicit names (labels) of the nodes involved, this proves unworkable when the goal of a desired script is to run automatically. Thus the original diamond sum function was designated as a "raw" diamond sum, and some functionality was built as an interface on top of it, with a specification of the degree of node which was to be excised as an input. Then, this interfacing function programmatically finds two nodes of the appropriate degree from the left and the right, temporarily stores the appropriate explicit names, and uses that information to call the raw diamond sum with all the requisite information.

This proved very useful when construction graph embeddings of arbitrarily large number of nodes.

One piece of testing code that was built was some code which checked for whether or not a graph was bipartite, by looping through all the current nodes and checking the number of outgoing edges of each. Only some distinct values (two if the graph is complete, or four if it is nearly complete, since missing edges are independent) should be present in the graph if it is actually bipartite – if there are more than

appropriate, then clearly the embedding being tested was not bipartite. As a final check, the script also checked the genus of the (ostensibly) bipartite graph using pre-built facetracing functionality and the relevant equation for the Euler characteristic, and output all of this information for verification.

As a side note, one could consider checking for completeness via the number of nodes and edges in a bipartite graph, though there were few enough instances where this came up such that manual checking was sufficient. Implementation would be relatively straightforward in the case that this was needed on a more frequent basis.

One quality of life piece of code made use of Java's built in file input/output library in order to take in or extract out embeddings in an external library compatible format. This involved some scripting and manipulation of a text file, and made interfacing with external validation programs easier. To be specific, the approach that this codebase took to calculating genus and so on was based on a "first-principles", mechanistic face tracing, while some validation testing was performed with a different library which used a cyclic permutation decomposition strategy. This cross validation with respect to two distinct approaches allowed for confidence moving forward with the intermediate results as the work progressed.

Chapter 4

Results

The work in this chapter is drawn directly from the published written work of, and is essentially mirrored here with the knowledge and consent of the authors of [1, 13].

4.1 Face-Simple Minimal Quadrangulations

Mentioned before, Liu et al.'s previous results are more formally reproduced here:

Theorem 4.1. *Given integers n,t such that* $n \geq 5, 0 \leq t \leq n-4$, *and* $t \equiv \frac{1}{2}(n(n-5))$*(mod 4), there exists an orientable (n,t) quadrangulation. In general, there exists an orientable (4,2) quadrangulation.*

Theorem 4.2. *Given integers n,t such that* $n \geq 4, n \neq 5, 0 \leq t \leq n-4$, *and* $t \equiv \frac{1}{2}(n(n-5))$*(mod 2), there exists a nonorientable (n,t) quadrangulation. In general, there exists an nonorientable (6,3) quadrangulation.*

They note at the end of the same work that they leave as an open question the

addition of *face-simplicity* to the qualities of **Theorem 4.2** (n.b., for all but a few smaller graph embeddings).

The previously described operation of the diamond sum comes back into play when answering this open question.

As a nice intuitive explanation for our proof approach for these graph theoretic results, we begin with a number of base cases of "smaller" graphs whose properties are established and which may be verified by hand (if desired) and then inductively construct (using the diamond sum operation) from those base cases until we have a "tranche" of graph embeddings of consecutive numbers of nodes which suffice to establish the result. We remind the general reader that once we have this tranche of results, any further results can be obtained from them by diamond summing one of the base cases with one of these results from the tranche (as appropriate) some number of times until any arbitrary further desired result is obtained. The fact that the diamond sum preserves the quadrilateral property of its inputs is also a crucial fact to note.

In particular, one useful lemma from [8] (Lemma 3.5), applied specifically for quadrangular embeddings:

Lemma 4.3. *Given ϕ, a face-simple quadrangular embedding derived from a simple graph G of minimum number of nodes $n \geq 3$, let $v \in G$ be a node with independent neighbors. Then suppose ϕ' is a quadrangular embedding of a simple graph, which is face simple except at some node v'. If v, v' have the same degree, then any diamond*

sum of ϕ, ϕ' at v, v' is face-simple as well as quadrangular.

This lemma allows us to use "nearly face-simple" building blocks and guarantee the results of diamond sums to be both face-simple and quadrangular.

4.1.1 Nonorientable Surfaces

The addition of the quality of face-simplicity to the previous result of **Theorem 4.2** from [8] forms the primary novel result of the portion of this work which concerns face-simple minimal quadrangulations of surfaces. The results discussed in this section (and particular figures as noted) can be found essentially unchanged in published format for further clarification [1].

A particular observation ("2.3") from [8] proves useful here: that every orientable quadrangular embedding of minimum number of nodes of 3 or greater is face-simple. This is because for quadrangular embeddings, due to the number of faces, edges, and the number of nodes of the embedding, there are a limited number of ways that two faces can share more than one edge, particularly when considering embeddings of larger number of nodes. For a nonorientable quadrangular embedding composed of 3 or more nodes, this also means that there is only one way in which two faces can share more than one edge.

As mentioned in the Previous Results section, Ringel showed something useful for quadrangular, face-simple embeddings (i.e. for $K_{m,n}$, when $m \equiv 2 \pmod 4$ and $\min(m, n) \geq 3$) [11]. Embeddings which fit these criteria form some of the base

cases used. Other embeddings used are quadrangular, and may be found in the relevant figure 4.1 as "unfolded" polygons with nodes labelled. Each embedding is denotated as a $\phi_{n,t}$ graph embedding homeomorphic to a corresponding simple graph on n nodes and $\binom{n}{2} - t$ edges. If a base case embedding is not face-simple, it is only "face-complex" at one node (designated as node x).

The various superscripts also carry the following information:

- "+" meaning this particular embedding is derived from a nearly identical embedding by dividing one edge of the orginal into two edges mediated by a new "inserted" node

- "$*$" meaning this particular embedding is orientable (other embeddings are nonorientable, since the "refolding" of the polygon shows that some edges in fact contain a half twist (i.e., are type 1 edges))

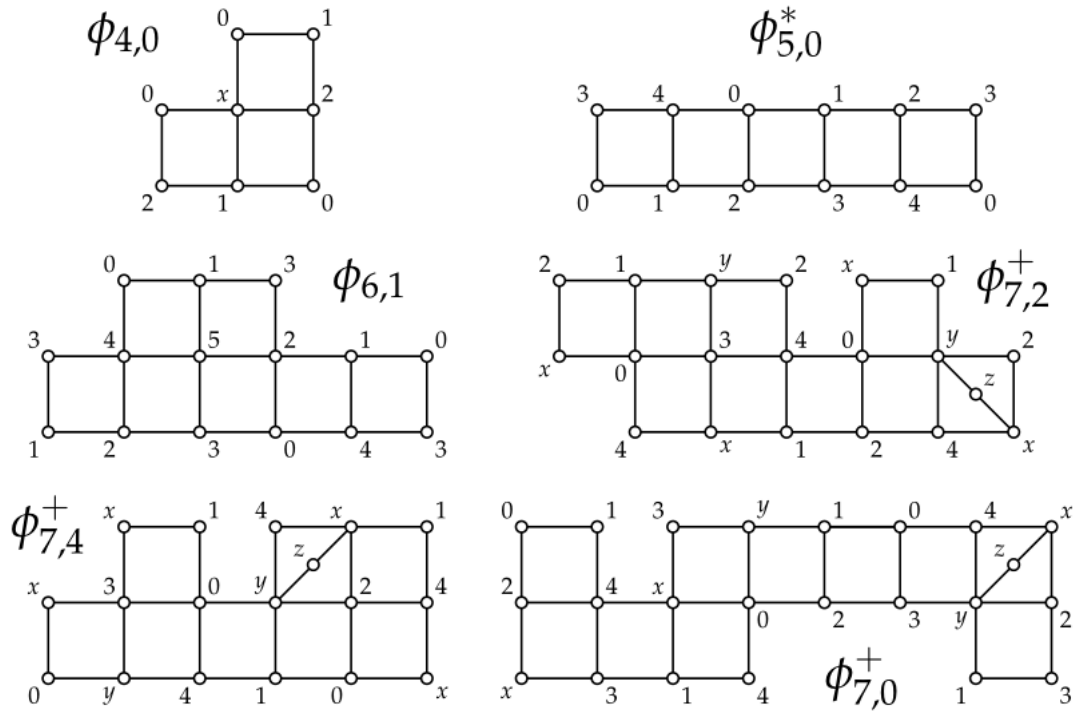

Figure 4.1: unfolded nonorientable quadrangular embedding base cases via [1]

The foundational result in the inductive construction is a generalization of a lemma from [9] (4.2), which outlines a very similar line of argument.

Lemma 4.4. *Given that for $n \geq 4$, some $(n,t)-$quadrangulation ϕ from a simple graph G that is face-simple except at a node $v \in G$, and v is adjacent to every other distinct node, then for all $i \in \{0,2,4\}$, there is some face simple nonorientable $(n+4,t+i)-$quadrangulation. This $(n+4,t+i)-$quadrangulation also has one*

node adjacent to every other node.

Another way to refer to this node adjacent to every other distinct node is as a *hub* node, which is the term we will use in the writing moving forward. Other terms include *universal* node, and *cone* node.

Proof. Take any arbitrary $\phi^+_{7,i}$ from above 4.1. By "breaking it down" through graph complements, joins, and (disjoint) unions, this arbitrary graph can be described as $\overline{K_2} + ((K_1 + H_i) \cup K_1)$, where H_i is a graph made by taking the complete graph on 4 nodes and deleting i edges from it. $K_1 + H_i$ are comprised of numbered nodes (as labelled), where node 0 *is* K_1, x, y are the two nodes of $\overline{K_2}$, and z is the remaining K_1. For H_2, (1,3) and (2,3) are the missing edges, while for H_4 they are (1,2), (1,3), (2,3), and (3,4).

Then let ϕ' be an orientable quadrangular $K_{6,n-1}$ embedding, and $u \in \phi'$ any node of degree 6 (of which, we may note, there are $n-1$ to choose from). Then we can call ϕ_r the result of diamond summing together $\phi^+_{7,i}, \phi$ at the nodes x, u, respectively. Using **Lemma 4.3**, we can see that ϕ_r is face-simple.

Similarly to before, we can describe ϕ_r as $((K_1 + H_i) \cup K_1) + \overline{K_{n-1}}$. Then let ψ as in **Lemma 4.4**, and take the diamond sum of ϕ_r, ψ at z, v (recall that ψ is face simple *except* at v), giving the result ϕ_s.

ϕ_s is a quadrangular embedding, able to be described as $(K_1 + H_i) + (G - v)$ where G is a simple graph.

Counting up, ϕ_s contains $(1+4)+(n-1) = n+4$ nodes and $i+t$ edges removed.

By **Lemma 4.3**, ϕ_s is face-simple, none of the diamond sums have removed any type 1 edges (from the arbitrary beginning $\phi^+_{7,i}$), so it is nonorientable, and node 0 is adjacent to every other node (ibid.) in ϕ_s, as desired.

□

Now, using **Lemma 4.4**, it becomes possible to strengthen **Theorem 4.2** and resolve Liu et al.'s open question.

Theorem 4.5. *Given any integers n, t such that $n \geq 6, 0 \leq t \leq n-4$, and $t \equiv \frac{1}{2}(n(n-5))$ (mod 2), there exists a face-simple, nonorientable $(n,t)-$quadrangulation which contains a hub node.*

Proof. By induction on n. Here, for base cases, we use all the embeddings from above 4.1. All of these satisfy the conditions of **Lemma 4.4**. In addition, we delete the node of degree 2 from the (7,2) and (7,0) embeddings to obtain (7,3) and (7,1) embeddings.

Then we consider that for any integer $n \geq 8$, the allowed values for t (that $0 \geq t \geq n-4$ and $t \equiv \frac{1}{2}(n(n-5))$ (mod 2)) with respect to either $n, n-4$ result in n, t having the same parity; no further accomodations are needed in order to satisfy the hypotheses of **Theorem 4.5**.

Then by inducting on n, meaning $n \geq 8$ and subsequently $t \geq 0$, we use **Lemma 4.4** on the $(n-4, t-i)-$quadrangulation.

There are three cases we have to consider for possible values of i. When $t = 0, 1$ then $i = 0$. When $t = 2, 3$ then $i = 2$. And when $t \geq 4$, $i = 4$. In each of these

cases, we can see that for this $(n-4, t-i)-$quadrangulation, $0 \leq t-i \leq (n-4)-4$. Since we started with base cases, this establishes that any arbitrary desired such quadrangulation exists by induction, as desired. $\square$

4.1.2 Orientable Surfaces

The previous proof for **Theorem 4.1** follows something called a "diagonal technique", attributed to N. Hartsfield [8], while in order to prove **Theorem 4.1** in a new way, the work discussed here instead uses the diamond sum operation.

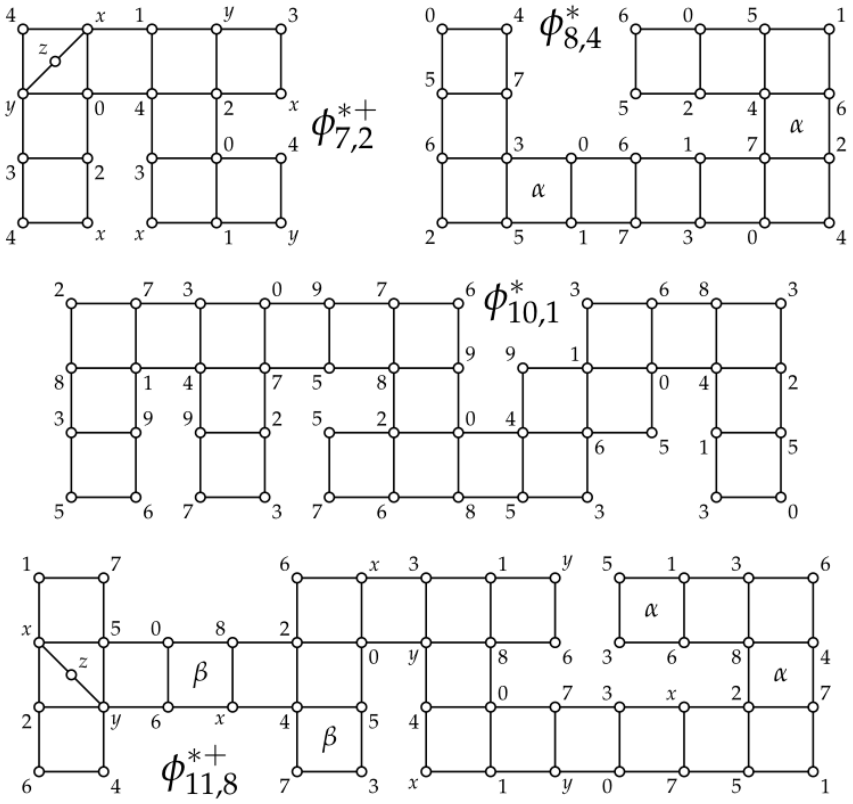

Figure 4.2: unfolded orientable quadrangular embedding base cases via [1]

The overall procedure follows very similar reasoning to the proof for **Theorem**

4.5 in that it takes an inductive approach from base cases using the diamond sum as the central constructive operation. There are more base cases required, and they are all shown in 4.2. We note that some necessary base cases can be derived from others, so the strictly necessary beginning embeddings can be kept to as small a set as possible, so long as we specify how to obtain the "secondary" base cases.

We first note that we have to use the "handle manipulation procedure" detailed in figure 4.3 in order to obtain two of the base cases (the embeddings $\phi^{+*}_{11,0}, \phi^{+*}_{11,4}$ by applying this handle manipulation to $\phi^{+*}_{11,8}$). Specifically, the pairs of faces which are labelled α, β are the targets for the handle manipulation.

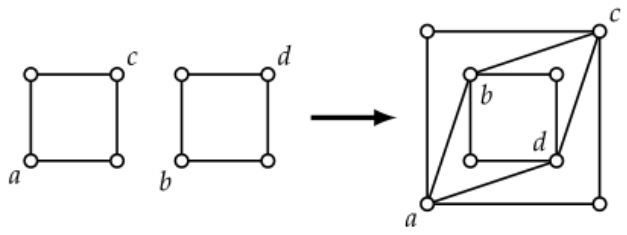

Figure 4.3: handle manipulation procedure via [1]

We use a hub node lemma similar to before:

Lemma 4.6. *Suppose ψ is an orientable $(n,t)-$quadrangulation which contains a hub node. For each $i \in \{0,4,8\}$, there then exists an orientable $(n+8, t+i)-$quadrangulation with a hub node.*

Proof. Denoting J_i as a graph derived by deleting i edges from K_8, there exists an

isomorphism between $\overline{K_2} + ((K_1 + J_i) \cup K_1)$ and $\phi^{+*}_{11,i}$. Then, similar to the proof approach for **Lemma 4.4**, take diamond sums of the embeddings $\phi^{+*}_{11,i}, K_{10,n-1}$, and ψ, in that order – passing through an "intermediate product" of a hub node containing embedding, and then ending with an embedding with the desired qualities.

□

Using $\phi^{+*}_{7,2}$ from 4.2, we can establish a helpful fact:

Remark. Suppose ψ is an orientable $(n, t)-$quadrangulation which has a hub node. Then there exists an orientable $(n + 4, t + 2)-$quadrangulation with a hub node.

This fact is helpful in building some of the secondary base cases.

Finally, we can walk through the different approach for a (slightly stronger, since it involves the additional condition of a hub node) version of Liu et al.'s orientable quadrangulation result:

Theorem 4.7. *Given integers n, t such that $n \geq 5, 0 \leq t \leq n - 4, t \equiv \frac{1}{2}(n(n-5))$ (mod 4), there is an orientable $(n, t)-$quadrangulation with a hub node.*

Proof. As an overall strategy, we use "primary" base cases from figures 4.1, 4.2 to obtain the secondary base cases, noting that once finished this set of base cases runs the gamut from $n = 5, ..., 14$. We can then use **Lemma 4.6** in successive rounds to get all desired higher order minimal quadrangulations. We also note that each embedding of this finished set of base cases contains a hub node.

Starting with $n = 5$, we can simply re-use $\phi^{*}_{5,0}$ from 4.1. There are no allowed values of t for $n = 6$. Deleting the degree 2 node from $\phi^{+*}_{7,2}, \phi^{+*}_{11,0}, \phi^{+*}_{11,4}$ gets us results

for $n = 7, n = 11$. The handle manipulation from 4.3 can be used on $\phi^*_{8,4}$ to get an $(8, 0)-$quadrangulation. For $n = 10$, $\phi^*_{10,1}$ from 4.2 is a $(10, 1)-$quadrangulation, while to obtain the allowed $(10, 5)-$quadrangulation we add a degree two node into one of the faces of $\phi^*_{5,0}$ to get a intermediate product of a $(6, 3)-$quadrangulation, then via **Observation 4.1.2** (i.e., a sneaky diamond sum) we can get the desired $(10, 5)-$quadrangulation.

We are left with $n = 9, 12, 13, 14$. These can be derived from the base cases already found, either via **Observation 4.1.2** or **Lemma 4.6**.

Finally, for the induction step, for any $n \geq 15$ we simply apply **Observation 4.1.2** to a $(n - 8, t - i)-$quadrangulation.

Specifically, if $t = 0, 1, 2, 3$ then $i = 0$; if $t = 4, 5, 6, 7$ then $i = 4$, and $i = 8$ if $t \geq 8$. Therefore an orientable $(n, t)-$quadrangulation of any arbitrary n and appropriate t can be obtained, as desired.

□

4.2 Establishing the nonorientable genus of the nearly complete bipartite graphs

As mentioned in the Previous Work section above, Mohar et al. establish the nonorientable genus of the nearly complete bipartite graphs, leaving only $G(n, n, n)$ and $G(2n + 1, 2n, 2n)$ as open questions, which is what these results fill in:

Theorem 4.8. *Given $n \geq 3, G(2n+1, 2n, 2n)$ has a nonorientable quadrangular embedding.*

Theorem 4.9. *Given $n \geq 6, G(n, n, n)$ has a nonorientable quadrangular embedding.*

The results discussed here are essentially wholly from (with permission and knowledge of the coauthors) and can be found in published format as well in [13].

Through the various processes, we diamond sum quadrangular embeddings of nearly complete bipartite graphs, the results of which are nonorientable (as well as quadrangular). This is because one or more of the input graph embeddings is nonorientable. We also (for the sake of consistency) diamond sum on left nodes (when considering labels, this is the group of nodes which belong to the m group) and thus black nodes (i.e., from the n group) will always be the ones merged.

We also recall that the "cleaning" operation involving node flips may be necessary when taking diamond sums, in order not to "drop" any type 1 edges in the diamond and thus accidentally turn an orientable embedding nonorientable (or the converse), or otherwise alter the output in inaccurate (and thus undesirable) ways.

We note in general that:

Remark. Up to reversal, the diamond sum operation preserves the rotation around all remaining left nodes.

Supposing that we take a diamond sum of two quadrangular embeddings $G(m_1, n, k_1), G(m_2, n, k_2)$, we can say that the result $G(m_1 + m_2 - 2, n, k_1 + k_2)$ is also a nearly complete bipartite graph, so long as we have the following:

- the two deleted left nodes were saturated (not missing any edges)
- when merging the right nodes in the process of the diamond sum operation, after the two "target" nodes have been excised, no two nodes are merged where both have missing edges

If these conditions are not met, then through an inductive construction process it cannot be guaranteed that the intermediate products are correct, since "missing edges" can accumulate in a cascading error process through the repeated diamond sums. In particular, since the definition of a "nearly complete" bipartite graph requires that at most, a full set of independent edges is not present in the graph, this means that should these conditions not hold, the end product could be have the correct number of nodes and edges, but not be a nearly complete bipartite graph; it would instead end up being just a *non-complete* bipartite graph, which is not a desired end result (since the set of missing edges might not be independent).

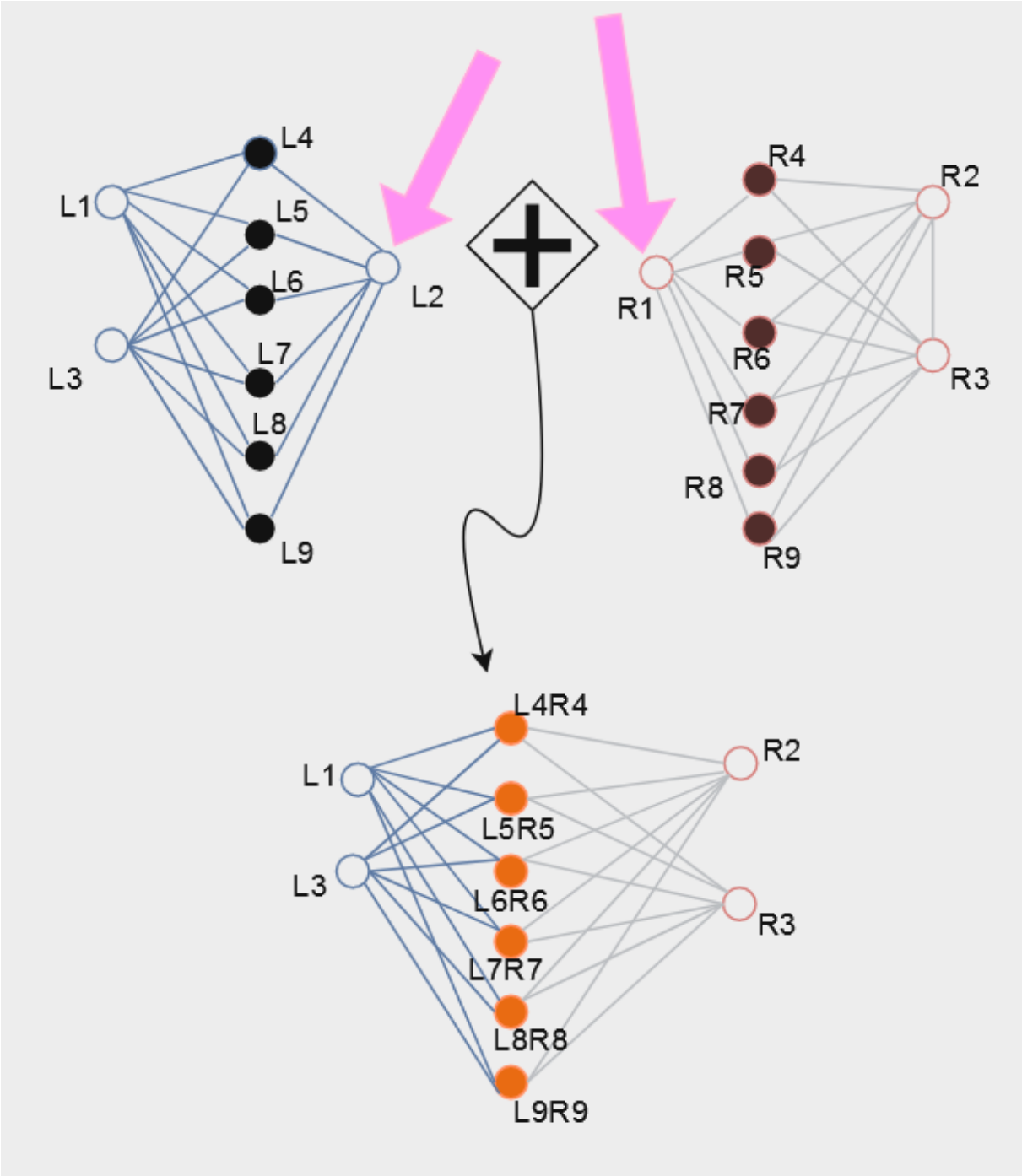

Figure 4.4: An example diamond summing of two instances of $K_{3,6}$ to form a $K_{4,6}$ product

In order to meet the second condition, it is useful to have information about the saturation (or lack thereof) on the right nodes of each of the input graphs. If such information is available and usable, then it becomes possible to to choose how to merge right nodes in the latter portion of the diamond sum operation in such a way that two unsaturated right nodes are not merged with each other.

This becomes directly relevant in the strategy taken to generate the graph $G(2n+1, 2n, 2n)$: beginning with $K_{n+1,2n}$ (in others words, the graph $G(n+1, 2n, 0)$ with *no* edges missing) and diamond summing this repeatedly with fresh instances of the graph $G(3, 2n, 2)$. We note that this means that each successive diamond sum of the intermediate, accumulating product with $G(3, 2n, 2)$ means introducing more missing edges into the accumulating result.

Our first task is to then establish that such successive diamond sums are possible without merging unsaturated right nodes together.

To that end, we consider:

Lemma 4.10. *Suppose $n \geq 2, p$ odd, and $p < 2n-2$. Then there is a quadrangular embedding of $G(3, 2n, 2)$ where there are p nodes in the rotation around the one saturated left node which separates (in cyclic ordering) the two unsaturated right nodes.*

Proof. Let q any quadrangular embedding of $K_{3,2n-2}$, and a, b, c the labels given to the left nodes. Since q is a simple graph, b, c as left vertices form the opposite corner

to the quadrilateral face for each of the faces "around" a, in an alternating fashion as one goes clockwise (without loss of generality) around in the cyclic rotation about a.

Then simply insert two more right vertices of degree 2 into two appropriately chosen quadrilateral faces, one of which will adjacent to a, c and the other to a, b. Now q is a quadrilateral instance of $K_{3,2n,2} = G(3, 2n, 2)$ with the choice of faces yielding any result for p as desired. □

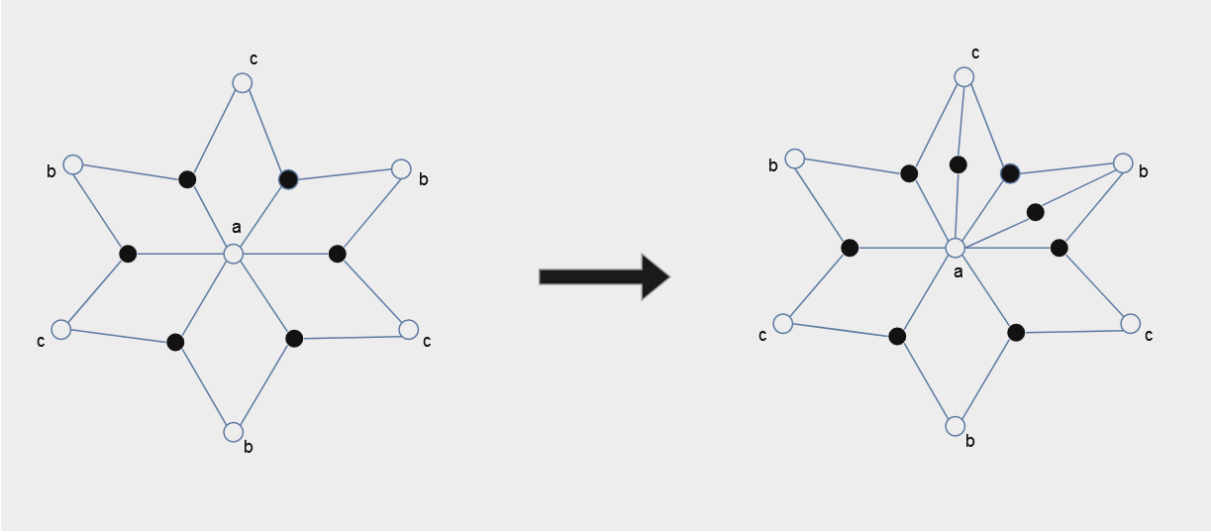

Figure 4.5: Constructing $G(3, 8, 2)$ from $G(3, 6, 0)$ with $p = 1$ (or $p = 5$)

Using the process of "node insertion" described in **Lemma 4.10** repeatedly gives rise to the notion of an *odd pairing*. A quadrangular embedding of the form $K_{m,2n}$ where $m \geq n$ is said to have an odd pairing if there is a partitioning of the right

nodes into pairs, such that if each pair is assigned to a left node α, then the two members of the pair are separated by an odd number of vertices in the rotation about α.

This allows us to quickly describe an embedding as being able to be used in successive node insertions as in **Lemma 4.10**.

We note that because the diamond sum excises its two target nodes (one from the each input graph), once a left node is targetted by a diamond sum, it cannot be targeted again by a successive diamond sum without errors in the results.

This odd pairing notion leads us to:

Lemma 4.11. *Let q a nonorientable, quadrangular embedding of $K_{m,2n}$ with $m \geq n$, having the odd pairing quality. Then there is a nonorientable quadrangular embedding taking the form $G(m+n, 2n, 2n)$.*

Proof. Let q_1 the initial, nonorientable quadrangular embedding of $K_{m,2n}$ with $m \geq n$, and having the odd pairing quality. Using **Lemma 4.10** there is a quadrangular embedding $G(3, 2n, 2)$ with two unsaturated right nodes. The "node distance" separating these two unsaturated right nodes is, by the same lemma, some desired p. We can determine p initially by looking at q_1 and finding two right nodes the appropriate distance, then diamond summing q_1 with the $G(3, 2n, 2)$ such that the right nodes are merged appropriately. We obtain a nonorientable, quadrangular graph embedding of the form $G(m+1, 2n, 2)$. **Observation 4.2** allows us to repeat this process without worrying about changing the node distances of partic-

ular pairs across diamond sums. We then repeat this process, terminating with $G(m+n, 2n, 2n)$, as desired.

□

While we noted above that Ringel gave the first construction for nonorientable quadrangular embeddings of bipartite graphs, he wrote down all the faces explicitly, meaning we are not *prima facie* given which edges are type 0 or 1. But so long as we use the diamond sum operation as directed (meaning some node flips are necessary to avoid excising any nodes which have type 1 edges), **Lemma 4.11** is still usable, since we can always flip nodes as needed such that the particular node(s) of the graph(s) being excised do not have type 1 edges. We also note that these node flips don't change the distances between any of the neighboring nodes, so the odd number of nodes separating pairs of nodes in a rotation will still hold as needed.

We can now prove **Theorem 4.8**:

Proof. From each of Ringel's [11] rotation systems of $K_{n+1,2n}$ where $n \geq 3$, we find an odd pairing. Each of the right nodes we label $1, 2, ..., 2n$. For all but two of the left nodes, the rotation is simply the ascending sequence from $1, ..., 2n$.

We consider the following two cases:

1. *n is even:* the first two left nodes are assigned the rotation:

$$(1 \quad 2 \quad 2n \quad 2n-1 \quad 3 \quad 4 \quad 2n-2 \quad 2n-3 \quad ... \quad n+2 \quad n+1)$$

$$(1 \quad 2n \quad 2 \quad 3 \quad 2n-1 \quad 2n-2 \quad 4 \quad 5 \quad ... \quad n \quad n+1)$$

respectively. The right nodes can be paired off in clusters of four, every other node in the following way: $\{1,3\},\{2,4\},...\{2n-3,2n-1\},\{2n-2,2n\}$. The first pair $\{1,3\}$ is assigned to the first left node, then all other pairs arbitrarily to any of the remaining left nodes except for the second.

2. *n is odd:* the first two left nodes are assigned the same rotation of the ascending sequence $1,...,2n$ with the exception of $2,3$ which are swapped to give $1,3,2,4,...,2n-1,2n$. Then, the right nodes are paired off as

$$\{1,2\},\{3,5\},\{4,6\},\{7,9\},\{8,10\}...\{2n-3,2n-1\},\{2n-2,2n\}$$

with the first pair assigned to the first left node, and all other pairs to any nodes except for the second.

Using **Lemma 4.11** we get nonorientable quadrangular embeddings of the form $G(2n+1,2n,2n)$. □

□

Beginning this process with $K_{n,2n}$ instead of the stated $K_{n+1,2n}$ gives quadrangular embeddings of the form $G(2n,2n,2n)$. The $n=2$ case is the cube graph, which is planar. For $n \geq 4$ Mohar [10] describes an approach used in **Lemma 4.13** below. This leaves only the $n=3$ case to be described:

Proposition 4.12. *$G(2n, 2n, 2n)$ has a nonorientable, quadrangular embedding for $n = 3$.*

Proof. In Ringel's embedding of $K_{3,6}$, the left nodes have the rotations:

$$(1 \quad 2 \quad 6 \quad 5 \quad 3 \quad 4)$$

$$(1 \quad 6 \quad 2 \quad 3 \quad 5 \quad 4)$$

$$(1 \quad 2 \quad 3 \quad 4 \quad 5 \quad 6)$$

which allows for the use of **Lemma 4.11** on the odd pairing defined by $\{1, 3\}$ with the first left node, $\{2, 5\}$ with the second, and $\{4, 6\}$ with the third. This allows for three appropriate diamond sums, accumulating the desired number of left nodes and missing edges, and using up all instances of the initial, necessary left nodes for the three diamond sums needed. □

In order to prove **Theorem 4.9**, we use current graphs to get embeddings. Previous work [7] asserts that $G(4s, 4s, 4s)$ exists as a family of orientable quadrangular embeddings, all of which are also current graphs. It would be possible to tweak that family and establish the desired result for the remaining family of nonorientable graphs, but instead we use a roughly diamond sum strategy, described by Mohar [10] as "Proposition 1":

Lemma 4.13. *Suppose $G(7, 7, 7)$ has a quadrangular embedding, then for all $s \geq 2$,*

$G(4s+2, 4s+2, 4s+2)$ *has a nonorientable quadrangular embedding.*

While Mohar used $G(5,5,5)$ to construct quadrangular embeddings of $G(4s, 4s, 4s)$ where $s \geq 2$, **Lemma 4.13** follows from using $n = 7$ instead of 5.

This result allows for the following proof of **Theorem 4.9:**

Proof. Using **Lemma 4.13** as well as **Proposition 4.12**, there exist nonorientable quadrangular embeddings of $G(4s+2, 4s+2, 4s+2)$ for any $s \geq 1$. We need to show only that $G(n,n,n)$ has a nonorientable quadrangular embedding when n is odd and greater than 5. We can do this through the explicit (current) graph generation, through the procedure outlined in the Implementation section.

The generated embeddings can either be checked for non-orientability and quadrangularity via node/edge arguments, the "closed walk with only one twisted edge" argument, or (the actual approach used) via implemented code checking (the spanning tree construction and node flipping procedure discussed in the Implementation section). This current graph generation procedure gives the desired nonorientable quadrangular embeddings of $G(n,n,n)$ with odd $n > 5$, as desired.

□

The results of proving **Theorems 4.8, 4.9** allow for the lower bound nonorientable genus formula for nearly complete bipartite graphs to be stated:

Theorem 4.14. *Given integers* $m, n, k \geq 0; m, n \geq 3; k \leq \min(m,n)$*, then the*

nonorientable genus $h(G(m,n,k))$ *of the graph* $G(m,n,k)$ *is:*

$$h(G(m,n,k)) \geq \left\lceil \frac{(m-2)(n-2)-k}{2} \right\rceil$$

996 *with the exceptions* $h(G(3,3,3)) = 0, h(G(5,4,4)) = 2, h(G(5,5,5)) = 3.$

Chapter 5

Conclusion

5.1 Limitations and Future Work

The diamond sum has shown its utility in inductive constructions for proving qualities of larger and larger families of graphs. Other graphs, such as $(n, t)-$quadrangulations with $t > n - 4$ could potentially be explored using the diamond sum in general, or the built functionality in particular.

It's also possible that the present work could be extended to explore the properties of nearly complete bipartite graphs, where the missing edges are **not** independent.

Minimal quadrangulations can also be explored further. For example, which graphs form minimal quadrangulations is not currently fully explicated in terms of minimal values or equivalence classes for n.

Regardless, given the ability to computationally build up iteratively larger graph

objects, the diamond sum allows for explorations into different types of graphs in the future beyond the work described here.

On a broader level, computational strategies for exploring mathematical results is, in the author's opinion, underutilized and highly underrated as a means for achieving novel results in theory; hopefully this will become a more widespread practice in the future.

Bibliography

[1] Sarah Abusaif, Warren Singh, and Timothy Sun. *Face-simple minimal quadrangulations of surfaces*. 2023. arXiv: 2304.09647 [math.CO].

[2] T.D. Parsons B. Mohar and T. Pisanski. "The genus of nearly complete bipartite graphs". In: *Ars Combinatoria* 20 (B 1985), pp. 173–183.

[3] André Bouchet. "Orientable and nonorientable genus of the complete bipartite graph". In: *Journal of Combinatorial Theory, Series B* 24.1 (1978), pp. 24–33.

[4] Jonathan L. Gross and Thomas W. Tucker. *Topological Graph Theory*. USA: Wiley-Interscience, 1987. ISBN: 0471049263.

[5] Nora Hartsfield and Gerhard Ringel. "Minimal quadrangulations of nonorientable surfaces". In: *Journal of Combinatorial Theory, Series A* 50.2 (1989), pp. 186–195.

[6] Nora Hartsfield and Gerhard Ringel. "Minimal quadrangulations of orientable surfaces". In: *Journal of Combinatorial Theory, Series B* 46.1 (1989), pp. 84–95.

[7] Mark Jungerman, Saul Stahl, and Arthur T White. “Imbeddings of hypergraphs”. In: *Congr. Numer* 29 (1980), pp. 545–557.

[8] Wenzhong Liu, MN Ellingham, and Dong Ye. “Minimal quadrangulations of surfaces”. In: *Journal of Combinatorial Theory, Series B* 157 (2022), pp. 235–262.

[9] Wenzhong Liu et al. “Quadrangular embeddings of complete graphs and the Even Map Color Theorem”. In: *Journal of Combinatorial Theory, Series B* 139 (2019), pp. 1–26.

[10] Bojan Mohar. “Nonorientable genus of nearly complete bipartite graphs”. In: *Discrete & computational geometry* 3 (1988), pp. 137–146.

[11] Gerhard Ringel. “Das Geschlecht des vollständigen paaren Graphen”. In: *Abhandlungen aus dem Mathematischen Seminar der Universität Hamburg*. Vol. 28. 3-4. Springer. 1965, pp. 139–150.

[12] Gerhard Ringel. “Der vollständige paare Graph auf nichtorientierbaren Flächen.” In: (1965).

[13] Warren Singh and Timothy Sun. *Settling the nonorientable genus of the nearly complete bipartite graphs*. 2023. arXiv: `2305.14048 [math.CO]`.