Computing Embodied Effort *in the* Constructible Design Space *of* Bobbin Lace

by

Nathaniel Joseph Elberfeld

B.S. Physics
The College of William & Mary, 2006

M.Arch.
Washington University in St. Louis, 2011

Submitted to the Department of Architecture in Partial Fulfillment of the Requirements for the Degree of

MASTER OF SCIENCE IN ARCHITECTURE STUDIES

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2020

Signature of Author: ____

Department of Architecture

May 8, 2020

Certified by: ____

Terry Knight

Professor of Design and Computation, Department of Architecture

Thesis Advisor

Certified by: ____

Caitlin Mueller

Associate Professor of Architecture and Civil and Environmental Engineering

Thesis Advisor

Accepted by: ____

Leslie K. Norford

Professor of Building Technology

Chair of the Department Committee on Graduate Students

Thesis Committee

Terry Knight, Ph.D.
Professor of Design and Computation, Department of Architecture
Thesis Advisor

Caitlin Mueller, Ph.D.
Associate Professor of Architecture and Civil and Environmental Engineering
Thesis Advisor

George Stiny, Ph.D.
Professor of Design and Computation, Department of Architecture
Thesis Reader

Computing Embodied Effort in the Constructible Design Space of Bobbin Lace

by
Nathaniel Joseph Elberfeld

Submitted to the Department of Architecture on May 8, 2020 in Partial Fulfillment of the Requirements for the Degree of Master of Science in Architecture Studies

Abstract

In recent years, research in design and computation has included processes of making as an expansion of the more established study of shapes with grammar formalism. This interest parallels a rise in craft practices as, perhaps, a counterpoint to the proliferation of digital fabrication in which fidelity to original specifications is considered crucial to the success of a project but whose means and methods are often obfuscated or of secondary importance. Making grammars (Knight and Stiny 2015), by contrast, offer an opportunity to examine one of the most important yet least understood considerations of a design: the effort it takes to physically produce it.

This thesis introduces embodied effort as a contribution from human beings or machines that includes the work, steps, routines, applied skill, cognitive processing, or other forms of output that directly contributes to the production of a design. To compute this effort, effort grammars are introduced to expand the formalism of making grammars to include an effort-cost tabulation that corresponds to moments of making. In these grammars, constructability is embedded in a design through an emergent topology in contrast to topologies that emerge through geometric optimization that may solve form or structural considerations but can be highly effortful and costly, or impossible to make.

As a case study for computing embodied effort, an effort grammar is developed for a textile production technique called bobbin lacemaking to show how a limited set of making rules can achieve an infinitely variable, complex, and constructible design space. The grammar is used in conjunction with primary sources to identify the physical and cognitive effort required in each step of making bobbin lace and a mathematical model for calculating this embodied effort is introduced. A computer program is written to automate the rules and effort computation on-the-fly and an exploration of the design space is discussed. Effort is situated as critical consideration of contemporary design practice.

Thesis advisor: Terry Knight
Title: Professor of Design and Computation
Thesis advisor: Caitlin Mueller
Title: Associate Professor of Architecture and Civil and Environmental Engineering

Acknowledgements

It is a pleasure to work with those whom you both respect and trust, and for that dynamic I am especially grateful to my thesis committee:

Terry Knight for her thoughtful and careful analysis and expert guidance during thesis-related studies. Her dedication to scholarship is remarkable.

Caitlin Mueller for working with me for two years and introducing me to many of the concepts presented in these pages. She has been generous with her time and generous in spirit.

George Stiny for showing that everything ordinary is a special case of the extraordinary.

I would also like to thank Larry Sass for his friendship and for the conversations that influenced this thesis. Paul Pettigrew has also been a friend, and is a great resource for the school. Neil Gershenfeld has condensed much of the Universe into two thrilling semesters.

Among all of the staff that keep the school running and do the work of many: Cynthia Stewart, Andreea O'Connell, and Renée Caso. Inala Locke in Design & Computation has been especially helpful to our group and always a joy to talk to.

I would also like to acknowledge my outstanding colleagues in the S.M.Arch.S. program. One of the best aspects about M.I.T. is that everybody else is so great at what they do that you have no choice but to be yourself. Among the many friendships I am grateful for: Anna, Maria, Michael, Rodrigo, Andrew, Kyle, Lukas, Darle, Mengqi, Gideon, Katie, Zach. Within our year in Design & Computation: Jim, Nof, Molly, Shaoying, Yichen, Yuxuan.

I would like to thank the elders: Eli, Delanie, Paloma, Carlos, Athina, Alex, Diego, Lavender. In the pursuit of doctoral degrees, they have fostered an exceptional environment in which to research design.

I am grateful for financial support from the M.I.T. Department of Architecture and for a year of work in the Self-Assembly Lab.

I am fortunate to have known Lavender for over a decade and without whom I would know nothing of bobbins. I am grateful for Frank's generosity and humor.

I thank my family for nurturing a spirit of curiosity and for the loving support of many kinds I have received in my lifetime: Mark, Russell, Leah, Sung Hui, and my parents Katie and Richard. I am indebted to my partner Alexandra for the love and friendship that has made these studies tenable and for her vision and talent that have been a steady source of inspiration for years.

The sudden exodus from campus left many conversations unfinished and ideas yet to be articulated. When it is safe to do so, I hope to remedy the situation with members of this wonderful community at the source—by which of course I mean the *Muddy Charles*.

Contents

Thesis Committee 3
Abstract 5
Acknowledgements 7

PART I

Conceptual Background

1.1 Introduction 13
1.2 Shape, Making, and Effort Grammars 14
1.3 Grammar-Based Analysis 15
1.4 Effort Analysis 17
1.5 Embodied Effort in Bobbin Lace 18
1.6 Roadmap 19

PART II

Textiles Background

2.1 Textiles Background 21
2.2 History of Bobbin Lace 21
2.3 Bobbin Lace Structure and Terminology 24
2.4 Making Bobbin Lace 27
2.5 Recent Work and Personal Interest 29

PART III

Computing Embodied Effort

3.1 Effort Rules & Computation 33
3.1 (a) High-Resolution Computation 33
3.1 (b) Directed Graph Representation 46

3.1 (c) Low-Resolution Making Grammar 48
3.2 Quantifying Embodied Effort 57
3.2 (a) Primary Sources 57
3.2 (b) Formalizing Embodied Effort 64
3.2 (c) Quantifying Effort Components 70
3.3 Effort Calculations of Lace Designs 70
3.3 (a) Manual Calculations of Existing Lace 70
3.3 (b) Automated Calculations of Generated Lace 76
3.4 Selections from the Design Space 85
3.4 (b) Different Designs with Equivalent Effort 95
3.4 (c) Similar Designs with Different Effortt 106

PART IV

Discussion

4.1 Contributions & Future Work 111
4.2 Concluding Remarks 112

Bibliography

115

Appendix

119

PART I

Conceptual Background

1.1 Introduction

In recent years, research in design and computation has included processes of making as an expansion of the more established study of shapes with grammar formalism. This interest parallels a rise in craft practices as, perhaps, a counterpoint to the proliferation of *digital fabrication* in which fidelity to original specifications is considered crucial to the success of a project but whose *means and methods* are often concealed or of secondary importance. *Making grammars* (Knight and Stiny 2015) by contrast, offer an opportunity to examine one of the most important yet least understood considerations of design: the effort it takes to transform an abstract representation into a physical instance. This *embodied effort*, whether contributed from human beings or machines, is the work, steps, routines, applied skill, cognitive processing, or other forms of output that directly contribute to the physical production of a design.

Not only is effort required to produce a design, it informs the *design space* in which it resides. Robert Woodbury and Andrew Burrow's definition of the design space as "the network structure of related designs that are visited in an exploration process" (2006) is a generous description for the realm of possibilities that designers research relative to particular interests. An *effort-bounded design space* includes the designs which are achievable given available effort. For example, if a person exerts effort to knit a scarf of a particular design, then a knit scarf of that particular design is in the effort-bounded design space for the knitting person, along with every other scarf the person is able to make. Outside of the design space might be, for example, scarves made from methods of knitting unknown to the person, or scarves made by other people. In an effort-bounded design space, designs requiring the same amounts of effort are in the same *equivalence class*.

To the extent that an *effort* contributes to a visual change in a design, the grammar formalism is well-suited to discovering the *effort-possible* forms in the design space.

1.2 Shape, Making, and Effort Grammars

Introduced in 1972 by George Stiny and James Gips, *shape grammars* are a computational production system in which a set of *shape rules* generate and define a set of designs. Each rule consists of two shapes on either side of an arrow that points from left to right. In a *shape computation*, a rule is applied when the shape on left-hand side of the arrow matches a shape in a design in progress. The matched shape can then be replaced with the shape on the right-hand of the arrow. Even with a small set of shape rules, a large design space is defined as new candidates for shape replacement propagate through the computation. In contrast to a classical system of design spaces, in which each instance is preconceived through symbolic calculation, the design space of shape grammars is open and improvisational (Charidis 2017).

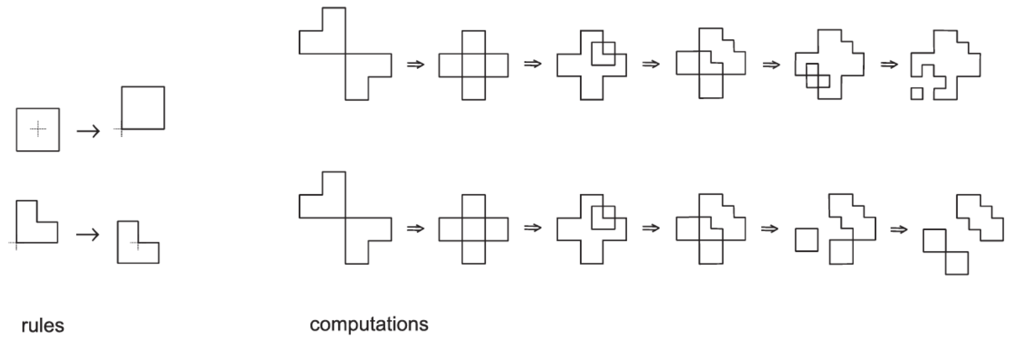


Figure 1.1: *Two shape grammar rules and two possible computations (Knight and Stiny, 2015).*

Making grammars extends the theory of computing with shapes to a theory of computing with "things." In these grammars, making is "*Doing* and *Sensing* with *Stuff* to make *Things*" (Knight and Stiny 2015). Doing and sensing are an interplay of physical actions such as drawing, knotting, or throwing and sensorial perceptions including touching, hearing, and seeing (2015). *Stuff* is material with physical properties that constitutes *things* after a *doing* transformation. An example is knotting (doing) and touching (sensing) with strings (stuff) to make knots (things) (2015).

Making grammars are of the same form as shape grammars, with the replacement arrow between shapes indicating a doing and/or a sensing action. These actions represent a discretization of an otherwise continuous process of making and, when considered in the context of effort, correspond to effortful moments of making. It should be noted that any discretization of an analogue process is subject to interpretation and is one of many ways by which continuous time can be segmented (Knight and Stiny 2015).

A feature that shape grammars, and now making grammars, share is a memoryless computation that propagates forward without predefined, and therefore limiting, symbolic, or nonvisual representations. Whereas classical computation requires a symbolic representation of an instance in order to act on it, grammars allow for the unexpected and can compute with the simple application of a replacement rule acting on points, lines, planes, and solids. As these elements interact under rules, complex and unforeseen designs emerge. In this "you perceive what you perceive" approach, calculating is visual.

In *effort grammars*, making grammars are developed to include an *effort-cost tabulation* that corresponds to a moment of making. Any moment of making necessarily requires effort. Certain ways of making will require more or less effort than others; some will require an equivalent effort but result in different designs. In this way, effort grammars are well-suited to the emergent design space of generative grammars: you do what you do.

1.3 Grammar-Based Analysis

If a strength of generative grammars, including shape, making, and effort grammars, is a computation without symbols, it is also one of the challenges in using grammars for numerical analysis. Analysis of design with numerical methods requires a numerical representation, for example. Two trajectories pursued development of shape grammars towards this end: a *parallel* approach and an *integrated* approach (Knight 2015). Towards a general theory of design, the shape grammar formalism was expanded to describe characteristics of designs in addition to shape. In the first paper to expand shape grammars with further descriptions, Stiny (1981) introduced functions to describe features of a design aside from, in parallel to, a shape representation, such as purpose, cost, or form (Knight 2015). An application of this approach was published in 1999 in a paper that paired grammar rules for the design of a coffeepot with associated costs of manufacturing (Agarwal, Cagan, and Constantine 1999).

Another trajectory for the expansion of shape grammar formalism is an inclusion of qualities in the shape specification. First proposed by Terry Knight, *color grammars* (1989) integrate a quality component within the shape specification as opposed to a parallel description of it. This concept was later generalized to *weights* (Stiny 1991) in which shapes are colored, textured, or assume other graphical qualities (Knight 2015). These qualities could be *ranked* numerically (2015) and, in an application of weighted grammars, were used to generate designs for micro-electromechanical system (MEMS) resonators (Agarwal, Cagan, and Stiny 2000).

Difficulties lie in the nature of an emergent design space; as unexpected outputs become inputs during a computation, the design space is constantly reconfigured. Recent work in shape grammars has investigated the nature of such a design space (Charidis 2017) and there has been recent work in grammar-based design space analysis in the engineering field of Computational Design Synthesis. In CDS, designs are generated by a computer to both reduce tedium and to offer novel solutions to engineering problems that are (relatively) unbiased by, for example design fixation or limited knowledge (Königseder, Stanković, and Shea 2016). In a 2016 paper, Corinna Königseder, Tino Stanković, and Kristina Shea recently adapted *transition graphs* for the purpose of gearbox synthesis, a common engineering design problem (2016). In the study, gearbox designs are generated from grammar rules and each design is mapped to a *node* connected by *edges* that represent the application of grammar rules that transform one design into another. This analysis gives the designer a consolidated and higher-level understanding of the rules and their implications relative to a standard tree-based representation. This approach can answer questions of reachability, that is if certain designs are possible under certain rules (2016). Applying this strategy to effort, it could be possible to determine if a particular design is *effort-possible* given a limited supply of effort.

In structural engineering, grammars have been developed to incorporate structural and fabrication knowledge, (Mitchell 1991) and to use a technique called *shape annealing* in which shapes are encoded with performance criteria and searched over an objective function (Cagan and Mitchell 1993). Caitlin Mueller notes that these approaches are best suited for post-conceptual development because the results find designs within a narrow problem space, such as a particular engineering typology (Mueller 2014). In her Ph.D. dissertation, Mueller used structural grammars to generate trans-typology bridge configurations and designs that "formulate broad, diverse design spaces that can generate unexpected and innovative design alternatives for conceptual structural design" (Mueller 2014). On the difficulty of reconciling the emergent design space of grammar-generated designs with techniques common in symbolic computation, Mueller states: "[o]f particular difficulty is the incorporation of grammar-based design spaces, since the majority of work in optimization, evolutionary algorithms, and machine learning centers around the parametric design vector" (2014).

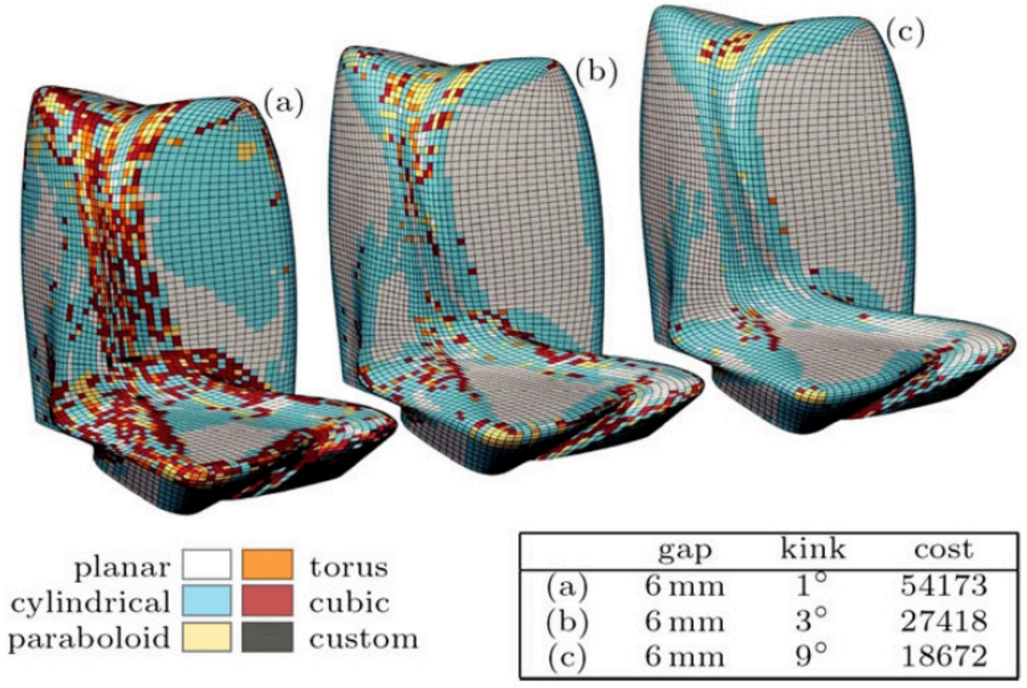
In order to evaluate the non-arbitrary consequences of effort propagating through a design process, a limited ruleset should be established. Without constraints, only the number of atoms in the universe and their interactions could bound a physical design. Similarly, with an inexhaustible supply of effort, any number of designs are possible simply through scaling—a bricklayer could build a wall that is 5 feet long or 10 feet long, and so forth. Instead, what is useful is a limited supply of effort operating within a limited ruleset of valid operations. A ruleset that allows discretized quantities of effort and produces continuously differentiated results avoids the combinatorial limitations of a single effort for a single outcome. Laying a yellow brick road, for example, requires the same effort as laying a red one, or any other color. This relationship between a particular design in a continuous spectrum of designs and a means of producing it without incurring extra costs is a feature of *mass customization* (Conner et al. 2014) and serves as a motivation for this thesis.

1.4 Effort Analysis

In making and effort grammars, constructability is embedded in the design space. A design that is produced with an effort grammar can by definition be made. By contrast, many geometrically optimized topologies solve form and structure considerations but are difficult, costly, or impossible to make. In architecture, there has been research into *construction-aware design* (Wallner and Pottmann 2011), but the methods presented are retrospective by nature. Regarding free-form architectural enclosures, for example, Johannes Waller and Helmut Pottmann state that "it is safe to assume the architect has firm ideas on their shape! We seek a decomposition of the facade into pieces which are easily manufacturable" (2011).

Recent work in construction cost estimation leverages Building Information Modeling (BIM) to track productivity—a related concept to that of effort— during construction by attaching data to predefined geometries in the model (Lee et al. 2017). This method is dependent on the resolution of detail in the model —which is then analyzed retrospectively; therefore, a criticism of BIM cost estimation is the lack of specificity that leads to inaccurate modeling predictions. A 2017 article cites "the main criticism of cost estimating in BIM is that cost estimators, with some justification, have less confidence in the level of detail of BIM designs. That lack of specificity can lead to substantial inefficiencies and reworking of estimates" (Ramos 2017).

By contrast, identifying effort at a corresponding moment of making allows for an *on-the-fly* knowledge of embodied effort at the point of computation. To understand how a visual state of a design changes from an exertion of effort, an example is needed.



	gap	kink	cost
(a)	6 mm	1°	54173
(b)	6 mm	3°	27418
(c)	6 mm	9°	18672

Figure 1.2: Optimal decomposition into panels (Wallner and Pottmann 2011).

1.5 Embodied Effort in Bobbin Lace

Textiles are well-known for their simple and algorithmic rulesets, but also for their association with complexity and difficulty. Handmade rugs may cost tens or hundreds of thousands of dollars, much of which is attributed to the time, and effort, it takes to create them. This dual—simplicity and complexity—renders textile creation an interesting case study for understanding the effects of effort in producing a design. In particular, *bobbin lacemaking* is a technique in the textile arts in which threads are braided together using a limited set of maneuvers, but which results in an infinitely variable and complex design space for pattern creation.

In bobbin lacemaking, threads are first wound from their ends onto pairs of handheld spools called *bobbins*. Designs typically employ 50-100 threads, but can consist of several hundred. Pairs of bobbins are then hung from an anchoring pin at the top of a permeable work surface, such as a firm pillow (bobbin lace is also referred to as *pillow lace*), onto which a pattern has also been attached. After this preparation, the lacemaker proceeds to braid two adjacent pairs of threads four threads altogether at a time, by maneuvering the bobbins above or below their neighbors until the lace is complete.

Historically, great value has been attributed to lace based on the fact that it is labor intensive, and thus rare and exclusive. In the production of bobbin lace, effortful moments of making include the preparation of bobbins and the work space, the movement of the bobbins during braiding, and the cognitive processing required to manage the complexity of the pattern. At intermittent steps, threads need to be detangled, tightened, organized, counted and the emerging piece inspected and corrected for errors. Time is also a consideration, as each piece, depending on the complexity and density of the pattern and can take hours to years to complete.

In an argument familiar to architects, Gottfried Semper in the *Four Elements of Architecture* (Semper 1851) posited wickerwork and carpet enclosures to be the essential origins of architectural space. Such entwinement with history is a parallel interest to craft practices as they relate to design; however, it is not the focus of this thesis. Rather, it is the computational aspect of textiles that is examined. Previous work in this area includes the development of weaving grammars (Muslimin 2014) and material computing with knitted assemblies (McKnelly 2015). A contribution of this thesis is to build on previous work in computation research related to textile production and introduce a method for calculating effort in it.

1.6 Roadmap

In Part II, I provide an historical overview of bobbin lace and its relationship to other methods of textile production. I review the fundamentals of making bobbin lace, lace structure, terminology, and different traditions within bobbin lacemaking.

In Part III, I develop a method to calculate embodied effort in bobbin lace and provide several examples of doing so. First, I develop a "high-resolution" making grammar detailing the effortful steps in the production of a particular lace. Using a mathematical description of lace, I then show that the making grammar can be generalized to describe other lace topologies, and I provide a "low-resolution" making grammar as an example. A review of primary sources elucidates further considerations regarding effort in lacemaking and is used, in conjunction with the making grammars, to develop a mathematical model for calculating embodied effort in bobbin lace. The low-resolution grammar is used as a basis for a computer program to simulate and automate the production of makeable bobbin lace designs and to calculate the embodied effort of each. Finally, I use the program to explore the design space of bobbin lace and find designs that are in the same effort-equivalence class but are visually distinct and I compare designs across equivalence classes to show that there are designs with similar visual qualities, but require different expenditures of effort to make.

I conclude in Part IV with a discussion of the work developed in this thesis and work proposed for future investigation. I discuss more broadly the implications of the ideas developed here.

The Appendix contains additional effort-cost tabulations and the code for the computer program.

PART II

Textiles Background

2.1 Textiles Background

From the Latin *texere*, "to weave," and *textilis*, "woven," a textile is a pliable material constructed from a network of intertwining fibers, here referred to as threads. There are many techniques to form a textile, including *weaving*, *knitting*, *crocheting*, and of interest to this thesis, *braiding*. The categories of textile production are characterized by how the fibers interact. In weaving, for example, two sets of threads—a vertical warp and a horizontal weft—are interlaced on a specialized machine called a loom. In knitting and crocheting, by contrast, a textile is produced by connecting loops of thread in rows using a knitting needle or small crochet hook. Braiding involves twisting two or more threads together. Additional means of textile production include *knotting*, *tatting*, and *felting*.

2.2 History of Bobbin Lace

Lace is a form of textile produced from knitting, crocheting, braiding, or weaving and is characterized by its delicate, web-like form. Diaphanous woven textiles and fine nets (in which threads are tied to one another at intersections) predate our current conception of lace, which sets its origin at the end of the fifteenth century in Italy. (Leader 2019) By the second half of the sixteenth of century, there was significant development in openwork textiles using a single thread and a needle or with multiple threads wound onto handheld spools called *bobbins*. Whereas *needle lace* was developed from embroidering techniques, *bobbin lace* is believed to have developed from a type of ornamental decoration called *passementerie*, in which gold, silver, or colorful silk braids are applied to velvet clothing or furnishings (Halley 2009a).

Early in its development, bobbin lace displayed formal complexity. *LePompe*, published in Venice in 1559, is the earliest known pattern book devoted to bobbin lace and displays multiple variant methods to *work* the lace (2009a).

Royal fashion and intermarriage facilitated the dissemination of lace across Europe and entrepreneurial manufactures sought innovative designs to supply a market need. By the beginning of the seventeenth century, Flanders, Spain, England, and France were centers of lace knowledge (2019).

Bobbin lacemaking in the eighteenth century achieved a zenith marked by delicacy, complexity, and quality in construction and design. Some of the simpler techniques in *LePompe* that used thick braids gave way to the extensive use of fine linens, sparse and delicate patterns, and notable scalloping details at the edges (2019).

The invention of machine-made net in the late eighteenth century influenced a style of lace in the mid-nineteenth century (and now classified as decorated nets) in which handmade motifs were appliqued to a fine net (Leader 2019). Lacemakers in the post-Napoleonic era sought designs with simpler structures that could be worked more quickly but, by the later part of the century, machines were capable of producing nearly all patterns that had been made by hand (Halley 2009a).

A revival period at the end of the nineteenth century witnessed a concerted effort among designers, merchants, and teachers to appropriate past styles to modern taste and simplify the construction methods in the process. By the end of the First World War, however, the production of handmade bobbin lace all but vanished, and it was no longer a sustainable means of employment (Leader 2019).

A resurgent, hobbyist interest in the mid- to late-twentieth century was marked by the foundation of several guilds dedicated to educating the public about lacemaking through publications, exhibitions, courses, and workshops. Among these organizations were The International Organization of Lace (1954), The Lace Guild (1976), and The International Bobbin and Needle Lace Organisation (1982). The 1976 publication of *The Technique of Bobbin Lace*, a pattern book with clear instructional diagrams, offered a counterpoint to the limited selection of patterns available at the time. The same year, polystyrene pillows became widely available and lowered the barrier to entry to bobbin lacemaking as the synthetic pillows were lighter to carry and did not require the extra step of fabricating straw-stuffed ones (Leader 2019).

In the last few decades, a proliferation of excellent bobbin lace pattern catalogues has been published including Bridget M. Cook and Geraldine Stott's *The Book of Bobbin Lace Stitches* (Dover, 1980) and Uta Ulrich's *Gründe mit System* (Barbara Fay Verlag, 2011), both of which have been extensively used in researching this thesis. More resources are listed in the Bibliography.

At present, there are many well-researched online resources for learning about and making

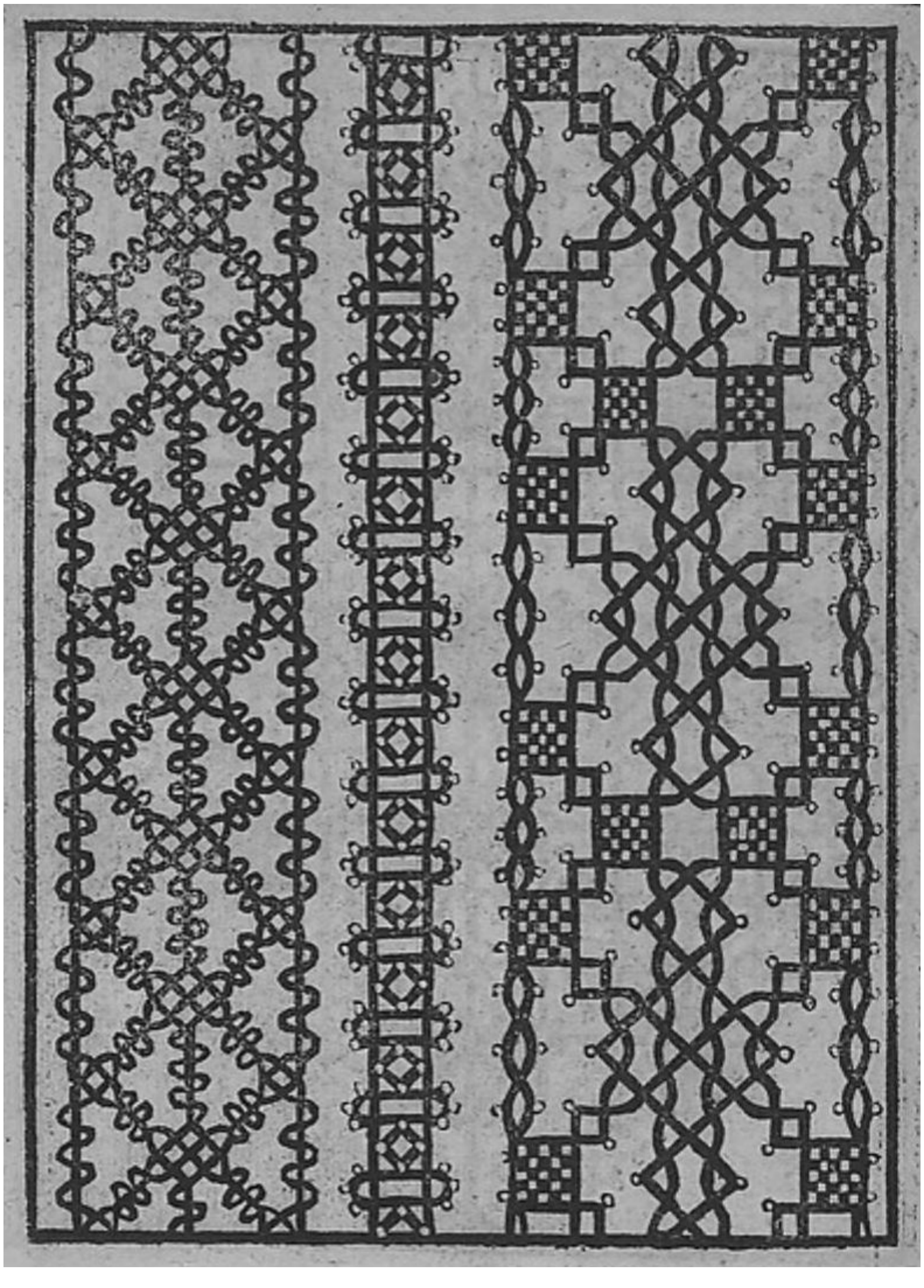


Figure 2.1: *A page from "LePompe," the earliest bobbin lace publication (1559).*

bobbin lace. Lorelei Halley (Chicago, USA) administers the websites Laceioli.ning.com and Lynxlace.com, and has documented hundreds of examples of lace from different periods and written about lace history in detail. Halley also provides resources for pattern-making, selling her own designs in addition to providing many freely available instructional tutorials. Jo Edkins (Cambridge, UK) has created many websites on topics ranging from mathematics to geography, and her eponymous Jo Edkins' Online Bobbin Lace School (Edkins 2017) offers many lacemaking tutorials, material suggestions, and detailed instructions complete with animations. Halley and Edkins' websites have been invaluable contributions to this thesis. The source for the content in Section 2.3 is from Halley's website articles "Bobbin Lace History – Overview" (Halley 2009a) and "Two Structural Classes of Bobbin Lace Distinctions of Style" (Halley 2009b). In turn, her website cites *Lace*: *A History* (Victoria & Albert Museum, 1983) and her own original research at the Art Institute of Chicago.

2.3 Bobbin Lace Structure and Terminology

Although threads are braided during their maneuvering in bobbin lace, Halley notes that bobbin lace is actually a form of weaving in which the warp, typically secured on both ends on a loom, are fixed only at the top and free to move relative to one another, allowing for much more complexity than traditionally afforded by fixed-warp weaving methods. The precision with which threads are individually maneuvered allows for structural and decorative design flexibility in bobbin lace, and many separate traditions developed with both shared and distinct features.

Bobbin lace designs typically employ less dense, lattice-like backgrounds, or *grounds*, in between more densely worked primary design features, or *motifs*. Grounds often occupy 10-85% of the design, depending on the style or tradition of lace, and many hundreds of grounds have been documented in the literature. The small, repeatable ground patterns are comprised of *stitches*, which are the fundamental unit of bobbin lace and may consist of one or more maneuvers in which threads pass over or under its neighbors. Most stitches may be used in grounds between motifs or in open areas within the design motifs, in which case they are referred to as *fillings*. A simple *half-stitch* resembles a basket-weave and is often used in parts of the lace which resemble woven cloth, referred to as *clothwork*.

Decorative motifs include geometric regions of higher density such as *diamonds*, *fans*, and *zigzags*, and more naturalistic motifs including *spiders*, *scallops*, *buds, shells, peas,* and *snowflakes*.

Part Lace *(Sectional Lace, Free Lace)*

Discrete Motifs

Threads are added and removed around motifs held together by sewings.

Honiton
Duchesse
Rosaline
Withof
Bruges

Meandering Tape

A continuous "tape" of cloth-work is held together by threads attached to completed parts.

Schneeberger
Idrija
Russian
Hinojosa
Milanese

Continuous Lace *(Straight Lace)*

Motifs and grounds are made at the same time from the same set of bobbins. Threads may travel through motifs, back to grounds, and vice versa.

Mesh Grounded

Two threads twisted together form the ground lattice.

Simple Clothwork

One pair enters clothwork at each pin.

Torchon
Point Ground
- *Bucks*
- *Tonder*
- *Bayeux*
- *Blonde*
- *Chantilly*

Complex Clothwork

Two pairs enter clothwork at each pin.

Binche
Flanders
Valenciennes
Paris
Mechlin
Antwerp

Plait Based

Four threads form the ground lattice. Motifs and grounds are indistinguishable. Also called *Braid Based* or *Guipure*.

Cluny
Matese
LePuy
Bedforshire

Figure 2.2: *Traditions in bobbin lace developed around structural categories. Redrawn from Lorelei Halley (2014).*

Often attached as a decorative trim to another fabric, the edges of lace are distinguished as *footside*, at the connecting side and often straight, and *headside*, on the free side and frequently scalloped or fanned.

Lace is broadly categorized by how it is structured. When the same set of bobbins is used for the ground and fillings in the entire lace, it is referred to as *straight* or *continuous lace*. In such designs, it is possible to trace individual threads from the top of the lace, through ground and motifs, to the bottom. Lace that is structured with *sewings* holding together discrete motifs or a wandering strip of clothwork, for example, is called *part lace.* Straight lace is further categorized as *mesh grounded*, in which two threads form the lattice of the ground, or *braid* or *plait based*, in which four or more threads form the lattice.

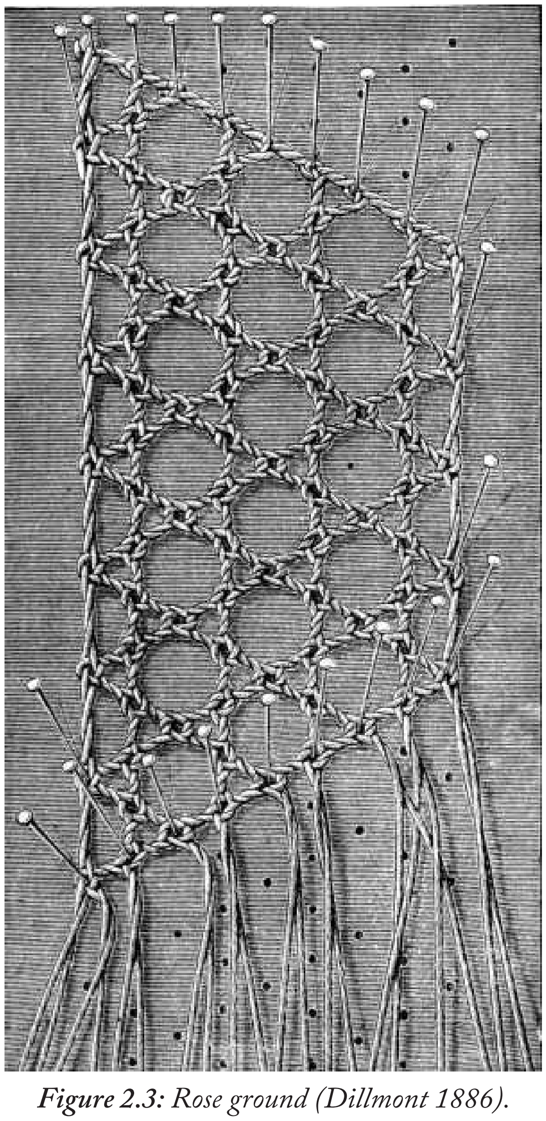


Figure 2.3: *Rose ground (Dillmont 1886).*

There is some discrepancy in lace tradition nomenclature. Lacemakers classify lace by its working methods and structure irrespective of where or when it was made. Curators, by contrast, are generally interested in a point of origin and date of production. Of particular interest to lacemakers is the way in which threads enter and exit clothwork. Mesh grounded lace is further distinguished as employing *complex clothwork*, in which two pairs of threads transition from the ground into the clothwork, or *simple clothwork*, in which one pair does the same. Additional structural or working methods can be used to further distinguish bobbin lace traditions. *Point Ground* and *Torchon* lace, for example, which are both straight laces of simple clothwork, can be distinguished by the flattened mesh lattice of Point Ground relative to the square diamond grid of Torchon. Additional distinctions can be made when classifying lace, including the use of a *gimp*, a thick thread that typically outlines a design motif, or which stitches are worked in the clothwork, although a thorough taxonomic review of all lace classifications is beyond the scope of this thesis.

Even at its early publication date, *LePompe* included plait-based straight lace, a part lace called *tape lace*, and the geometric *Torchon*, which will serve as the primary type of lace analyzed in this thesis.

2.4 Making Bobbin Lace

Bobbin lace is made in three phases: preparation, working, and finishing.

Preparation

Long threads are first wound from each end onto pairs of handheld spools called *bobbins*. Approximately four inches in length, bobbins are often made out of wood and consist of three parts: a narrow *neck* to hold the spooled thread, a *head* to prevent the thread from unspooling, and a *shank* for the lacemaker to grip. Some lacemakers attach beads to the bobbins as ballast to help keep tension in the threads.

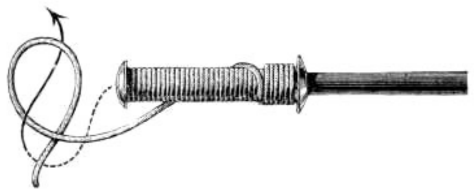


Figure 2.4: *Stoppage of the thread at the end of the bobbin (Dillmont 1886).*

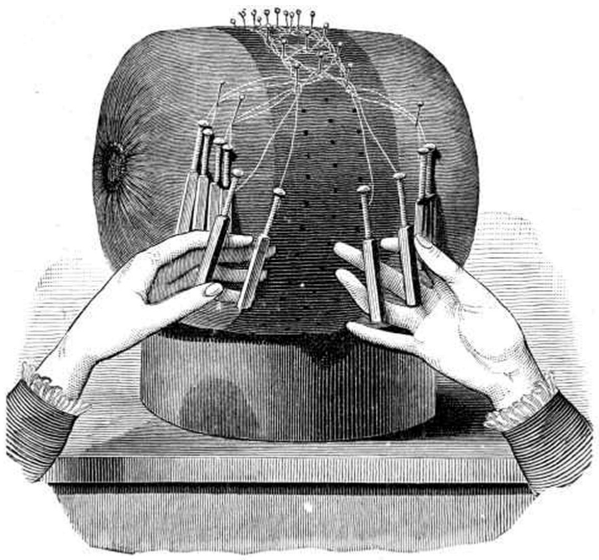


Figure 2.5: *Position and movement of the hands (Dillmont 1886).*

In traditional bobbin lace, pattern designs are transferred onto sturdy paper called *prickings*, which are perforated at the locations where *pins* will hold braided threads in place. Depending on the design, the pricking is affixed to either a disk-shaped *cookie pillow* or to a cylindrical *bolster pillow*, into which the pins will be pushed and secured.

After the pillow is prepared and the thread is spooled, the pairs of bobbins are suspended from pins in the top of the pattern and an initial stitch is worked to secure them.

Working

After preparation, the lacemaker proceeds to *work* the lace by braiding two adjacent pairs of threads together, four threads altogether, at a time. There are only two legal operations for this braiding: the *cross* (*C*) and the *twist* (*T*). The *cross* consists of moving the right-hand thread of the left-hand pair over the left-hand thread of the right-hand pair. The *twist* consists of moving the right-hand threads of each pair over the left-hand threads of its own pair. A slight variation of the twist is occasionally used in which only one of the pairs is twisted (*right-twist* or *left-twist*).

During a braiding sequence, the lacemaker might also *pin* (*p*) between two pairs in order to tension individual threads without affecting those nearby. The pin may occur in the middle of the braiding sequence (*closed pin*) or at the end of the sequence (*open pin*) and is inserted into the pillow through a perforation in the pricking. Some grounds can be worked without pins, while other grounds require pins to secure the braids in place.

After a sequence is completed on four threads, a new set is selected and a sequence is applied. The new set may contain one of the previous pairs, but may also consist of two new pairs depending on the pattern and how the lace is worked.

The cycle of selection and braiding continues until the pattern is realized.

Finishing

When a pattern is completed, the threads are tied off or woven back into the design and the bobbins are cut from the threads. The pins are removed and the lace is complete.

2.5 Recent Work and Personal Interest

In addition to a renewed interest in making bobbin lace among hobbyists, evidenced by the formation of guilds and online forums and mentioned in Section 2.2, there has been recent scholarship about bobbin lace from other fields.

Veronika Irvine and Frank Ruskey published *Developing a Mathematical Model for Bobbin Lace* (2014) in the *Journal of Mathematics and the Arts*, and created an exhaustive mathematical method for generating new grounds in bobbin lace. In the paper, Irvine and Ruskey build on work done in in textile topology by Sergei Grishanov, Vadim Meshkov, and Alexander Omelchenko who have shown that textiles are a special case of knots and whose periodicity may be represented as a diagram on a torus (2009).

In a paper entitled *Labor Optimization in Structural Bobbin Lace*, written at M.I.T. in 2018, I formulated a method for prioritizing less labor-intensive processes when making bobbin lace for structural applications. This research was in turn based on two experimental fabrications using bobbin lace and that I completed with colleagues.

In *Hedge* (2017) our team (Nathaniel Elberfeld, Lavender Tessmer, Jason Butz) suspended resin-hardened carbon-fiber laced panels from a steel trellis in the courtyard of the Contemporary Art Museum in St. Louis and loaded the panels with synthetic "vegetation" consisting of thousands of CNC-milled plastic shapes cumulatively weighing over a thousand pounds. Thirty braided panels arranged in two rows covered an area of approximately 38' by 10.5'. To our knowledge, this was the first such large scale application of bobbin lace for structural purposes.

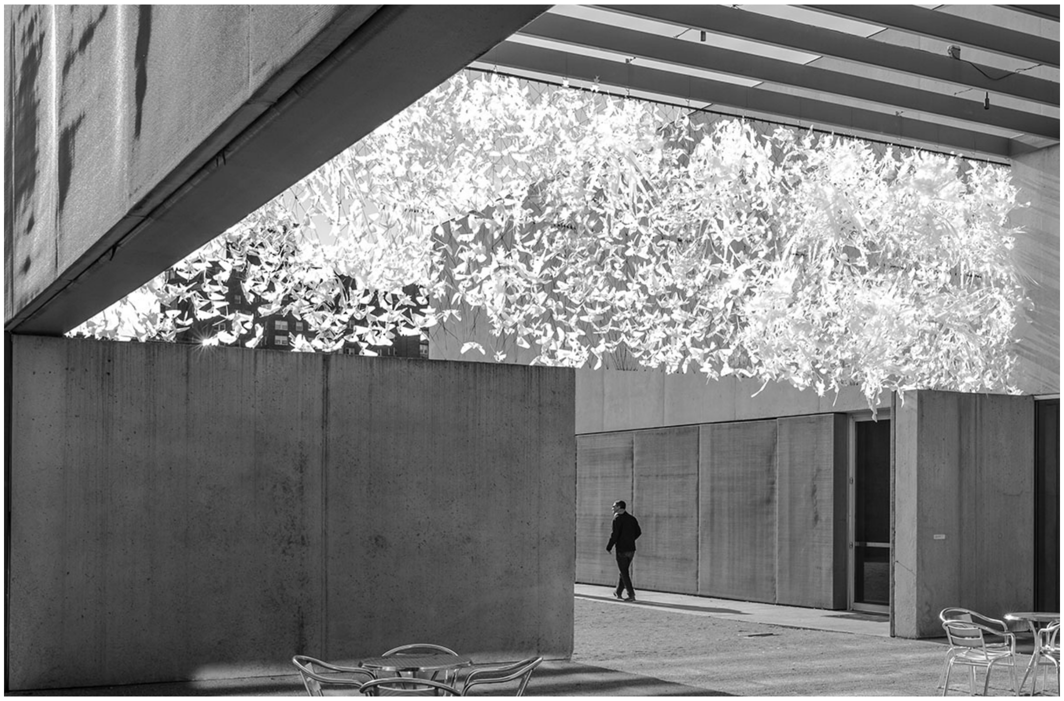


Figure 2.6: *"Hedge" by Nathaniel Elberfeld, Lavender Tessmer, and Jason Butz (2017).*

In *Concrete Tapestry* (2018) we (Nathaniel Elberfeld, Lavender Tessmer, Alexandra Waller) applied a coating of concrete to four large, laced panels each approximately 3' x 7' that were similarly prepared as in *Hedge*, but introduced localized subdivisions within the lace to give greater density, and structural integrity, in the self-supporting panels. We believe this to be the first application of bobbin lace for concrete reinforcement.

In both projects, the team noticed that certain processes were more difficult than others in bobbin lace. In *Hedge*, we discovered that when we wanted to make the lace denser for visual and structural reasons, it required more effort to add columns than rows because of the extra steps of preparation required for additional bobbins at new columns. In *Concrete Tapestry*, we found that the cognitive load in managing the complexity of the lace subdivision was especially taxing. Our relative inexperience at lacemaking and the large scale of the projects compounded the problems, and serve as an underpinning motivation for this thesis. In order to continue working with bobbin lace for experimental fabrications, a method for computing the embodied effort required to make them is needed.

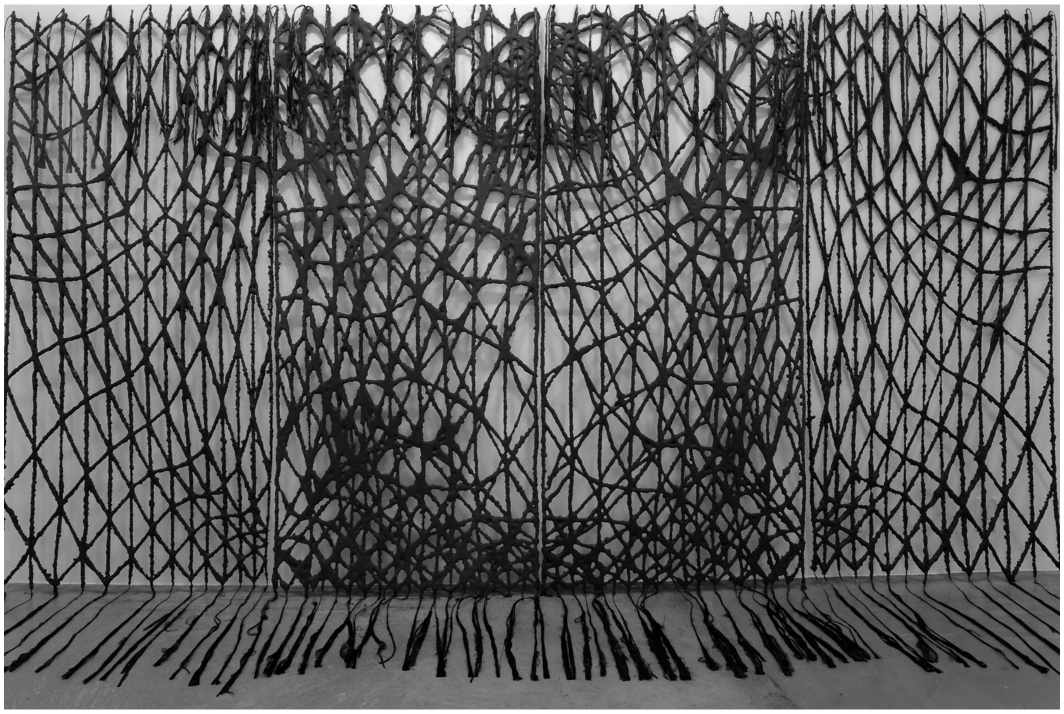


Figure 2.7: *"Concrete Tapestry" by Nathaniel Elberfeld, Lavender Tessmer, and Alexandra Waller (2018).*

PART III

Computing Embodied Effort

3.1 Effort Rules & Computation

In this chapter, I will introduce two methods of computing embodied effort in bobbin lace. The first method is a development of making grammars to include an effort-cost tabulation at each step of the making process. This method reflects more closely the actual process of creation and therefore I will refer to it as a "high-resolution" calculation of embodied effort.

The second method of computing embodied effort that I will introduce is a further abstraction of lacemaking concerned primarily with the emergent topology as the lace is created. Here, topology refers to the ways in which threads are braided over and under one another to make the shape of the lace. The effort calculations are informed by the "high-resolution" method, but are made implicit in the more abstracted representation of the making process. This method will be more general and more readily reconfigured to examine different lace designs. However, it does not reflect as closely the making process; I therefore refer to this method as a "low-resolution" calculation of embodied effort.

3.1 (a) High-Resolution Computation

To demonstrate the components of effort that are required in lacemaking, a high-resolution embodied effort computation is developed for a simple whole-stitch lace. This method of computation can be applied to other lace designs, but is limited to one example here for illustrative purposes.

The primary reason to develop the high-resolution making grammar is to identify the significant contributors to effort in lacemaking. Each step in the computation requires an input of effort by the lace maker. By the conservation of energy, if the lace changes shape then external energy (effort) must be applied.

In shape and making grammars, it is conventional to show the rules, followed by the computation as prescribed by the rules. It is my experience, however, that the rules and computation inform each other during development. This back and forth situates making grammars as a productive forensic device capable of finding the significant forms of effort in the making process.

As described in Part II, all bobbin lace is created from crossing, twisting, and pinning threads that are wound up onto a collection of bobbins and interact with one another throughout the design. These actions can easily be translated into a collection of making rules:

- Crossing action
- Twisting action
- Pinning action

Much of the literature and hobbyist guides include prototypical making grammars in the prolific use of diagramming to illustrate these core actions.

Each bobbin used in a design will need to be wound with thread and mounted to a pin at the top of the design.

- Winding of bobbins
- Mounting of bobbins

The bobbins and thread also need to be manufactured, but typically the lacemaker will purchase these components and thus we can curtail the recursive process of identifying every component of making at a reasonable level.

Throughout the lacemaking process, the lacemaker will need to maintain the design by tightening the threads after each interaction.

- Tightening threads

At the end of the lace, the bobbins need to be cut and tied off and the pins need to be removed.

- Tying off ends
- Removing pins

The components mentioned above identify motor effort, operate discretely, and are readily correlated with a making grammar. However, there is also cognitive effort, including perceptual effort, to consider in the production of lace. The tangle of bobbins is difficult to manage and small mistakes—crossing instead of twisting, for example—will affect the structural and visual integrity of the lace and can only be fixed by moving backwards (also subject to errors) through each step to rectify them. We can therefore identify two more components of effort:

- Identifying active threads
- Recalling stitch pattern

These components of effort, whether motor or cognitive, are reformulated below as a set of making rules. Each instance of them is accounted for in the following computation.

High-Resolution Making Rules & Computation

The high-resolution making rules begin with selecting the size of the lace and identifying an adequate working space (Rule 1). The rules are grouped together by related actions or phases of lacemaking: the pinning and unpinning actions (Rule 2, Rule 3), the winding and mounting of the bobbins (Rule 4, Rule 5), the cross, twist, and one-pair twist (Rule 6, Rule 7, Rule 8), the identification of active threads (Rule 9A) and the tightening of them (Rule 9B), and the finishing procedures of tying off the bobbins (Rule 10), and removing the workspace markers (Rule 11, Rule 12).

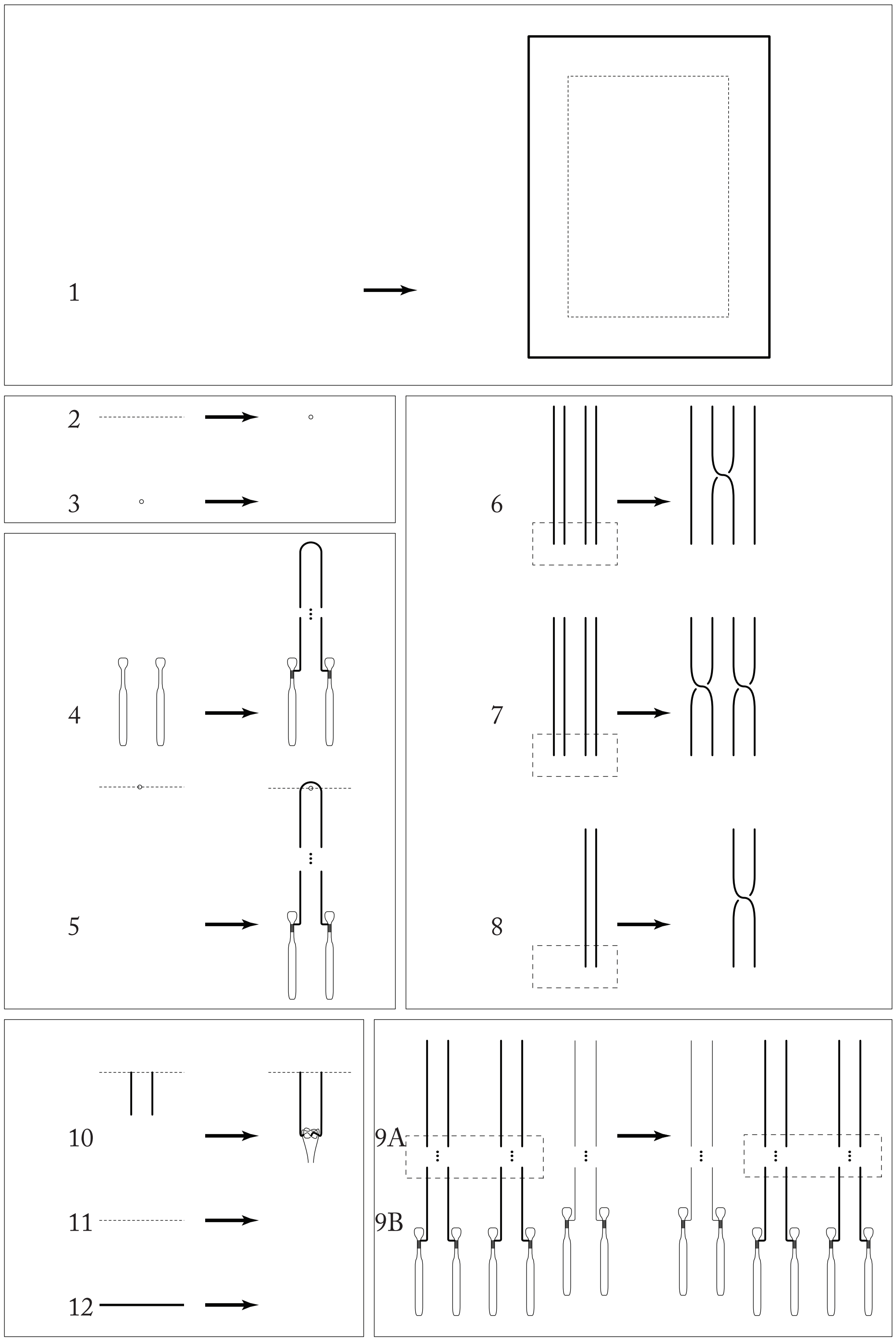


Figure 3.1: High-resolution rules for bobbin lace.

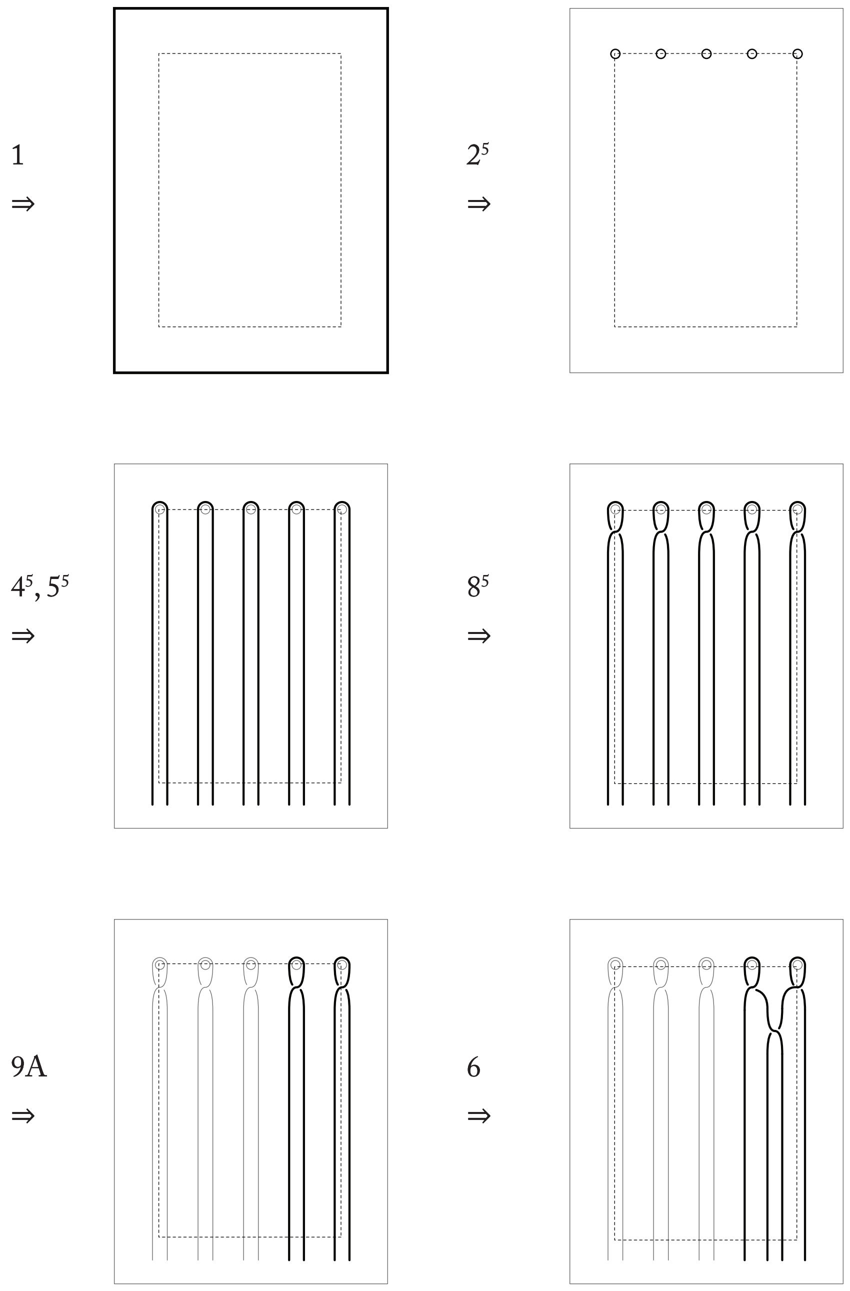
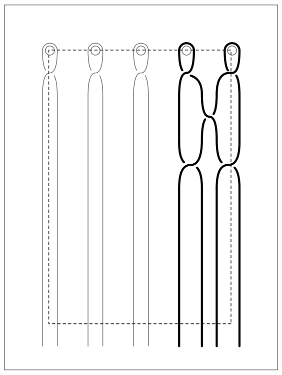


Figure 3.2: *High-resolution computation.*

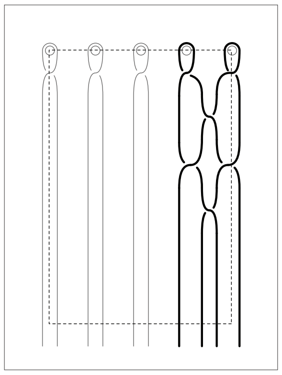
7

⇒



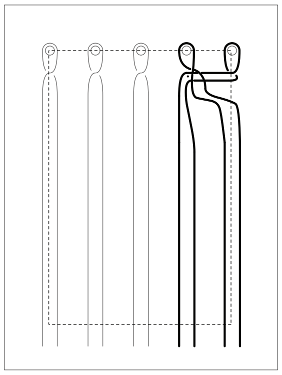
6

⇒



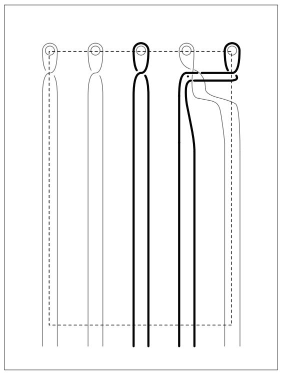
9B

⇒



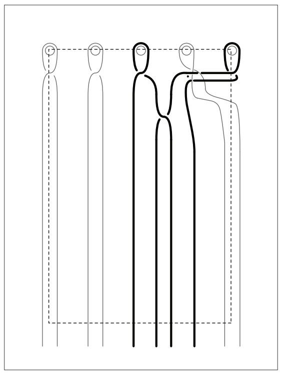
9A

⇒



6

⇒



7

⇒

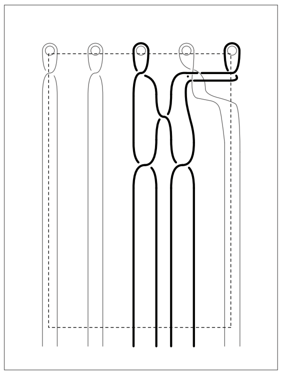
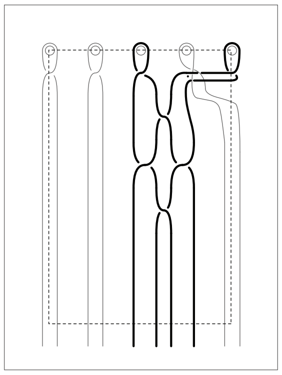
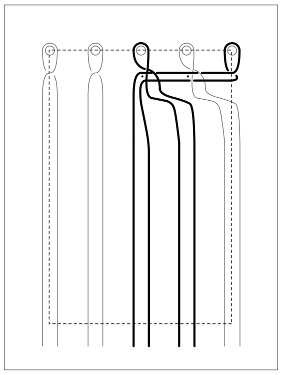


Figure 3.2 (continued): *High-resolution computation.*

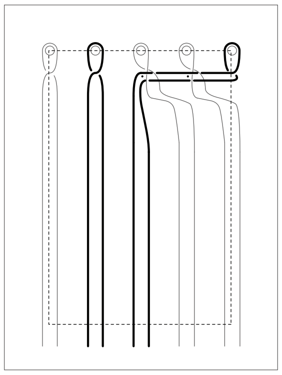
6
⇒



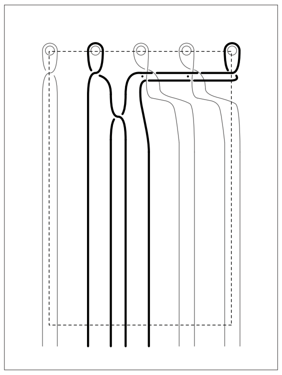
9B
⇒



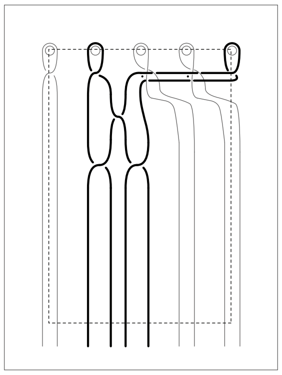
9A
⇒



6
⇒



7
⇒



6
⇒

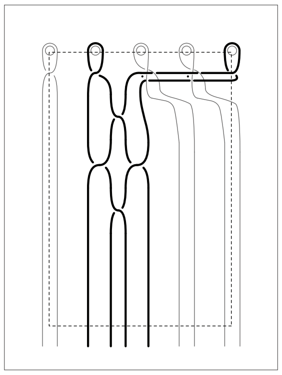
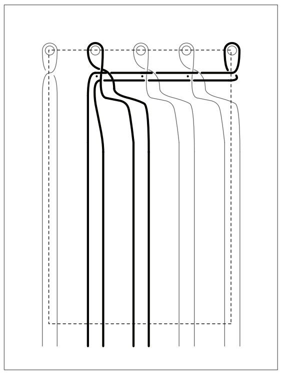


Figure 3.2 (continued): *High-resolution computation.*

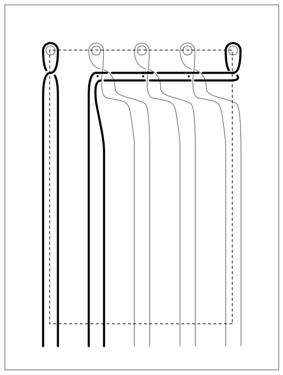
9B

⇒



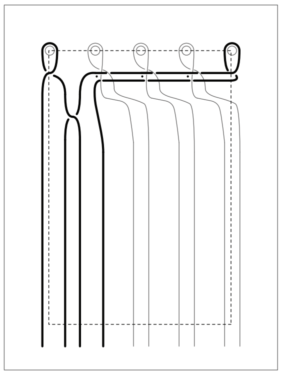
9A

⇒



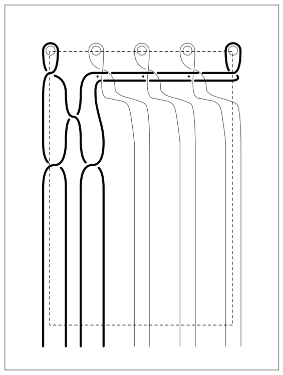
6

⇒



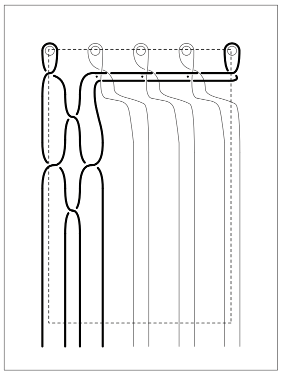
7

⇒



6

⇒



9B

⇒

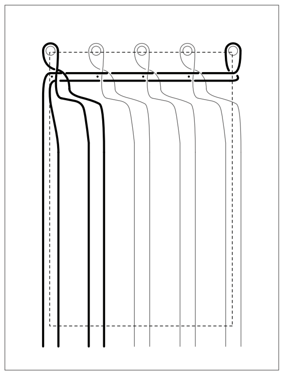
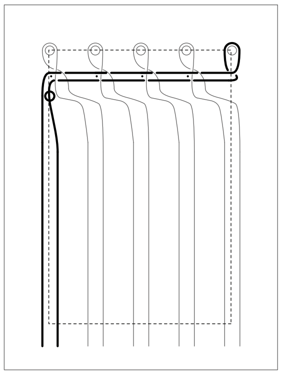
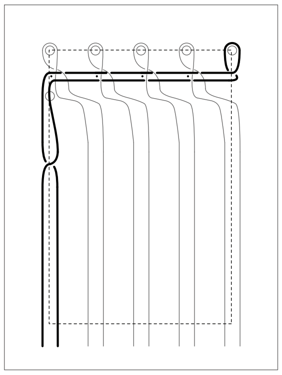


Figure 3.2 (continued): *High-resolution computation.*

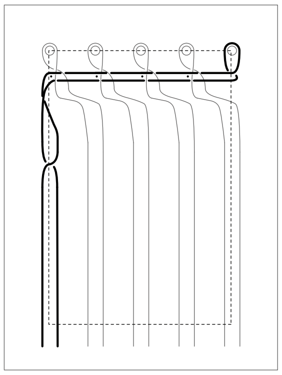
2 ⇒



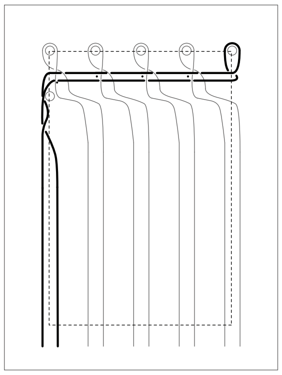
8 ⇒



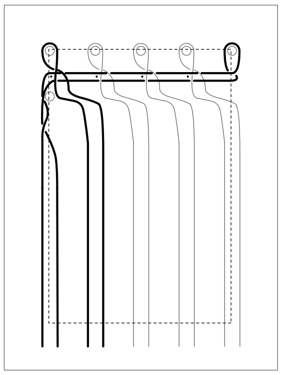
9B, 8 ⇒



9B ⇒



9A ⇒



6 ⇒

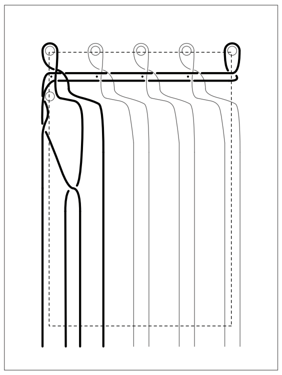
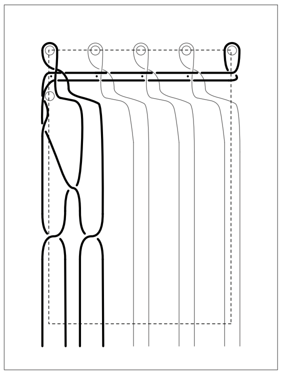
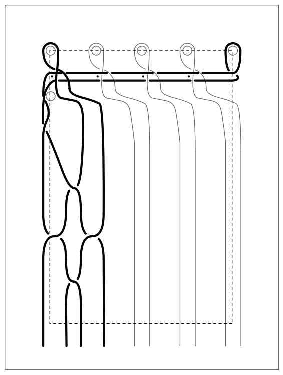


Figure 3.2 (continued): *High-resolution computation.*

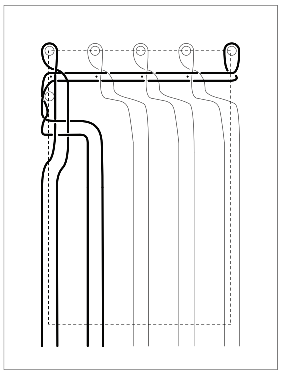
7
⇒



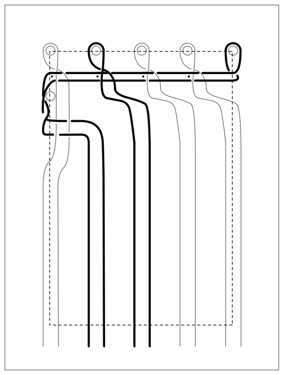
6
⇒



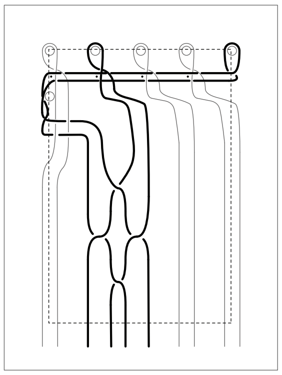
9B
⇒



9A
⇒



6, 7, 6
⇒



9B
⇒

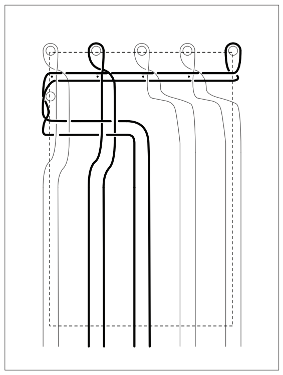
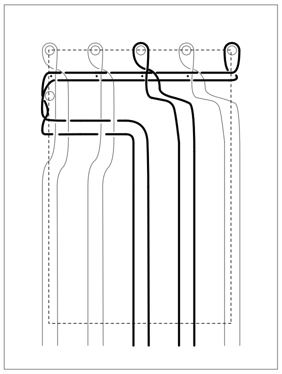


Figure 3.2 (continued): *High-resolution computation.*

9A

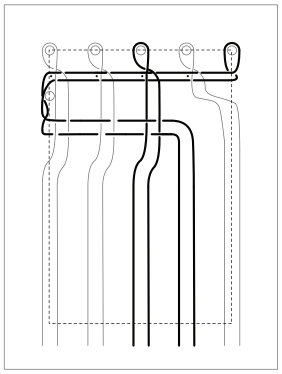
⇒



6, 7, 6,

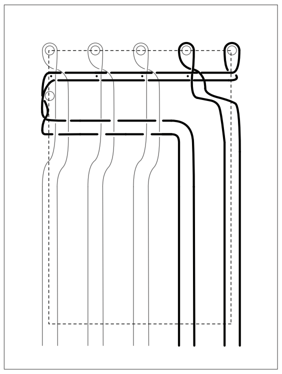
9B

⇒



9A

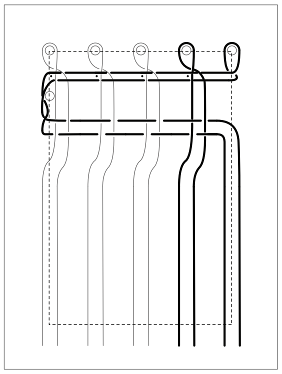
⇒



6, 7, 6,

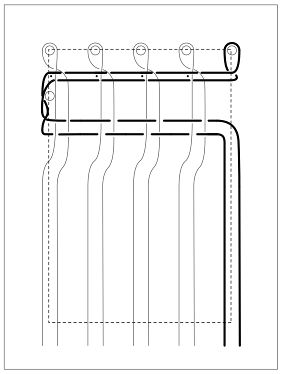
9B

⇒



9A

⇒



2, 8

⇒

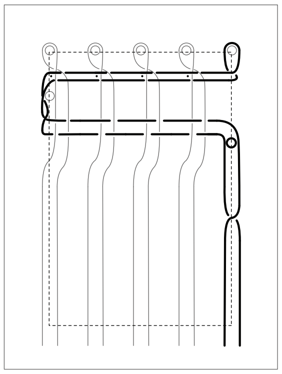
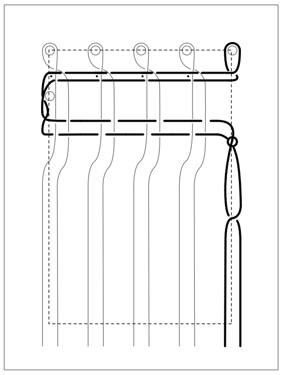
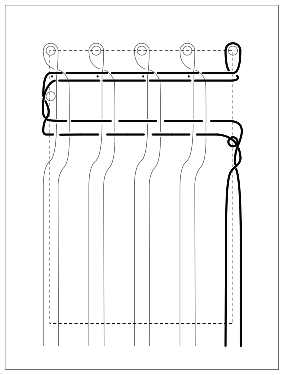


***Figure 3.2 (continued):** High-resolution computation.*

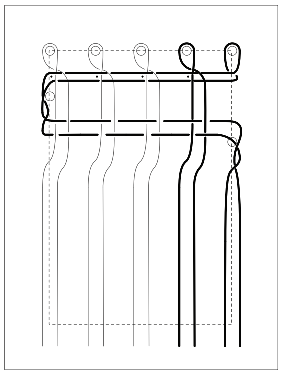
9B
⇒



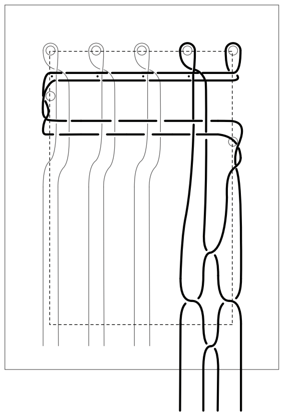
2
⇒



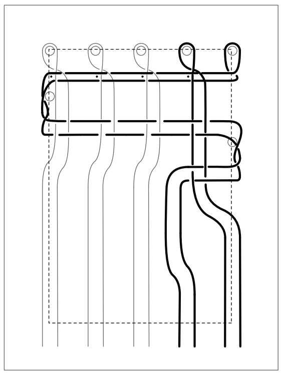
9A
⇒



6, 7, 6
⇒



9B
⇒



(9A, 6,
7, 6,
9B)^3
⇒

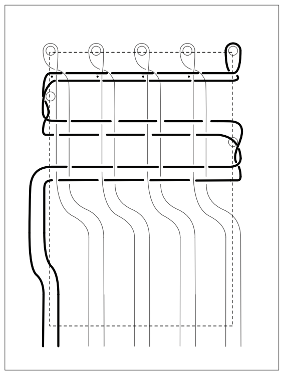
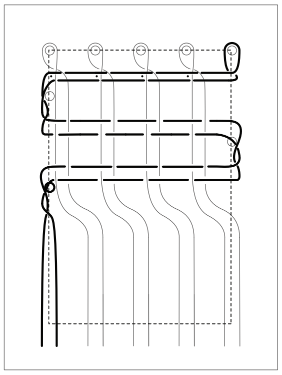
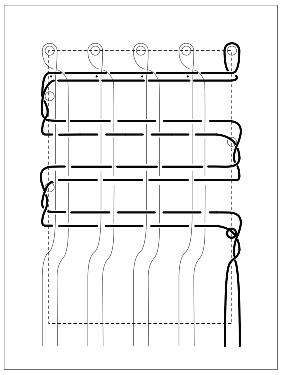


Figure 3.2 (continued): *High-resolution computation.*

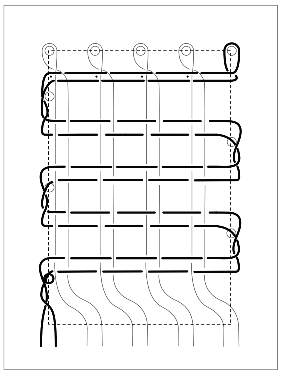
9A, 2,
8, 8, 9B
⇒



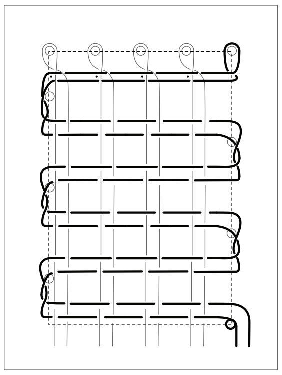
(9A, 6,
7, 6,
9B)^4,
9A, 2,
8, 8, 9B
⇒



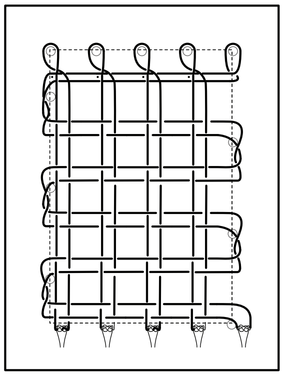
(9A, 6,
7, 6,
9B)^4,
9A, 2,
8, 8, 9B
⇒



(9A, 6,
7, 6,
9B)^4
⇒



10^5
⇒



11, 12
⇒

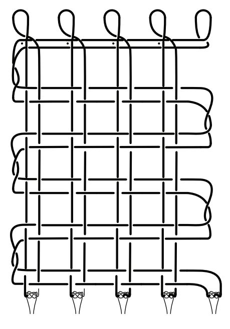


Figure 3.2 (continued): *High-resolution computation.*

High-Resolution Effort Calculation

In this whole-stitch lace computation, each step is shown either individually or as a group of steps once a pattern has been established. The high-resolution computation approximates the emergence of the lace as it is being made. The effort tabulation is a record of the effortful moments of making and is found in the Appendix.

It is important to note that while the high-resolution making rules are representative of the creation of lace in that a recognizable lace design emerges in the computation, they are nonetheless symbolic rpresentations of the making process and must be considered as such. In this way making grammars behave somewhat differently from shape grammars: the rules are implicit and the onus is on the maker to see the lace, interpret the abstracted drawn representation of a rule, and act. Making grammars discretize the continuous process of creation in the world. However, through this abstraction they render important concepts in creative processes clear: the order and context in which to do things.

The high-resolution making rules operate discretely and are readily correlated with a making grammar. However, other components of effort are more contextual. The next chapter will seek to broaden the ruleset to account for the context of the design and the expertise of the designer.

3.1 (b)   Directed Graph Representation

In the high-resolution making computation illustrated in section 3.1(a) above, discrete steps were first shown individually and then combined into larger processes for a more economical means of representation. For example, instead of separately representing a cross, then a twist, then another cross, and tightening the stitch, a completed *whole-stitch* may be shown (a sequence of Rule 6, Rule 7, Rule 6). While this visual shorthand might be slightly less didactic, its compact form is a powerful representation of the braid topology that emerges through the correct application of making rules. In the language of coding and compilers, it is a higher language representation. In the literature on lace patterns, stitches are graphically represented through symbols and described separately through words and actions. The lace maker "compiles" the verbal instructions and executes the minutia while reading the higher-level graphical representation.

The minutia of making bobbin lace may be represented mathematically. In bobbin lace, two pairs of threads labelled consecutively (*a, b, c, d*) begin an *interaction* when *b* crosses *c*.

The interaction ends when a thread in the original set crosses over or under a new thread $x \notin \{a, b, c, d\}$ (Irvine and Ruskey 2014). This definition of an interaction allows the representation of a lace ground as a *directed graph* in which "a set of objects (called *vertices* or *nodes*) are connected together with edges that are directed from one vertex to another"(Nykamp 2020). Each stitch is an interaction between four threads, and therefore can be represented as a vertex with two incoming edges (a pair of threads represented by a single line) and two outgoing edges (Irvine and Ruskey 2014).

Each vertex in the directed graph can be labeled with a function ζ, for example $\zeta: V \rightarrow \{C, T, p\}$. For torchon ground interactions $\zeta(v)=CTpCT$ for each $v \in V$ (2014). Each edge in the directed graph represents two threads, but the interaction at the vertex mapped by the ζ function will determine which threads propagate through the design and what shape will emerge. Note the difference between torchon ground and *Gravenmoer* ground when represented by conventional thread diagrams. By contrast, their corresponding directed graph representations are identical apart from their unique ζ function.

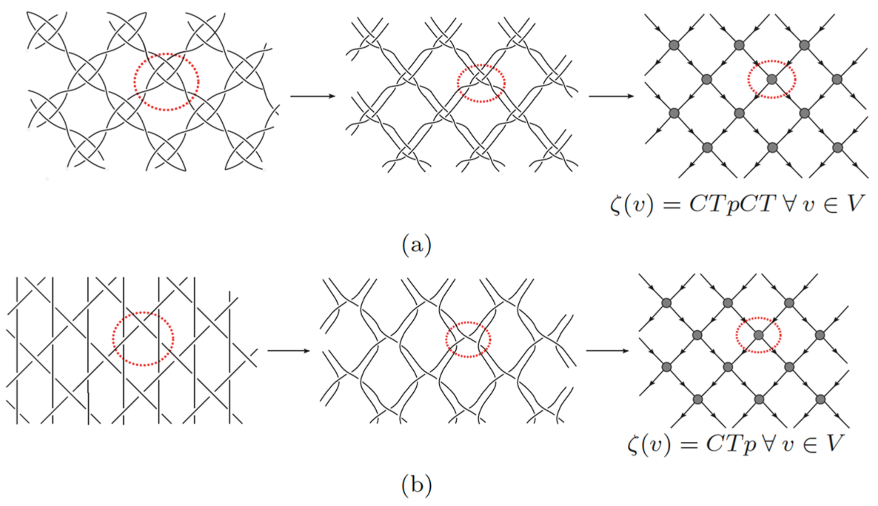

$\zeta(v) = CTpCT \; \forall \, v \in V$
(a)
$\zeta(v) = CTp \; \forall \, v \in V$
(b)

Figure 3.3: *Two example lace grounds: (a) torchon ground and (b) Gravenmoer ground. (Irvine and Ruskey 2014).*

An abstracted representation of lace permits different grounds to be described by the same directed graph. The topology of a directed graph representing lace ground, called the *ground embedding* in Irvine and Ruskey's paper, will be identical for different grounds, provided that the thread pairs travel between interactions in the same way. In their paper, Irvine and

Ruskey discovered over 100,000 previously unknown ground embedding representations, each without any restrictions for the ζ function (2014).

3.1 (c) Low-Resolution Making Grammar

Mathematical formalism aside, the abstract representation permits a more general approach to the formulation of making grammars for lace. Each interaction of threads is composed of the familiar cross, twist, and (frequently) pin actions but in differing frequency and sequence. Therefore, a more succinct making grammar absorbs the actions into the nodes and addresses more clearly the topology of the design. Given the multiplicity of interactions described at the node, this grammar representation is also more general. This abstraction, the low-resolution making grammar, identifies the topological conditions for the production of lace through a representative making grammar based on the ground embedding. It distinguishes between "worked nodes" and "unworked nodes" and gives the topological context in which an unworked node can change state. The "worked" state of the node indicates that four threads (*a, b, c, d*) have interacted according to the ζ function at the node of interaction.

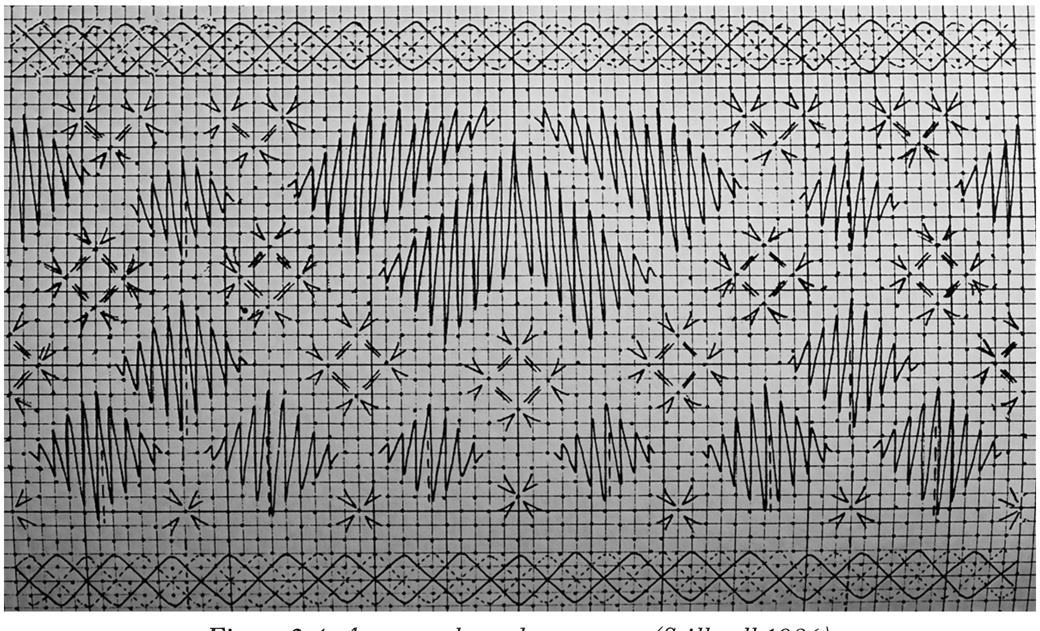


***Figure 3.4:** An example torchon pattern (Stillwell 1986).*

When designing bobbin lace, lacemakers use a grid of points to locate the interactions and plan the lace. The grid may be diagonal or square, and some lace makers have used logarithmic graphing paper, for example, to produce lace based on uneven spacing (Edkins

2010). In this thesis, I will use the forty-five degree diagonal square grid associated with Torchon lace as the basis for rules and computation.

In the low-resolution making grammar, the node states are identified as follows:

- Worked nodes: black dots
- Unworked nodes: grey dots
- Temporary node: red dots

Analogous to the "selection of the workspace" rule in the high-resolution grammars, the low-resolution grammar selects a subset of the ground embedding. Two points initialize the diagrid generation and a grey tone on the right-hand side of the rule distinguishes the area within the embedding so that lace edges can be identified in subsequent rules. The rule is applied repeatedly until the subset dimensions represent the number of rows and columns in the lace design.

Rule 1: Selection of ground embedding subset

Orienting the grid to the page, each vertex along the top row is marked with a symbol representing the winding and mounting of one or more pairs of bobbins at that location. After all pairs are hung for each node, initial braids are worked among the pairs to hold the threads together.

Rule 2: Winding, mounting, and initial braiding of pairs of bobbins at nodes along the top edge

Along the top edge, bobbins from adjacent nodes interact at the node below to change the state of that node from "unworked node" to "worked node." The bobbin symbol is removed from the node once a pair of bobbins at that node has been worked at another node.

Rule 3: Work node below two adjacent nodes on the initial row that contain only unworked bobbins

As nodes on the initial row are worked, nodes containing only unworked bobbins will become adjacent to nodes in which a pair of bobbins has been worked in another direction;

the nodes beneath these locations may be worked.

Rule 4: Work node below two adjacent nodes on the initial row only one of which contains only unworked bobbins

As the lace is made, adjacent nodes outside of the initial row will be worked; the unworked node below adjacent worked nodes may be worked.

Rule 5: Work node below two adjacent worked nodes

Torchon lace may begin from a diagonal edge. The most effective way to begin lace on a diagonal edge is to mount bobbins on temporary pins offset from the starting edge (Edkins 2016a).

Rule 6: Place temporary mounting pin offset from diagonal starting edge
Rule 7: Mount bobbins at temporary mounting pin
Rule 8: Work the node beneath the temporary node
Rule 9: Remove the temporary pin and pull the threads through to the edge

At the side edges of the lace, an unworked node on the edge of the lace and below a set of worked nodes containing a pair of adjacent nodes, one of which is located on the edge, and a third node that is worked beneath the pair, may be worked.

Rule 10: Work nodes at edges beneath a set of worked nodes

The rules for the making grammar outlined here are not determined from discrete instances of effort that contribute to production of lace. Rather, this low-resolution method operates more generally to generate valid torchon lace topologies. In comparison to the high-resolution rules, these rules are less concerned with documenting each step that the lacemaker takes. Instead, a completed interaction is represented by each node in a "worked" state. The low-resolution grammar ensures that the lace is made in a valid sequence of interactions as a node that is to be "worked" must first be "legal."

1
2
3
4
5
6
7
8
9
10

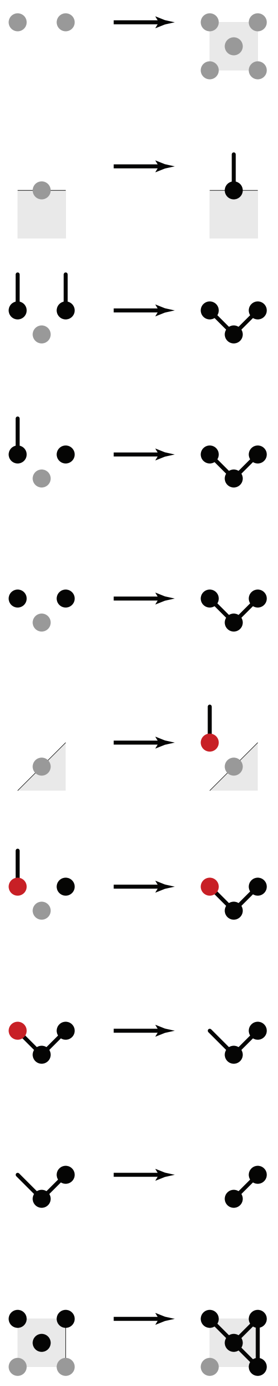


Figure 3.5: Low-resolution rules for torchon lace.

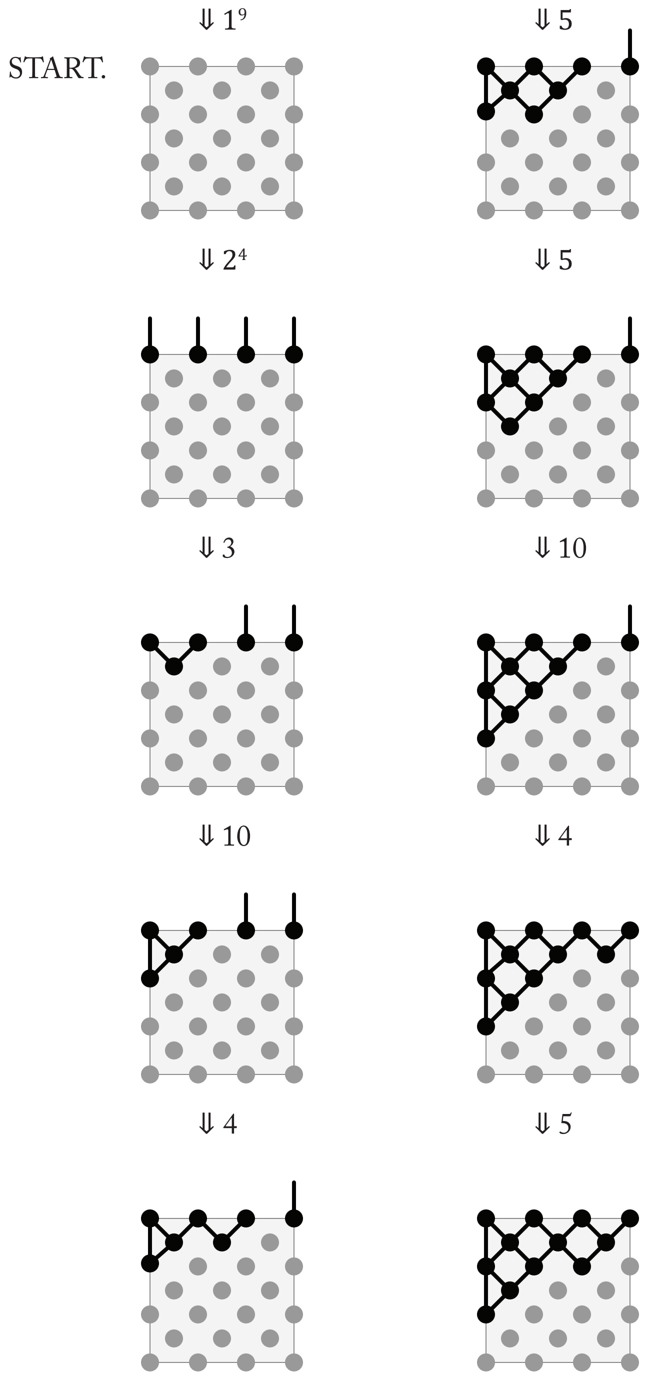


Figure 3.6: Low-resolution computation.

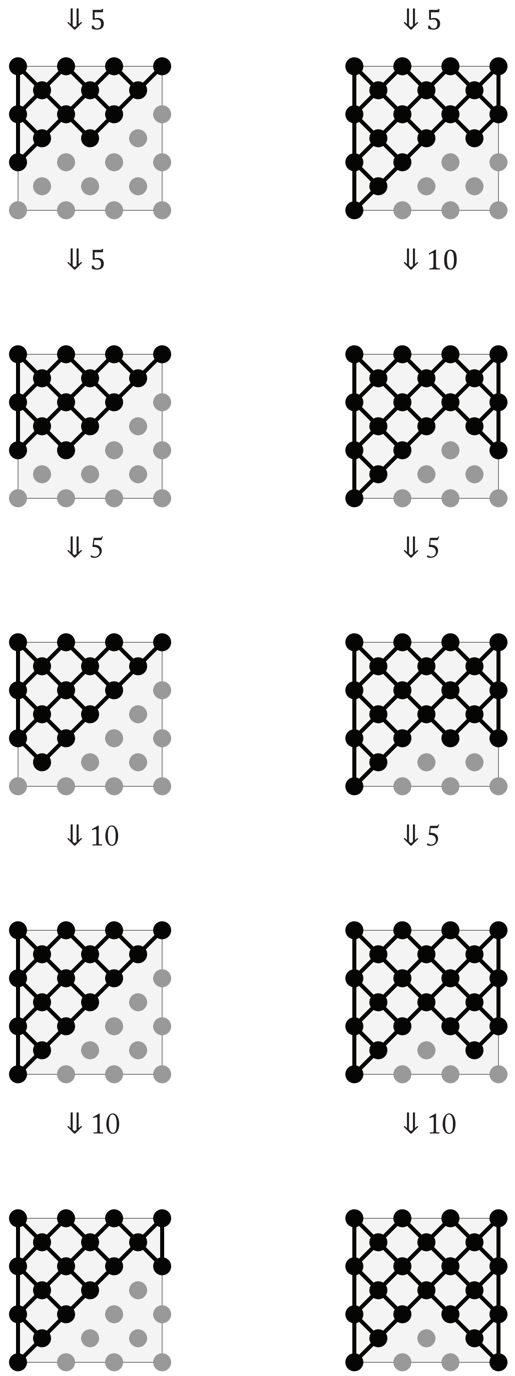


Figure 3.6 (continued): *Low-resolution computation.*

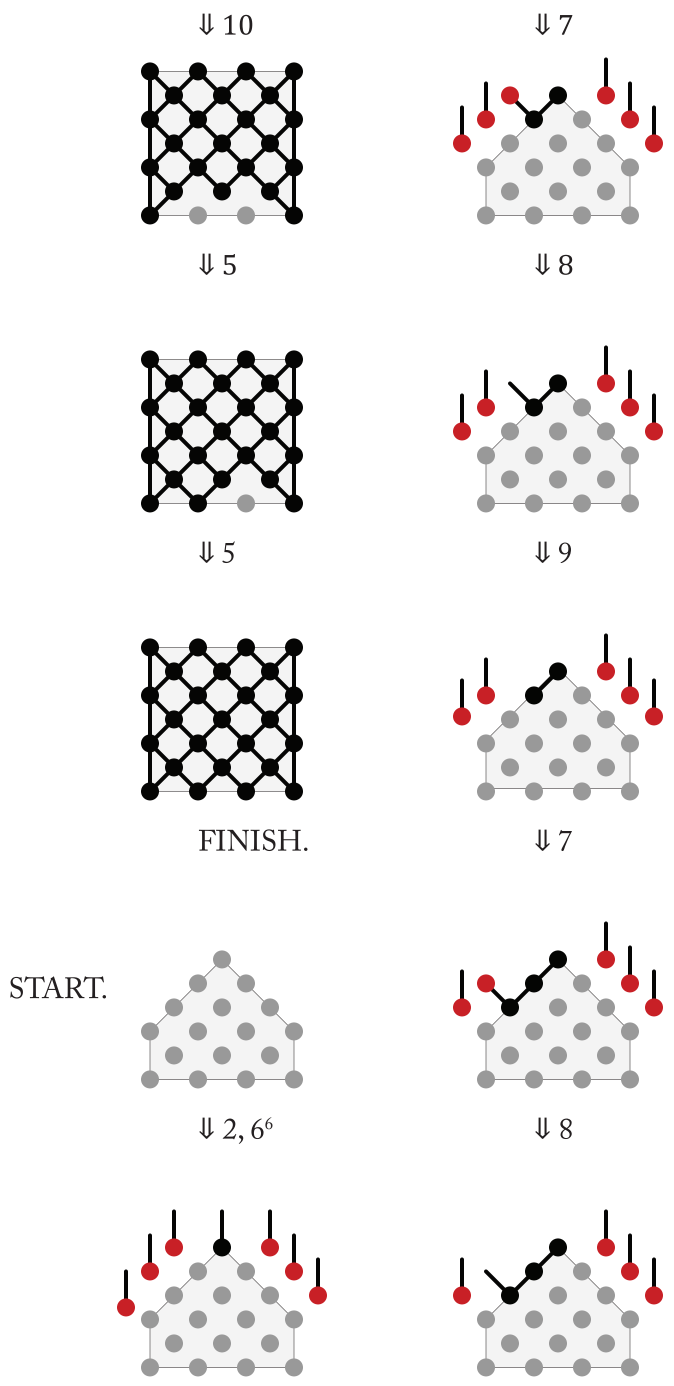


Figure 3.6 (continued): Low-resolution computation.

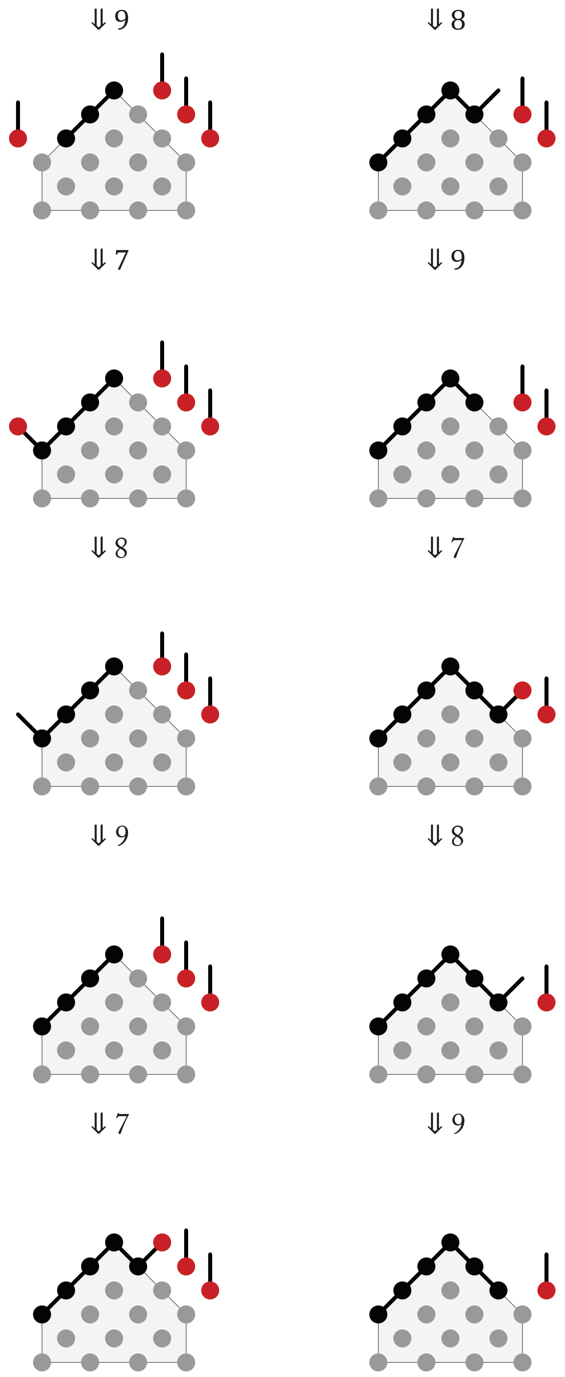


Figure 3.6 (continued): Low-resolution computation.

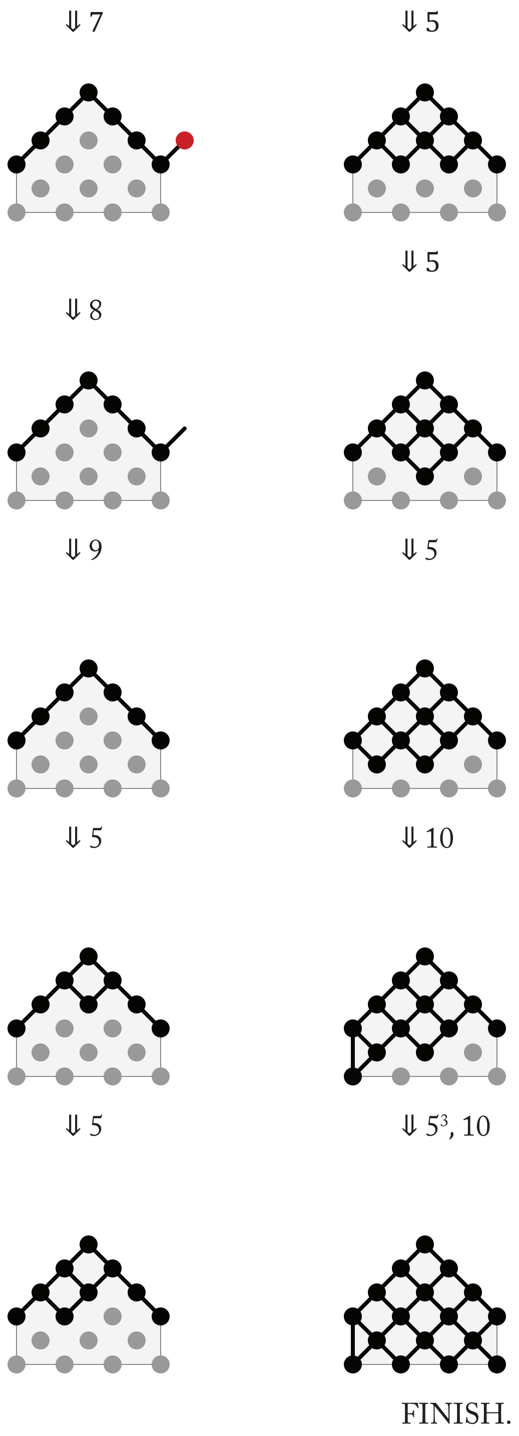


Figure 3.6 (continued): Low-resolution computation.

The application of the rules guarantees the constructability of the lace design, but the representation of the lace is abstracted. The effort associated with each step in the production of lace is also less explicit than in the high-resolution computation. Each step in the low-resolution computation corresponds to many components of effort: from the cross, twist, and pin actions to the cognitive effort of managing the pattern. Chapter 3.2 discusses the individual components of effort in greater detail.

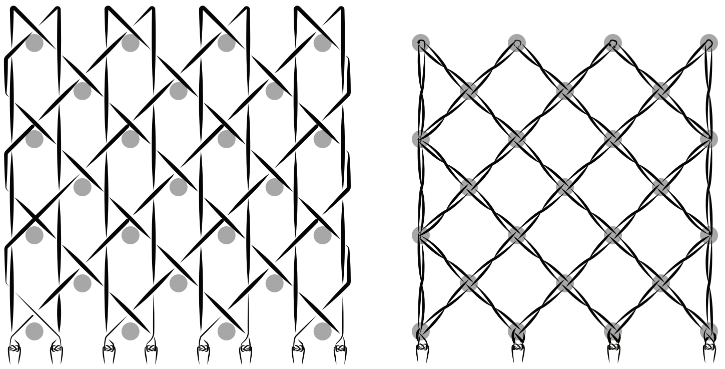


Figure 3.7: Multiple laces are possible with a shared low-resolution grammar.

3.2 Quantifying Embodied Effort

In the previous chapter, I identified the primary components of effort in the production of bobbin lace. I have yet to discuss those components as they relate and scale to one another and how they might be affected by the context in which they are deployed. This chapter introduces a method to quantify effort beyond the simple counting of discrete instances of it.

3.2 (a) Primary Sources

The descriptions of lacemaking in the literature and in hobbyist forums elucidate the relative difficulty of certain processes and introduces broad categories of effort, including that which is physical and that which is cognitive.

Bobbin Preparation: Winding and Mounting
Before the lacemaker can begin working the lace, the bobbins must be prepared as described in Section 2.4. Winding and mounting the bobbins require an exertion of physical effort, and Jo Edkins also notes that care must be taken to avoid extra effort: "I once dropped my pillow at this stage. Half the threads bounced off the pins, and got entangled in other pins, and it was an appalling mess!" (2016c). Edkins offers an alternative practice:

> *Once you have hung a few pairs, start making the lace with them, and gradually hang more pairs, and work them in as you go. You need a little knowledge of lace to know which pairs to hang first, and how to work them, so perhaps you will not be able to do this right at the start. But once you can, I do recommend it, especially once the number of pairs in a pattern increases.* (2016c)

The effort required to wind and mount the bobbins needed to make a pattern includes a physical effort of each and a cognitive effort of doing so carefully to avoid entangling threads. The effort will scale with the quantity of bobbins and can be decreased through the application of expertise.

Cross & Twist Actions
The cross and twist actions are the fundamental components of physical effort in lacemaking. These actions determine the topology of the lace as bobbins move over and under their neighbors. Edkins' description below also identifies the cognitive effort related to identifying the working group of bobbins for the current stitch:

> *A stitch is made of two pairs (four bobbins). When making a stitch, move the bobbins on the pillow slightly to left or right, so the four that you are working have some space on either side (make sure that you do not disturb their order!) That will give you the room to lift one bobbin over another and put it down in the required place. Beginner patterns do not have many bobbins, so this is not too much of a problem, but very wide patterns have* lots *of bobbins and this means that finding room to make a stitch can be challenging! It is possible to lift one bobbin over another, and shove the underneath bobbin across with a finger, to allow room for the lifted bobbin. But don't worry about that yet as you will not need to do it as a beginner.* (2016d)

Cognitive and motor effort will increase with an increase in cross and twist actions and number of bobbins in the design.

Pin Action

The pinning action is a straightforward application of physical effort. Edkins remarks: "You will be glad to know that there is only one way to put in a pin! Do not push the pin in up to its head, because it has to hold the threads around it. Only push it in about a quarter of the way - enough to keep it in place."

There is, however, cognitive effort required to recognize where to place the pin:

> *When you put in a pin, it will be between two bobbins (belonging to different pairs). Find where that is from the description of the design, and slide the pin between the threads until the point of the pin is over where it should go. Remember, you have already pricked all the pinholes, so you can almost feel exactly where this is. Slide the pin into the hole and push it in far enough (but not too far). The pin will need to be fairly upright. You are going to put in a lot of pins, and if they do not go in straight, there will not be room. It can help to tilt the pins at the edge slightly away from the lace. The threads tend to tug them inwards, and that tilt helps to counteract that. But do not make the tilt too great, even so.* (2016d)

With expertise, the lacemaker may choose not to use pins:

> *Some lace makers work ground without using any pins at all! They tighten the work before starting the ground, and after finishing. This means that you need a pattern where you can do this. It's obviously a lot quicker to work as putting in pins takes time.* (2016d)

Pinning requires cognitive effort and physical effort in order to locate the proper place for the pin and then to put the pin in place. With expertise, the lacemaker may forgo pinning actions and thereby decrease the related exertions of effort.

Interaction Complexity

In the introduction to *The Book of Bobbin Lace Stitches*, the authors indicate that in organizing the publication "the stitches are arranged according to their degree of difficulty with the simpler stitches appearing early on, progressing through to the more complex stitches. Towards the end are 'Spiders', "Buds', 'Shells', 'Toiles' and 'Peas'."(Cook and Stott 2002) The first stitch in the book is *Half Stitch Ground* and, in the mathematical notation introduced in Section 3.1(b), may be described as $\zeta(v)=CT$. Aside from twisting one pair, this is the simplest operation in bobbin lace. A *whole stitch* builds on the half stitch: $\zeta(v)=CTC$ and is often referred to as *cloth stitch*. The same stitches appear first in *Gründe Mit System* (Ulrich 2009) and are foundational to other stitches where knowledge of a half-stitch (h.s.) and whole-stitch (w.s.) is assumed.

Increasing in complexity—and difficulty—more maneuvers may be added to the interactions. In the ground *Flemish Filling*, for example, $\zeta(v)=CTCTTTpCTCTTT$. While this notation may help us count single instances of effort as in the high-resolution computation, it should be noted that the ζ notation is not the means by which instructions are conveyed in the hobbyist literature. Rather, the same Flemish Filling is described by the following written instructions (Cook and Stott 2002):

w.s., tw. 3
pin
w.s., tw. 3

A still more difficult ground in *The Book of Bobbin Lace Stitches* is listed, for example, on page 55: *Triangular Ground 1*. The instructions follow (2002):

w.s. throughout
pin to support top and bottom of lozenge
w.s., pin, w.s. at apex of triangle
tw. 1 between triangles

This ground consists of several whole stitches and pinning actions. The ground requires contextual awareness as each node must be differentiated in order to deploy the appropriate maneuver in the correct location. Triangular Ground 1 may be described as:

$\zeta(v_1) = CTCp$
$\zeta(v_2) = CTC$
$\zeta(v_3) = pCTC$

where each node v_1, v_2, v_3 is repeated more than one time in a single unit of ground.

The sequence of stitches that comprise interactions may vary in length and complexity and require differing contributions of physical and cognitive effort. As an increased amount of interactions are introduced into a design, the lacemaker will also need to exert cognitive effort to recall which sequence to deploy during production. It will be more effortful for a lacemaker to work with multiple, complex interactions than it will be to work with fewer or simpler ones.

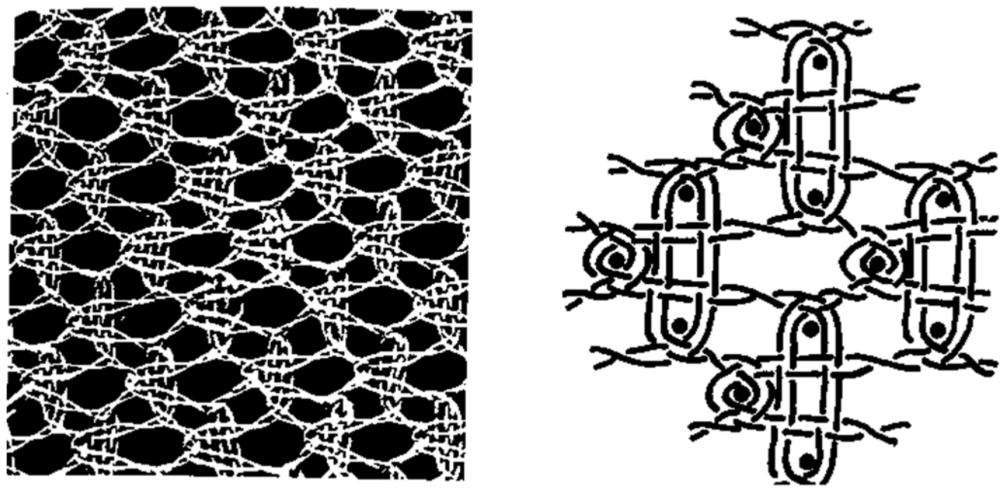


Figure 3.8: Triangular ground (Cook and Stott, 2002).

Thread Tightening

A component of lacemaking that requires both physical and cognitive effort is the tightening of the lace as it is being produced. Edkins remarks, "Tightening thread while making lace is very important. If you don't tighten the threads enough, you can be left with little loops in the finished lace, which looks ugly" (2014).

Edkins outlines several guidelines for when and how to tighten the lace:

- Tighten after pinning
- Tighten when threads are loose above a pin
- Pull bobbins slightly while crossing or twisting
- Tighten by tapping on top of the bobbins
- Tighten by putting a hand over several and moving it downwards

- Tightening is easier when the thread lengths are the same
- Use a pin at "misbehaving" stitches to probe the stitch and pull out slack

Edkins also offers a warning: "Tugging too hard pulls the pins out, which leads to chaos, so be careful!" (2008a). This point emphasizes the importance of tightening correctly in order to avoid disturbing the lace.

Tightening is also contextual. Edkins notes:

> *Sometimes the threads change direction quite abruptly…Sometimes this change of direction makes a thread go slack or even make a little loop. You can also get this with more complicated stitches, or if the thread is a little rough, or has a knot in it, or just because it is in a bad mood! These little loops or looseness will show in the final lace. Tug each bobbin in turn until you find the right one, then pull gently until the loop disappears.* (2014)

Maintaining the lace design requires cognitive effort in identifying where and when to tighten, and the physical effort associate with how it is done. It is also related to the lacemaker's expertise in anticipating moments where tightening will be necessary or difficult due to the circumstances or context of the emerging piece.

Lace Density

The manual dexterity required in making lace increases as the spacing between interactions decreases. The bobbins are a fixed size, but as the threads move closer together, there is an increased possibility of confusion or tangling. In her description of working lace designed from an experimental logarithmic grid, Edkins notes:

> *I printed off the pattern…a little too small, which meant that the narrow part of the grid was rather tricky to work, and I kept missing holes. I had to undo the strips several times to get it right! But I think that if I had made it bigger, it would have been a lot easier.* (2010)

An increasing density of stitches not only increases the count of actions required for the same area of lace, it also makes working the stitches more difficult.

Working Order

As discussed in the "Cross & Twist Actions" subsection above, the lacemaker must exert cognitive effort to identify the correct pairs of bobbins when working the lace. When adjacent interactions on the same row are worked, there is no shared pair between the selected pairs. By contrast, working the lace in diagonal rows will lead to a shared pair of bobbins between sets and will be easier to identify. Edkin's notes:

> *When working grounds, you need to figure out which two pairs of bobbins make the next stitch…The best way to work ground is in diagonal rows. Whether left or right does not matter (and you can switch from one to the other). You may need to work a stitch or so to get the diagonal started, but once that is done, you can work one pair of bobbins right across the others. This helps to guard against making the stitch with the wrong pairs.* (2016b)

As the lace is worked, different nodes will become "legal" when there are two pairs of incoming threads from nodes that have already been worked. The order in which the legal nodes are worked will affect the amount of effort required to do so.

Completion: Removing Pins and Bobbins

When the lace is worked to the final row or point, the bobbins are cut from the threads and removed. The method of finishing the lace with the least effort is to then simply remove all of the pins. Edkins notes:

> *The simplest way to finish lace is just to cut off the bobbins, unpin the lace, and there you are! …However, a little bit might get unraveled, and you spent time working that bit, and anyway it looks untidy, so you might want a better way. One way is to tie off the threads in knots.* (2008b)

Several knots may be used, including a square or granny knot between each pair of threads:

> *First unwind the bobbins for a bit, then cut them off leaving long threads. Then tie knots between threads, or pairs of threads, or larger groups of threads. There is a choice of knots that you can use. If you want to make the knots to be as inconspicuous as possible, then tie each pair of threads as a reef knot…It doesn't even matter if you tie a granny knot by mistake! Then trim the threads close to the knot.* (2008b)

Alternatively, an overhand knot may secure multiple pairs together:

> *Another technique is to make a feature of the finish by creating a fringe. Tie the threads in an overhand knot…and when they have all been tied off, trim them to the same length, short or long as you please. An advantage of this is that you can tie more than one pair at a time. I often tie two pairs, round their final pin, in a horizontal line, which I think makes a neat finish, although it's quite possible to have a slanting or pointed end.* (2008b)

Completing lace requires the cognitive and physical efforts to remove the bobbins and pins from the workspace and decide on a finishing technique. The completion effort is proportional to the number of bobbins and pins in the design and is subject to the expertise of the lace maker.

3.2 (b) Formalizing Embodied Effort

An inspection of primary sources combined with the author's experience in making bobbin lace support a mathematical representation of effort.

Categories of Effort

The total effort exertion required to make bobbin lace includes two broad sub-categories: the physical—or motor—effort exerted to physically change the state of materials at play and the cognitive effort exerted to perceive the current state of materials and recall what motor effort to deploy to advance the making process.

It is difficult to quantitatively untangle motor movement from cognition, however, and therefore in this thesis ε_{motor} will be defined as effort exerted towards a change in the physical state of materials while $\varepsilon_{cognitive}$ will be defined as effort that assess the materials and pattern without physical exertion.

With these qualifications, the first effort equation for bobbin lace is:

$$\varepsilon_{total} = \varepsilon_{cognitive} + \varepsilon_{motor}$$

where cognitive effort $\varepsilon_{cognitive}$ includes $\varepsilon_{perceiving}$ effort to perceive the work and $\varepsilon_{recalling}$ effort to remember what to do next:

$$\varepsilon_{cognitive} = \varepsilon_{perceiving} + \varepsilon_{recalling}.$$

ε_{motor} motor is further categorized into the three phases of lacemaking outlined in Section 2.4:

$$\varepsilon_{motor} = \varepsilon_{preparing} + \varepsilon_{working} + \varepsilon_{finishing}.$$

Each category of effort receives contributions from individual instances of effort exerted in the context of the design and making process.

Components of Effort

Each discrete component of effort (cross, twist, pin, for example) may be counted as each instance occurs, but may also be subject to the context in which it is deployed. As made evident in Jo Edkins' detailed descriptions of lacemaking, each stage of lacemaking includes a contribution of motor effort and cognitive effort.

In enumerating the components of effort, any cognitive effort that cannot be readily decoupled from its motor counterpoint will be included as motor effort. For example, while winding bobbins clearly requires cognitive effort, the cognition only supports the motor effort and is not otherwise contextual; only the bobbin held is being wound and nothing else need be considered. I am excluding cases of "thinking about" winding bobbins or other instances where cognition does not lead directly to a physical change of state of the materials.

However, in the cases of identifying the two pairs of bobbins participating in an interaction and identifying where threads need tightening, the cognitive effort is decoupled from a physical change. The identification processes require perceptual effort that is subject to the entire context of the lace, specifically requiring more effort when the lace is denser and distinction is harder to perceive, and must be exerted before the appropriate motor effort can be deployed. Similarly, the cognitive effort required to recall the proper steps to create the pattern is contextual, does not change the material directly, and is a prerequisite to exerting physical effort to do so. These cognitive components of effort are thus treated separately from motor components:

$e_{perceive.pairs}$

$e_{perceive.tight}$

e_{recall}

The “dot” notation in the component subscript is used to differentiate related instances of effort. The prefixes *perceive.* and *motor.* differentiate perceptual from motor components of a related effort while the suffixes *.prep* and *.finish* differentiate related motor efforts in separate phases of lacemaking.

Motor effort is contributed to from instances of effort in three phases of lacemaking:

1. Preparing
    - Winding bobbins
    - Mounting bobbins

2. Working
    - Crossing threads
    - Twisting threads
    - Pinning
    - Tightening threads

3. Finishing
    - Removing bobbins
    - Unpinning

Winding and mounting bobbins will typically happen in tandem; it is assumed that there are no pre-wound bobbins. Therefore, the effort for winding and mounting bobbins may be combined more generally as the effort required to prepare the bobbins:

$e_{bobbin.prep}$

The effort to cross, twist, pin, and tighten loose threads are most effectively counted as separate instances; the difficulty of the lace will scale proportionally to total counts of these components across all interactions:

e_{cross}

e_{twist}

e_{pin}

$e_{motor.tight}$

When the lace is completed to the last row, the pins are removed and the bobbins are cut from the threads, which are then tied with finishing knots:

e_{unpin}

$e_{bobbin.finish}$

Embodied Effort Equations

Towards a general mathematical description of lacemaking effort, lace parameters are identified. Each node of the lace, s_i, is the i^{th} braiding sequence in a total count of n interactions in the design. Each column of the lace, l_i, is the i^{th} column in a total of k columns. The design consists of a set of possible interaction types described by the ζ function at each node; the total count of these unique sequences is S.

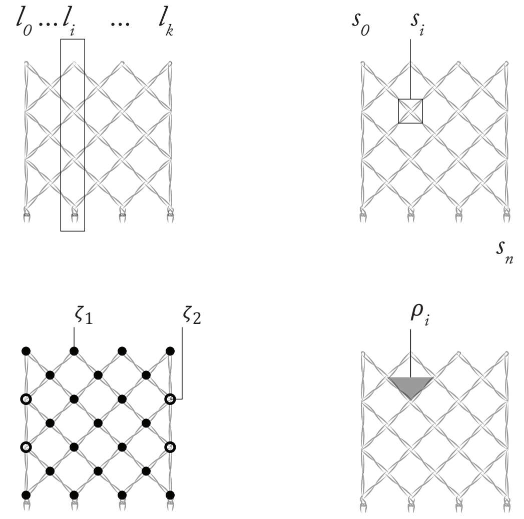


Figure 3.9: Variables used for effort calculations.

The density at each sequence location, $\rho(s_i)$ is proportional to the triangular area between s_i and the two nodes attached to the incoming arcs on the representative directed graph.

This triangle is used to calculate density because the threads will have already been worked above this point, are not worked below this point, and is a good approximation of the work area the lace maker will need to identify during production.

To make a particular design, one or more pairs of bobbins are mounted at each node on the top of each column. The function $b(l_i)$ describes the number of pairs of bobbins required for each of those nodes.

The functions $C(s_i)$, $T(s_i)$, and $p(s_i)$ return, respectively, the counts of cross, twist, and pin at each stitch in the design (similar to the ζ function, but irrespective of order and consistent with the stitch indexing used in this model). The function $m(s_i)$ returns a Boolean value reflecting whether tightening maintenance is needed the stitch location or not. The function $\delta(s_i)$ returns a value corresponding to how the stitch was worked compared to the previous stitch. If it was worked so that it shared a pair of bobbins, the value is 1; if the bobbins are not shared, the value is 2. The function $\alpha(s_i)$ returns the length of the action sequence to be recalled at each interaction.

Noting that the effort to identify where tightening is needed and to identify the active pairs of bobbins is proportional to the density of the lace, the perceptual effort equation may be written:

$$\varepsilon_{perceiving} = \sum_{i=1}^{n} \rho(s_i)[e_{perceive.tight} + \delta(s_i)e_{perceive.pairs}]$$

Recall effort is a function of how many unique sequences S there are to remember in the pattern. S has a lower bound of 2, indicating that one possible interaction is "do nothing." The term:

$$\frac{S-1}{S}$$

returns 0.5 when there is one actionable sequence (in addition to "do nothing") and approaches 1 as more interaction types are added to the pattern. The recalling effort equation is written:

$$\varepsilon_{recalling} = \frac{(S-1)}{S}\sum_{i=1}^{n} \alpha(s_i)[e_{recall}]$$

The effort required in preparing the lace is proportional to the total count of bobbins that are wound and mounted at the top of each column in the design:

$$\varepsilon_{preparing} = \sum_{i=1}^{k} b(l_i) e_{bobbin.prep}$$

The effort from cross, twist, and pin actions are grouped together in the making phase:

$$\varepsilon_{working} = \sum_{i=1}^{n} (C(s_i)e_{cross} + T(s_i)e_{twist} + p(s_i)e_{pin} + m(s_i)e_{tight})$$

Finishing the lace requires the effort of removing the pins at all interactions that included them and cutting the bobbins off and tying the finishing knots at the bottom of all columns:

$$\varepsilon_{finishing} = \sum_{i=1}^{k} b(l_i) e_{bobbin.finish} + \sum_{i=1}^{n} p(s_i) e_{unpin}$$

Making bobbin lace is subject to the expertise of the lace maker. As Jo Edkins' descriptions in Section 3.2(a) indicate, an expert lace maker will have access to time-saving routines and will have built muscle memory and perceptual acuity to decrease the effort required in each step. It is also possible that this expertise might be distributed unevenly so that a lace maker might be expert in winding the bobbins, for example, but not as much of an expert in tightening maintenance, for example. Therefore, an *expertise coefficient* r_x, where the subscript x refers to the effort category and the "expertise level" is a scale of 1-10 indicating the proficiency of the lacemaker in a particular category.

$$r_x = \frac{1}{\text{expertise in category} x}$$

Lacemaking expertise is accounted for in a final set of equations:

$$\varepsilon_{perceiving} = r_p \big(\sum_{i=1}^{n} \rho(s_i) [e_{perceive.tight} + \delta(s_i) e_{perceive.pairs}] \big)$$

$$\varepsilon_{recalling} = r_r \left(\frac{(S-1)}{S} \sum_{i=1}^{n} \alpha(s_i) [e_{recall}] \right)$$

$$\varepsilon_{preparing} = r_a \big(\sum_{i=1}^{k} b(l_i) e_{bobbin.prep} \big)$$

$$\varepsilon_{working} = r_w \big(\sum_{i=1}^{n} (C(s_i)e_{cross} + T(s_i)e_{twist} + p(s_i)e_{pin} + m(s_i)e_{tight}) \big)$$

$$\varepsilon_{finishing} = r_f\Big(\sum_{i=1}^{k} b(l_i)e_{bobbin.finish} + \sum_{i=1}^{n} p(s_i)e_{unpin}\Big)$$

3.2 (c) Quantifying Effort Components

While the counts of cross, twist, and pin actions and pairs of bobbins required are explicit for a particular design, the values for effort components must be approximated. The primary interest of this thesis is comparing the effort-costs of designs, and therefore a unit-less metric of comparison rather than an attempt to explicitly quantify a value of effort is preferred. For example, the twist action requires moving two bobbins instead of one in the cross action. The score for twist is then given a score twice that of cross.

Analogous to the energy stored in a smartphone battery, which is drained by computational tasks and also discharges over time, effort is here defined as a function of time and processes requiring energy expenditure. Processes that might require little energy expenditure but take a long time are therefore given higher effort scores than they would receive if only energy were counted. Winding bobbins, for example, is not particularly energy intensive, but it takes a lot of time. Bobbin preparation, therefore, receives an effort score higher than either cross or twist.

The values are flexible and may be tuned as the model is developed. Table 3.1 provides an initial list of relative effort scores for each component.

3.3 Effort Calculations of Lace Designs

With a model of embodied effort in bobbin lacemaking in place, effort-scores are calculated for a variety of lace designs. First, I will introduce a method of annotating and scoring existing lace designs. In the subsequent sections, I will automatically generate lace designs and calculate their effort scores during their emergence, or *on-the-fly*.

3.3 (a) Manual Calculations of Existing Lace

Lace patterns locate the pin holes for the *pricking* and use symbolic ground descriptions to instruct the lacemaker on how to work the lace. A hybrid notation that combines the nodes and edges of the directed graph representation with the spatial accuracy of the *pricking* is developed here to account for all instances of effort components in the design.

The two lace patterns below are analyzed to determine all of the interactions in the lace. Each unique interaction is assigned a symbol which is then superimposed on the diagram. This functions as a visual shorthand for the ζ function. Twisting between interactions is notated using a stroke across the line indicating a pair of threads per the convention in lace diagrams. More strokes may be added in parallel to denote the number of twists the pair should receive.

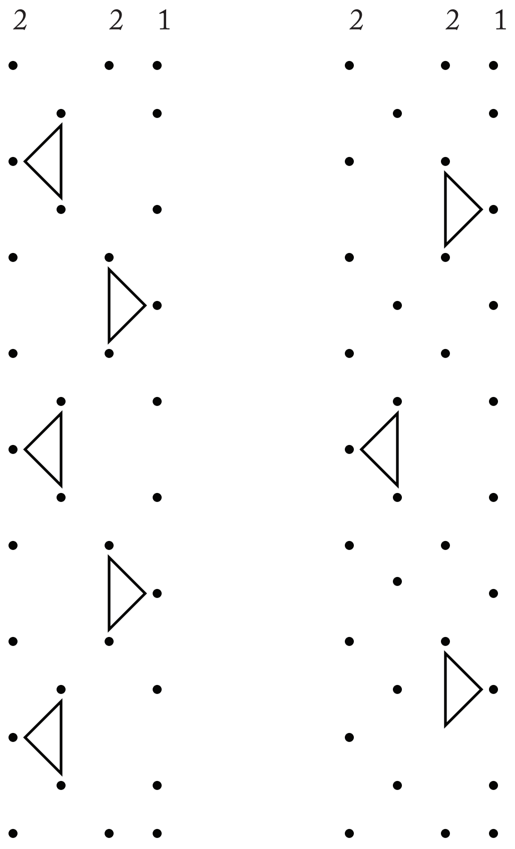
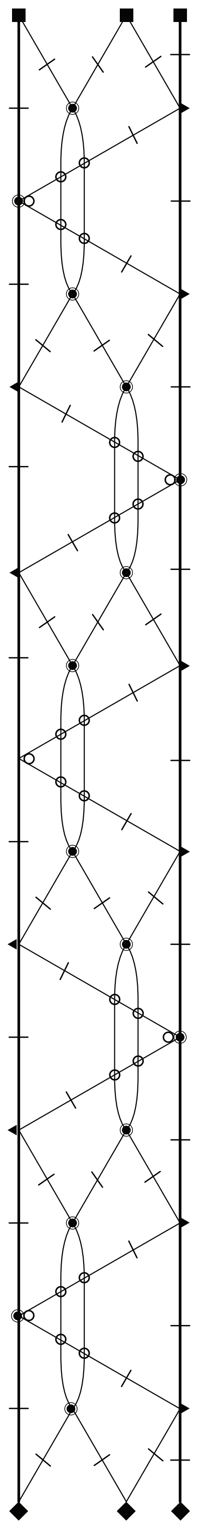


Figure 3.10: *Two bobbin lace patterns. Redrawn from Jo Edkins (2016).*



s(■) = 3

b_m(■) = 5

s(◉) = 15

α(◉) = 4

(p-CTC | CTC-p)

s(○) = 25

α(○) = 3

(CTC)

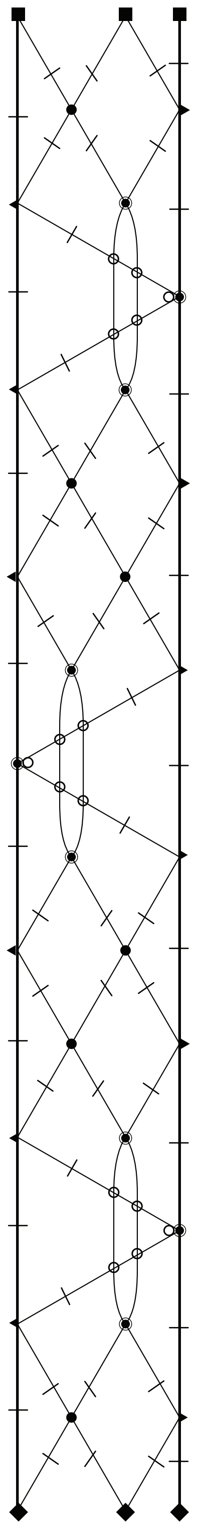
s(‣, ◂) = 10

α(‣, ◂) = 9

(CTCT-Tr-p-CTC)

s(┼) = 45

b_r(◆) = 5



s(■) = 3

b_m(■) = 5

s(●) = 6

α(●) = 5

(CT-p-CT)

s(◉) = 9

α(◉) = 4

s(○) = 15

α(○) = 3

s(‣, ◂) = 12

α(‣, ◂) = 9

s(┼) = 53

b_r(◆) = 5

Figure 3.11: *Manual effort analysis for two patterns: A (L) and B (R).*

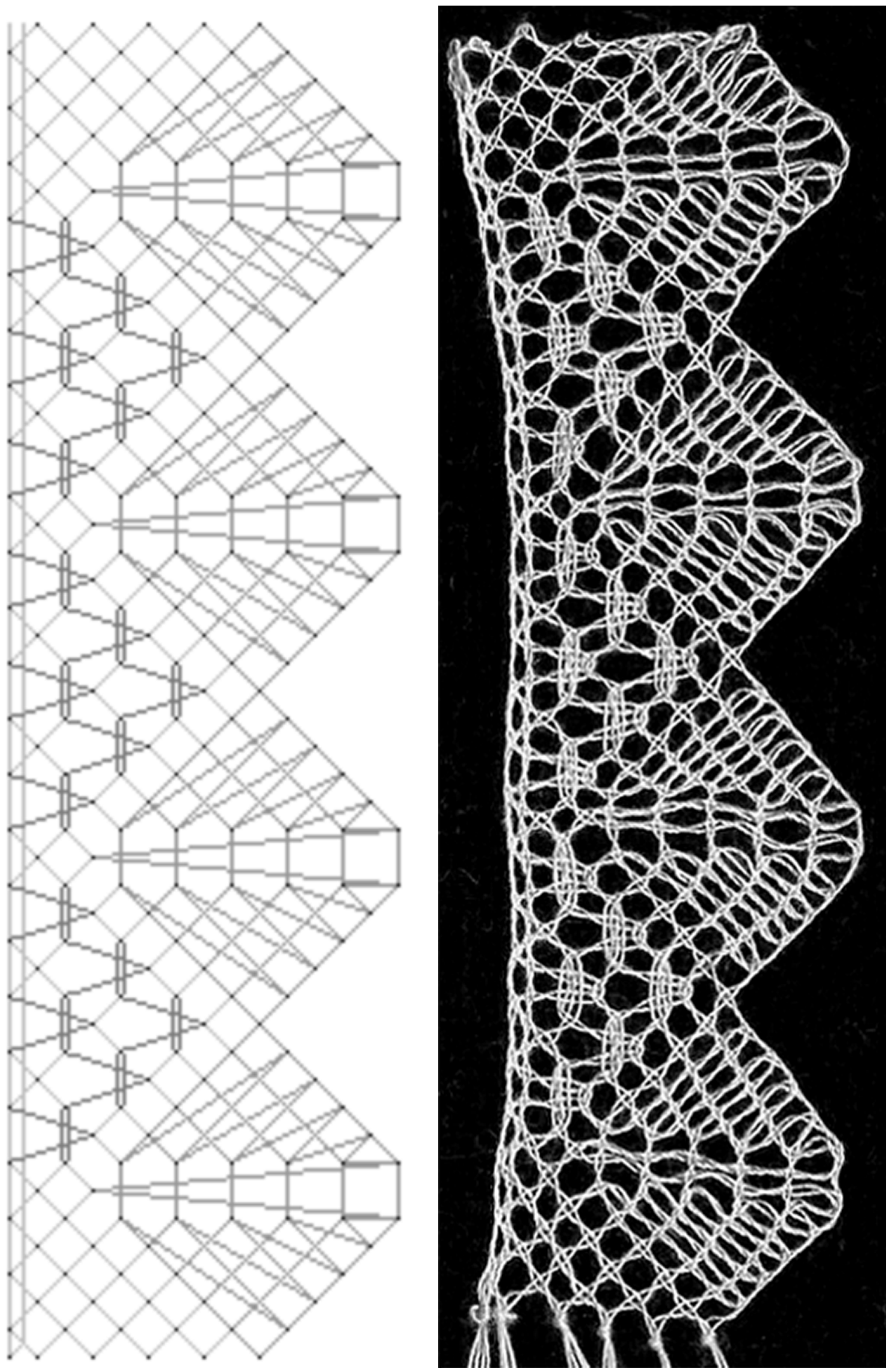


Figure 3.12: *A computer-drawn pattern (L) and a completed lace (R). Jo Edkins (2016).*

	C	T	p	α	ρ	m	b_m	b_r
Pattern A								
s(■)	0	0	1	1	1	1	5	0
s(◉)	2	1	1	4	1	1	0	0
s(○)	2	1	0	3	2	1	0	0
s(▸,◂)	4	4	1	9	1	1	0	0
s(┼)	0	1	0	1	1	1	0	0
s(◆)	0	0	0	0	0	1	0	5
Pattern B								
s(■)	0	0	1	1	1	1	5	0
s(●)	2	2	1	5	1	1	0	0
s(◉)	2	1	1	4	1	1	0	0
s(○)	2	1	0	3	2	1	0	0
s(▸,◂)	4	4	1	9	1	1	0	0
s(┼)	0	1	0	1	1	1	0	0
s(◆)	1	0	0	0	0	1	0	5

$e_{\text{bobbin.prep}}$	10
e_{cross}	2
e_{twist}	4
e_{pin}	2
e_{unpin}	1
$e_{\text{perceive.pairs}}$	5
$e_{\text{perceive.tight}}$	6
$e_{\text{bobbin.finish}}$	5
e_{tight}	4
e_{recall}	2
$(S-1)/S$	0.75

Table 3.1: *Complete effort-tabulation for a lace worked in diagonal rows.*

$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{subtotal}}$	$s(\ )_{\text{total}}$	$\varepsilon_{\text{total}}$
11	2	50	6	1	70	3	209
11	6	0	14	1	32	15	480
22	5	0	12	0	39	25	963
11	14	0	30	1	56	10	555
11	2	0	8	0	21	45	923
0	0	0	4	25	29	5	145
							3274
11	2	50	6	1	70	3	209
11	8	0	18	1	38	6	225
11	6	0	14	1	32	9	288
22	5	0	12	0	39	15	578
11	14	0	30	1	56	12	666
11	2	0	8	0	21	53	1087
0	0	0	6	25	31	5	155
							3207

Through visual inspection, lace density at interactions labeled ○ appear about twice as dense as other interactions.

Each category of effort—perceiving, recalling, preparing, working, finishing—is calculated according to the equations in Section 3.2(b). An expertise coefficient of 1 has been assigned in all categories, and the effort variables used are listed in the table. A final effort score is tallied as a summation over these categories.

These manual examples serve to illustrate how to calculate effort from a lace design. As a hybridization of standard lace patterns with the succinct and powerful directed graph representation, this notation retains the spatial accuracy of prickings while labeling the interactions to facilitate counting effort. Based on existing lace, no consideration was given to the generation of design, and features that might be more difficult to generate automatically, such as spiders or triangles, for example, could be included in the examples.

An obvious limitation to this approach is that the effort required to manually notate the pattern prohibits the analysis many or large designs. In particular, calculating stitch density is either prohibitively time-consuming or subject to large approximations as was the case in the examples. Another limitation is that the analysis is retrospective and provides no information related to how the lace was physically made. For these reasons, it is necessary to generate lace designs automatically for more comprehensive analysis.

3.3 (b) Automated Calculations of Generated Lace

The following objectives outline the requirements of a computer program to generate lace designs for effort analysis:

- Generate valid (makeable) lace designs
- Calculate instances of effort
- Track the order in which stitches are made
- Track individual thread locations as they propagate through the design

As a form of computation, the low-resolution making grammar described in Section 3.1(c) is the basis for an automated generation of valid lace designs. The making grammar guarantees that the lace can be made when its rules are applied and the pseudocode outlined in Algorithms 1-11 show the steps by which the making grammars are translated into computer code.

First, a procedure initializes a torchon grid. The grid is made of nodes in horizontal rows and vertical columns and are zero-indexed from the left and top of the lace when oriented to the page. For example, the top left corner is in position [0,0]. A spacing parameter ensures that the odd rows are staggered from the even ones so that the grid is forty-five degrees diagonal square. Node objects are initialized at the (x, y) location of each point on the grid.

```
Algorithm 1: TORCHON($cols, rows$)
  Input: $cols$, Number of columns in design
  Input: $rows$, Number of rows in design
  Input: $cell$, spacing parameter for grid
  Data: $i$, Index of a column in the design
  Data: $j$, Index of a row in the design
  Data: $n$, A node of the design located at $[i, j]$
  Result: $N$, A set of $n$ nodes on a 45° Torchon grid
  begin
      for $i = 0$ to $cols - 1$ do
          for $j = 0$ to $rows - 1$ do
              if $row$ $is$ $even$ then
                  Initialize $n$ at $(i \times cell, j \times \frac{cell}{2})$;
                  Increment $j$ by 1;
              else
                  Initialize $n$ at $(\frac{cell}{2} + i \times cell \times i, j \times \frac{cell}{2})$;
                  Remove $n$ at $[i = cols, j]$;
                  Increment $j$ by 1;
              end
              Increment $i$ by 1;
          end
      end
  end
```

These nodes form the input to the Distorchon routine that uses a physics engine to simulate a mass-spring system with edges connecting nodes according to the directed graph representation of torchon lace. When the computer mouse is clicked, *attractor points* are generated to change the shape of the grid through bunching behavior. When a key is pressed, the physics simulation stops and a new set of nodes is initialized in place.

The node objects are capable of being in a variety of states: Initial, Legal, Selected, and

Worked. These states parallel those for the nodes in the low-resolution making grammar.

The Initial state is applied to a node when it is located in the first row of the design.

```
Algorithm 2: DIS-TORCHON($n$)
  Input: $N$, A set of $n$ nodes on a 45° Torchon grid
  Data: $i$, Index of a column in the design
  Data: $j$, Index of a row in the design
  Data: $v$, particle object in physics simulator located at $[i, j]$
  Data: $v.x$, The $x$ position of particle $v$ during ATTRACTOR
  Data: $v.y$, The $y$ position of particle $v$ during ATTRACTOR
  Data: $mouseX$, The $x$ location of the mouse on the screen
  Data: $mouseY$, The $y$ location of the mouse on the screen
  Data: $work$, Boolean variable
  Data: ATTRACTOR, A function that creates "attractor" points with
        mass-spring physics simulation behavior acting on $v$
  Data: MOUSEPRESSED, A function that executes when the computer
        mouse is clicked
  Data: KEYPRESSED, A function that reads the keyboard input
  Result: $P$, A set of $p$ nodes on a distorted Torchon grid
  begin
      Initialize $v$ at every $n$;
      Connect all $v$ with springs with torchon grid topology;
      if MOUSEPRESSED then
          ATTRACTOR($mouseX, mouseY$)
      end
      if KEYPRESSED = 'S' then
          Stop the physics simulation;
      end
      if KEYPRESSED = 'P' then
          Initialize $p$ at $(v.x, v.y)$;
          if KEYPRESSED = 'W' then
              Set $work$ to $true$;
          end
      end
  end
```

Before a node can be selected or worked, it must be Legal. The program assumes that bobbins have been properly wound and mounted at the top row, and therefore nodes that are Initial are also Legal and ready to be worked. Further down in the lace, a node is Legal if the corresponding nodes above it have been Worked. In the directed graph representation of lace, the corresponding nodes are located in a row above and share an *edge* with the node. In the low-resolution making grammar, Legal nodes are represented with a grey circle.

If a node is Legal it may then be Selected. In lacemaking, this step is at the discretion of the lacemaker. An experienced lacemaker will know that in torchon ground, it is easier

Algorithm 3: INITIAL(p)

```
Input: $p$, A node of the design located at $[i, j]$
Data: $i$, Index of a column in the design
Data: $j$, Index of a row in the design
Data: $work$, Boolean variable
Result: True if $p$ is located on the first row
begin
    if $work$ then
        Retrieve location $[i, j]$ from $p$;
        if $[i, 0]$ then
            return $true$
        end
    end
end
```

Algorithm 4: LEGAL(p)

```
Input: $p$, A node of the design located at $[i, j]$
Data: $p'$, A node in a row above $p$ that shares an edge with $p$
Data: $p''$, A node in a row above $p$ that shares an edge with $p$
Data: $i$, Index of a column in the design
Data: $j$, Index of a row in the design
Result: True if $p$ is located on the first row or criteria are met
begin
    if INITIAL($p$) then
        return $true$
    else if $not$ INITIAL($p$) then
        Ensure WORKED($p'$);
        Ensure WORKED($p''$);
        return $true$ if both conditions are met
    end
end
```

to work the lace in diagonal rows because the new working group will share bobbins from the previous working group and make the process of identifying the four active bobbins easier. It possible, however, to work the lace wherever a Legal node is; the Routine procedure returns a list of ordered node locations reflecting three possible ways to work the lace: in diagonal columns, in back and forth rows, or at random as nodes become legal.

The list of node locations returned from Routine is then iterated through by the Search procedure to Select a Legal node.

```
Algorithm 5: ROUTINE($L$)
  Input: $raster$, Boolean variable
  Input: $diagonal$, Boolean variable
  Input: $random$, Boolean variable
  Input: $L$, A list of node locations in the design stored as $[i, j]$ and
         ordered by column and row position
  Data: $i$, Index of a column in the design
  Data: $j$, Index of a row in the design
  Result: $L'$, A list of node locations in the design stored as $[i, j]$ and
          ordered by the search routine
  begin
      if $raster$ then
          $L'$ is a reordering of $L$ such that node locations are listed as a
           progression of rows in alternating directions
      else if $diagonal$ then
          $L'$ is a reordering of $L$ such that stitches are retrieved on
           successive diagonal columns
      else if $random$ then
          $L'$ is a random reordering of $L$
      end
  end
```

```
Algorithm 6: SEARCH($L'$)
  Input: $L'$, A list of node locations in the design stored as $[i, j]$ and
         ordered by the search routine
  Data: $i$, Index of a column in the design
  Data: $j$, Index of a row in the design
  begin
      for $l = 0$ to $L'$ do
          Retrieve location $[i, j]$ from $l$;
          SELECT($p$) at $[i, j]$;
          Increment $l$ by 1;
      end
  end
```

```
Algorithm 7: SELECT($p$)
  Input: $p$, A node of the design located at $[i, j]$
  Data: $i$, Index of a column in the design
  Data: $j$, Index of a row in the design
  Result: True if $p$ meets all criteria
  begin
      Ensure LEGAL($p$);
      Ensure not WORKED($p$);
      return true if all conditions are met
  end
```

The node at a position retrieved from Routine is checked in Select to be Legal and not Worked; this check ensures that the node is capable of being worked—it is legal—and has not been worked already so that the computation will only move forward (especially important in the random routine, where locations might otherwise be worked a second time and the effort-score miscalculated).

If all conditions are met, Select returns true and changes the state of the node to Worked.

The Worked node is unable to be selected again, but the node retains the state of Worked and can therefore meet a condition for corresponding nodes below to be Legal according to that procedure.

```
Algorithm 8: WORKED($p$)
  Input: $p$, A node of the design located at $[i, j]$
  Data: $i$, Index of a column in the design
  Data: $j$, Index of a row in the design
  Result: True if $p$ meets all criteria
  begin
      Ensure SELECT($p$);
      return true if condition is met
  end
```

It is important to note that identical lace can be produced from alternative ways of working a pattern. If the lacemaker chooses to work the lace on diagonal columns, for example, she will produce an identical lace by working it in alternating rows. In fact, a lacemaker will not work the lace in any one particular way all the way through. This is especially true when the design features spiders, zigzags, or other motifs that require certain topological context in order to be deployed. It is an important point for this thesis, however, to show that alternative ways of working (or generating) a pattern can entail varying efforts. Therefore, the program assigns a unique number to each thread "mounted" on the initial row and tracks the propagation of the threads through the design.

Each node in the first row of the design is assigned a unique set of four integer values corresponding to four threads—two pairs of bobbins—at each location. Starting on the left side, the threads are zero-indexed and counted incrementally by one until the end of the row. The tetrad of threads at each node is stored in a set ordered $\{a, b, c, d\}$. When a node below the initial row is WORKED, a new set is created from $\{c, d\}$ of the corresponding node above on the left and $\{a, b\}$ of the corresponding node above on the right, reflecting the way in which the working group is selected from partial sets of bobbins at the corresponding nodes above. Without changing position, the input threads are stored as $\{a, b, c, d\}$ and written to a Table as "Threads In." Threads change position according to the ζ function at each vertex. In the program, each node is treated as a *torchon ground*: $\zeta(v) = CTpCT \; \forall \; v \in V$.

The input threads in position $\{a, b, c, d\}$ are in position $\{d, c, b, a\}$ after the interaction and the reordered set is written to a Table as "Threads Out." Without reordering, the output threads are stored as $\{a, b, c, d\}$ for use in the corresponding nodes below.

Algorithm 9: THREADS(p)

```
Input: $p$, A node of the design located at $[i, j]$
Data: $p'$, A node in a row above $p$ that shares an edge with $p$
Data: $p''$, A node in a row above $p$ that shares an edge with $p$
Data: $i$, Index of a column in the design
Data: $j$, Index of a row in the design
Data: $threadsIn$, A set of unique thread identification numbers in
      position $\{a, b, c, d\}$ at the beginning of the interaction at $p$
Result: $threadsOut$, A set of unique thread identification numbers in
        position $\{a, b, c, d\}$ at the end of the interaction at $p$
begin
    if SELECT($p$) then
        if INITIAL($p$) then
            $g = i * 4$;
            $threadsOut = \{g, g + 1, g + 2, g + 3\}$;
        else
            Determine $threadsIn$ at $p$ from $threadsOut$ at $p'$ and $p''$;
            Reorder $threadsIn$ as $\{d, c, b, a\}$ and store as $threadsOut$;
        end
        Write $threadsIn$ and $threadsOut$ to Table;
    end
end
```

As the program generates a valid lace, data is written out to a Table for each node where Select returns true. Writing the data at each instance where Select is true tracks the effort-cost at each step of production.

The resultant Table is a history of how the lace was produced, ordered from the first interaction at the top of the table through the final interaction at the bottom. Each interaction is labeled by a location identification double of the form $[i, j]$ indicating the column and row index in the design.

A comparison of "Threads In" at the present instance of Select to "Threads Out" at the immediately previous instance determines whether the working group is "Different" or "Shared" between the two interactions. A "Shared" instance, which occurs when a node has been worked at a diagonal (see Routine), will trigger a reduced multiplier for calculating the perceptual effort of identifying the working pairs, $e_{perceive.pairs}$.

The context in which Select occurs is reflected by a low-resolution making grammar rule that is written to the column "Rule Applied." Rule 2, for example is the "mounting rule" and will occur when Select occurs on the initial row (see Initial). The history of applied rules underscores a claim of this thesis that a program for generating lace is an automated making grammar.

From the effort equations, perceptual effort is a function of the density at a node being worked. Here, the density is defined as a function of an area between the present node p and the two corresponding nodes p' and p'' in a row (or rows if the p is located on an edge—see low-resolution making grammar Rule 10 in Section 3.1(c)) above p.

To calculate the area, a unique color—a function of the $[i,j]$ node identification—fills the triangle with vertices p, p', p''. The number of pixels is counted in this triangle and output to the Table as "Working Area" and used to calculate the density at each node.

```
Algorithm 10: TABLE($p$)
Input: $p$, A node of the design located at $[i, j]$
Data: $p'$, A node in a row above $p$ that shares an edge with $p$
Data: $p''$, A node in a row above $p$ that shares an edge with $p$
Data: $i$, Index of a column in the design
Data: $j$, Index of a row in the design
Data: $threadsIn$, A set of unique thread identification numbers in
      position $\{a, b, c, d\}$ at the beginning of the interaction at $p$
Data: $threadsOut$, A set of unique thread identification numbers in
      position $\{a, b, c, d\}$ at the end of the interaction at $p$
Output: "$[i, j]$", The location identification of $p$
Output: "Threads In", A string of characters identifying the threads
        and positions of threads of $threadsIn$
Output: "Threads Out", A string of characters identifying the threads
        and positions of threads of $threadsOut$
Output: "Working Group", A String of characters indicating if
        bobbins at $p$ were "DIFFERENT" from or "SHARED" with
        the immediately prior instance of SELECT($p$)
Output: "Rule Applied", A string of characters indicating what
        low-resolution making grammar rule was applied at $p$
Output: "Working Area", The number of pixels in the triangular
        space between $p$, $p'$, and $p''$ and used to calculate $\rho(p)$
Result: A Table of data
begin
    if SELECT($p$) then
        Write a new row of data to each Output column in the Table;
    end
end
```

```
Algorithm 11: displayPoints
Input: $i$, Index of a column in the design
Input: $j$, Index of a row in the design
Result: A display
begin
    if select($p$) then
        Display a yellow circle;
        Display a rectangle around $p$;
    else if initial($p$) then
        Display a red circle;
    else if legal($p$) then
        Display a grey circle;
    else
        Display the outline of a circle;
    end
end
```

3.4 Selections from the Design Space

There is a large design space of makeable lace designs generated by the program outlined in Section 3.3(b). The lace may be of any number of rows and columns and attractor points may be placed anywhere within the boundaries of the lace to promote bunching behavior. The simulated spring constant at each segment of the lace may be adjusted, and affects the strength of the bunching behavior.

The initial density of the lace is also adjustable by changing the initial grid spacing. In the examples provided here, the same initial scaling factor is used and corresponds to interactions spaced approximately every half inch before distortion. The designs are presented at a 1:1 scale.

The values for the effort components may also be adjusted to change the relative effort contributions from each category. It should be noted that the values have been chosen to approximate effort for working lace of the size presented here and not, for example, large experimental work such as that described in Section 2.5. Of particular interest is changing the search routine to show that embodied effort is dependent on how the lace was made.

3.4 (a) Equivalent Designs with Different Effort

Effort-cost is calculated for a small design made in three different ways. First working in diagonal rows, the complete effort tabulation is given in the next pages. Complete tabulations for working in horizontal rows and at random are given in the Appendix. These tabulations show that the topology of the lace remains the same irrespective of how it is worked. A comparison of thread identities entering and leaving each node demonstrates the topological equivalence. Setting all other values for the effort components equal, the difference in embodied effort is dependent on the delta coefficient of $e_{perceiving}$ when the lace is worked in different ways. This coefficient is chosen as $\delta = 2$ when the lace is worked in diagonal rows because in that way of working two of the four bobbins are selected for the next interaction. When working in horizontal rows, a new set of bobbins is selected and $\delta = 4$. In the case of working at random, a new set of bobbins is selected, but will (likely) not be adjacent and will require perceiving more of the work to identify the next set. Therefore, a higher

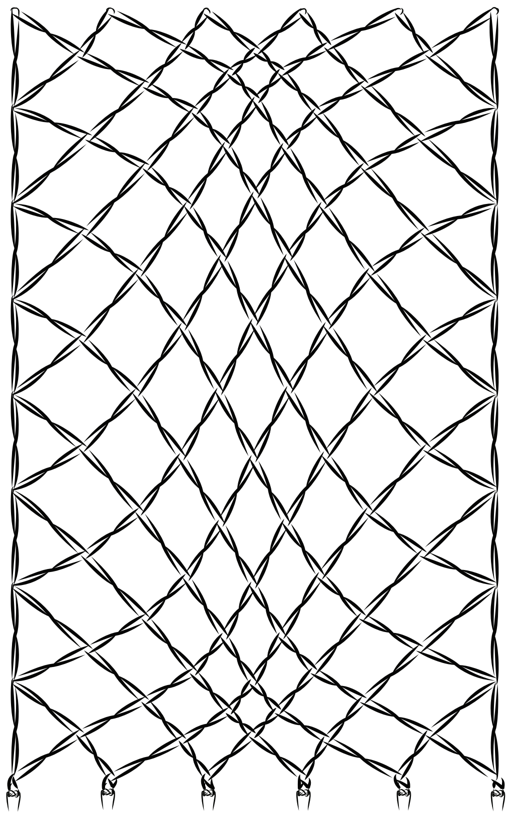


__Figure 3.13:__ A lace pattern worked three ways.

value still is attributed: $\delta = 8$. While these δ values are estimations, they reflect the fact that it is easiest to work torchon lace in diagonal rows, and most difficult to work it at random. An alternative way of producing an equivalent design with different expenditures of effort is to apply differing levels of expertise to the process. A larger design (Figure 3.14) is used to make the effects more pronounced and Table 3.2 shows the effect of working this design three ways. Expertise coefficients are adjusted to tabulate effort required for a novice (level 1) through an expert (level 10).

Working Routine	***Total Embodied Effort***
Diagonal rows	13125
Back-and-forth rows	16156
Random	21830

Table 3.2: *Effort scores for a lace worked in three ways.*

Expertise Level *(all categories)*	***Total Embodied Effort***
1 (novice)	13125
2	6563
3	4375
4	3281
5	2625
6	2187
7	1875
8	1640
9	1458
10 (expert)	1313

Table 3.3: *Effort scores for a lace worked with different levels of expertise.*

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[0,0]	0, 0, 0, 0	0, 1, 2, 3	Different	Rule 2	2	2	1
[1,0]	0, 0, 0, 0	4, 5, 6, 7	Different	Rule 2	2	2	1
[0,1]	2, 3, 4, 5	5, 4, 3, 2	Shared	Rule 3	2	2	1
[0,2]	0, 1, 5, 4	4, 5, 1, 0	Shared	Rule 10	2	2	1
[2,0]	0, 0, 0, 0	8, 9, 10, 11	Shared	Rule 2	2	2	1
[1,1]	6, 7, 8, 9	9, 8, 7, 6	Shared	Rule 4	2	2	1
[1,2]	3, 2, 9, 8	8, 9, 2, 3	Shared	Rule 5	2	2	1
[0,3]	1, 0, 8, 9	9, 8, 0, 1	Shared	Rule 5	2	2	1
[0,4]	4, 5, 9, 8	8, 9, 5, 4	Shared	Rule 10	2	2	1
[3,0]	0, 0, 0, 0	12, 13, 14, 15	Different	Rule 2	2	2	1
[2,1]	10, 11, 12, 13	13, 12, 11, 10	Shared	Rule 4	2	2	1
[2,2]	7, 6, 13, 12	12, 13, 6, 7	Shared	Rule 5	2	2	1
[1,3]	2, 3, 12, 13	13, 12, 3, 2	Shared	Rule 5	2	2	1
[1,4]	0, 1, 13, 12	12, 13, 1, 0	Shared	Rule 5	2	2	1
[0,5]	5, 4, 12, 13	13, 12, 4, 5	Shared	Rule 5	2	2	1
[0,6]	8, 9, 13, 12	12, 13, 9, 8	Shared	Rule 10	2	2	1
[4,0]	0, 0, 0, 0	16, 17, 18, 19	Different	Rule 2	2	2	1
[3,1]	14, 15, 16, 17	17, 16, 15, 14	Shared	Rule 4	2	2	1
[3,2]	11, 10, 17, 16	16, 17, 10, 11	Shared	Rule 5	2	2	1
[2,3]	6, 7, 16, 17	17, 16, 7, 6	Shared	Rule 5	2	2	1
[2,4]	3, 2, 17, 16	16, 17, 2, 3	Shared	Rule 5	2	2	1
[1,5]	1, 0, 16, 17	17, 16, 0, 1	Shared	Rule 5	2	2	1
[1,6]	4, 5, 17, 16	16, 17, 5, 4	Shared	Rule 5	2	2	1
[0,7]	9, 8, 16, 17	17, 16, 8, 9	Shared	Rule 5	2	2	1
[0,8]	12, 13, 17, 16	16, 17, 13, 12	Shared	Rule 10	2	2	1
[5,0]	0, 0, 0, 0	20, 21, 22, 23	Different	Rule 2	2	2	1
[4,1]	18, 19, 20, 21	21, 20, 19, 18	Shared	Rule 4	2	2	1
[4,2]	15, 14, 21, 20	20, 21, 14, 15	Shared	Rule 5	2	2	1
[3,3]	10, 11, 20, 21	21, 20, 11, 10	Shared	Rule 5	2	2	1
[3,4]	7, 6, 21, 20	20, 21, 6, 7	Shared	Rule 5	2	2	1
[2,5]	2, 3, 20, 21	21, 20, 3, 2	Shared	Rule 5	2	2	1
[2,6]	0, 1, 21, 20	20, 21, 1, 0	Shared	Rule 5	2	2	1

Table 3.4: *Complete effort-tabulation for a lace worked in diagonal rows.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	4	2	4	26	2	20	18	11	77	77
1	4	2	1	6	2	20	18	11	57	134
1	2	0	1	16	2	0	18	1	37	171
1	2	0	0	12	2	0	18	1	33	204
1	2	2	1	16	2	20	18	11	67	271
1	2	0	1	16	2	0	18	1	37	308
1	2	0	1	24	2	0	18	1	45	353
1	2	0	0	14	2	0	18	1	35	388
1	2	0	0	11	2	0	18	1	32	419
1	4	2	1	26	2	20	18	11	77	496
1	2	0	1	16	2	0	18	1	37	533
1	2	0	3	49	2	0	18	1	70	604
1	2	0	2	31	2	0	18	1	52	655
1	2	0	1	17	2	0	18	1	38	693
1	2	0	0	12	2	0	18	1	33	726
1	2	0	0	11	2	0	18	1	32	758
1	4	2	1	26	2	20	18	11	77	835
1	2	0	1	16	2	0	18	1	37	872
1	2	0	4	66	2	0	18	1	87	959
1	2	0	5	74	2	0	18	1	95	1054
1	2	0	2	31	2	0	18	1	52	1106
1	2	0	1	19	2	0	18	1	40	1146
1	2	0	0	13	2	0	18	1	34	1180
1	2	0	0	11	2	0	18	1	32	1212
1	2	0	0	11	2	0	18	1	32	1244
1	4	2	1	26	2	20	18	11	77	1321
1	2	0	1	16	2	0	18	1	37	1358
1	2	0	2	26	2	0	18	1	47	1405
1	2	0	2	36	2	0	18	1	57	1462
1	2	0	2	37	2	0	18	1	58	1520
1	2	0	2	26	2	0	18	1	47	1567
1	2	0	1	19	2	0	18	1	40	1607

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[1,7]	5, 4, 20, 21	21, 20, 4, 5	Shared	Rule 5	2	2	1
[1,8]	8, 9, 21, 20	20, 21, 9, 8	Shared	Rule 5	2	2	1
[0,9]	13, 12, 20, 21	21, 20, 12, 13	Shared	Rule 5	2	2	1
[0,10]	16, 17, 21, 20	20, 21, 17, 16	Shared	Rule 10	2	2	1
[5,2]	19, 18, 22, 23	23, 22, 18, 19	Different	Rule 10	2	2	1
[4,3]	14, 15, 23, 22	22, 23, 15, 14	Shared	Rule 5	2	2	1
[4,4]	11, 10, 22, 23	23, 22, 10, 11	Shared	Rule 5	2	2	1
[3,5]	6, 7, 23, 22	22, 23, 7, 6	Shared	Rule 5	2	2	1
[3,6]	3, 2, 22, 23	23, 22, 2, 3	Shared	Rule 5	2	2	1
[2,7]	1, 0, 23, 22	22, 23, 0, 1	Shared	Rule 5	2	2	1
[2,8]	4, 5, 22, 23	23, 22, 5, 4	Shared	Rule 5	2	2	1
[1,9]	9, 8, 23, 22	22, 23, 8, 9	Shared	Rule 5	2	2	1
[1,10]	12, 13, 22, 23	23, 22, 13, 12	Shared	Rule 5	2	2	1
[0,11]	17, 16, 23, 22	22, 23, 16, 17	Shared	Rule 5	2	2	1
[0,12]	20, 21, 22, 23	23, 22, 21, 20	Shared	Rule 10	2	2	1
[5,4]	15, 14, 18, 19	19, 18, 14, 15	Different	Rule 10	2	2	1
[4,5]	10, 11, 19, 18	18, 19, 11, 10	Shared	Rule 5	2	2	1
[4,6]	7, 6, 18, 19	19, 18, 6, 7	Shared	Rule 5	2	2	1
[3,7]	2, 3, 19, 18	18, 19, 3, 2	Shared	Rule 5	2	2	1
[3,8]	0, 1, 18, 19	19, 18, 1, 0	Shared	Rule 5	2	2	1
[2,9]	5, 4, 19, 18	18, 19, 4, 5	Shared	Rule 5	2	2	1
[2,10]	8, 9, 18, 19	19, 18, 9, 8	Shared	Rule 5	2	2	1
[1,11]	13, 12, 19, 18	18, 19, 12, 13	Shared	Rule 5	2	2	1
[1,12]	16, 17, 18, 19	19, 18, 17, 16	Shared	Rule 5	2	2	1
[0,13]	21, 20, 19, 18	18, 19, 20, 21	Shared	Rule 5	2	2	1
[0,14]	23, 22, 18, 19	19, 18, 22, 23	Shared	Rule 10	2	2	1
[5,6]	11, 10, 14, 15	15, 14, 10, 11	Different	Rule 10	2	2	1
[4,7]	6, 7, 15, 14	14, 15, 7, 6	Shared	Rule 5	2	2	1
[4,8]	3, 2, 14, 15	15, 14, 2, 3	Shared	Rule 5	2	2	1
[3,9]	1, 0, 15, 14	14, 15, 0, 1	Shared	Rule 5	2	2	1
[3,10]	4, 5, 14, 15	15, 14, 5, 4	Shared	Rule 5	2	2	1
[2,11]	9, 8, 15, 14	14, 15, 8, 9	Shared	Rule 5	2	2	1

Table 3.4 (continued): *Complete effort-tabulation for a lace worked in diagonal rows.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	2	0	0	15	2	0	18	1	36	1643
1	2	0	0	12	2	0	18	1	33	1676
1	2	0	0	11	2	0	18	1	32	1708
1	2	0	0	12	2	0	18	1	33	1740
1	4	0	0	20	2	0	18	1	41	1781
1	2	0	0	15	2	0	18	1	36	1817
1	2	0	1	18	2	0	18	1	39	1856
1	2	0	1	20	2	0	18	1	41	1897
1	2	0	1	19	2	0	18	1	40	1938
1	2	0	1	18	2	0	18	1	39	1976
1	2	0	0	16	2	0	18	1	37	2013
1	2	0	0	13	2	0	18	1	34	2047
1	2	0	0	12	2	0	18	1	33	2080
1	2	0	0	11	2	0	18	1	32	2113
1	2	0	0	11	2	0	18	1	32	2145
1	4	0	0	18	2	0	18	1	39	2184
1	2	0	0	12	2	0	18	1	33	2217
1	2	0	0	14	2	0	18	1	35	2252
1	2	0	0	15	2	0	18	1	36	2288
1	2	0	0	15	2	0	18	1	36	2324
1	2	0	0	16	2	0	18	1	37	2360
1	2	0	0	15	2	0	18	1	36	2397
1	2	0	0	15	2	0	18	1	36	2432
1	2	0	0	15	2	0	18	1	36	2468
1	2	0	0	13	2	0	18	1	34	2502
1	2	0	0	11	2	0	18	1	32	2533
1	4	0	0	18	2	0	18	1	39	2572
1	2	0	0	11	2	0	18	1	32	2605
1	2	0	0	12	2	0	18	1	33	2638
1	2	0	0	14	2	0	18	1	35	2673
1	2	0	0	16	2	0	18	1	37	2710
1	2	0	1	17	2	0	18	1	38	2748

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[2,12]	12, 13, 14, 15	15, 14, 13, 12	Shared	Rule 5	2	2	1
[1,13]	17, 16, 15, 14	14, 15, 16, 17	Shared	Rule 5	2	2	1
[1,14]	20, 21, 14, 15	15, 14, 21, 20	Shared	Rule 5	2	2	1
[0,15]	22, 23, 15, 14	14, 15, 23, 22	Shared	Rule 5	2	2	1
[0,16]	19, 18, 14, 15	15, 14, 18, 19	Shared	Rule 10	2	2	1
[5,8]	7, 6, 10, 11	11, 10, 6, 7	Different	Rule 10	2	2	1
[4,9]	2, 3, 11, 10	10, 11, 3, 2	Shared	Rule 5	2	2	1
[4,10]	0, 1, 10, 11	11, 10, 1, 0	Shared	Rule 5	2	2	1
[3,11]	5, 4, 11, 10	10, 11, 4, 5	Shared	Rule 5	2	2	1
[3,12]	8, 9, 10, 11	11, 10, 9, 8	Shared	Rule 5	2	2	1
[2,13]	13, 12, 11, 10	10, 11, 12, 13	Shared	Rule 5	2	2	1
[2,14]	16, 17, 10, 11	11, 10, 17, 16	Shared	Rule 5	2	2	1
[1,15]	21, 20, 11, 10	10, 11, 20, 21	Shared	Rule 5	2	2	1
[1,16]	23, 22, 10, 11	11, 10, 22, 23	Shared	Rule 5	2	2	1
[5,10]	3, 2, 6, 7	7, 6, 2, 3	Different	Rule 10	2	2	1
[4,11]	1, 0, 7, 6	6, 7, 0, 1	Shared	Rule 5	2	2	1
[4,12]	4, 5, 6, 7	7, 6, 5, 4	Shared	Rule 5	2	2	1
[3,13]	9, 8, 7, 6	6, 7, 8, 9	Shared	Rule 5	2	2	1
[3,14]	12, 13, 6, 7	7, 6, 13, 12	Shared	Rule 5	2	2	1
[2,15]	17, 16, 7, 6	6, 7, 16, 17	Shared	Rule 5	2	2	1
[2,16]	20, 21, 6, 7	7, 6, 21, 20	Shared	Rule 5	2	2	1
[5,12]	0, 1, 2, 3	3, 2, 1, 0	Different	Rule 10	2	2	1
[4,13]	5, 4, 3, 2	2, 3, 4, 5	Shared	Rule 5	2	2	1
[4,14]	8, 9, 2, 3	3, 2, 9, 8	Shared	Rule 5	2	2	1
[3,15]	13, 12, 3, 2	2, 3, 12, 13	Shared	Rule 5	2	2	1
[3,16]	16, 17, 2, 3	3, 2, 17, 16	Shared	Rule 5	2	2	1
[5,14]	4, 5, 1, 0	0, 1, 5, 4	Different	Rule 10	2	2	1
[4,15]	9, 8, 0, 1	1, 0, 8, 9	Shared	Rule 5	2	2	1
[4,16]	12, 13, 1, 0	0, 1, 13, 12	Shared	Rule 5	2	2	1
[5,16]	8, 9, 5, 4	4, 5, 9, 8	Shared	Rule 10	2	2	1

Table 3.4 (continued): *Complete effort-tabulation for a lace worked in diagonal rows.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	2	0	1	20	2	0	18	1	41	2789
1	2	0	1	22	2	0	18	1	43	2832
1	2	0	1	20	2	0	18	1	41	2873
1	2	0	0	15	2	0	18	1	36	2909
1	2	0	0	12	2	0	18	1	33	2942
1	4	0	0	19	2	0	18	1	40	2982
1	2	0	0	11	2	0	18	1	32	3015
1	2	0	0	13	2	0	18	1	34	3048
1	2	0	0	15	2	0	18	1	36	3084
1	2	0	1	21	2	0	18	1	42	3126
1	2	0	2	28	2	0	18	1	49	3175
1	2	0	3	46	2	0	18	1	67	3242
1	2	0	2	34	2	0	18	1	55	3297
1	2	0	1	21	2	0	18	1	42	3339
1	4	0	0	20	2	0	18	1	41	3380
1	2	0	0	12	2	0	18	1	33	3413
1	2	0	0	15	2	0	18	1	36	3449
1	2	0	1	22	2	0	18	1	43	3492
1	2	0	3	42	2	0	18	1	63	3555
1	2	0	5	79	2	0	18	1	100	3655
1	2	0	2	36	2	0	18	1	57	3712
1	4	0	0	19	2	0	18	1	40	3752
1	2	0	0	13	2	0	18	1	34	3786
1	2	0	1	20	2	0	18	1	41	3827
1	2	0	2	31	2	0	18	1	52	3879
1	2	0	2	37	2	0	18	1	58	3937
1	4	0	0	18	2	0	18	1	39	3975
1	2	0	0	14	2	0	18	1	35	4011
1	2	0	1	21	2	0	18	1	42	4053
1	2	0	0	12	2	0	18	1	33	4086
									4086	

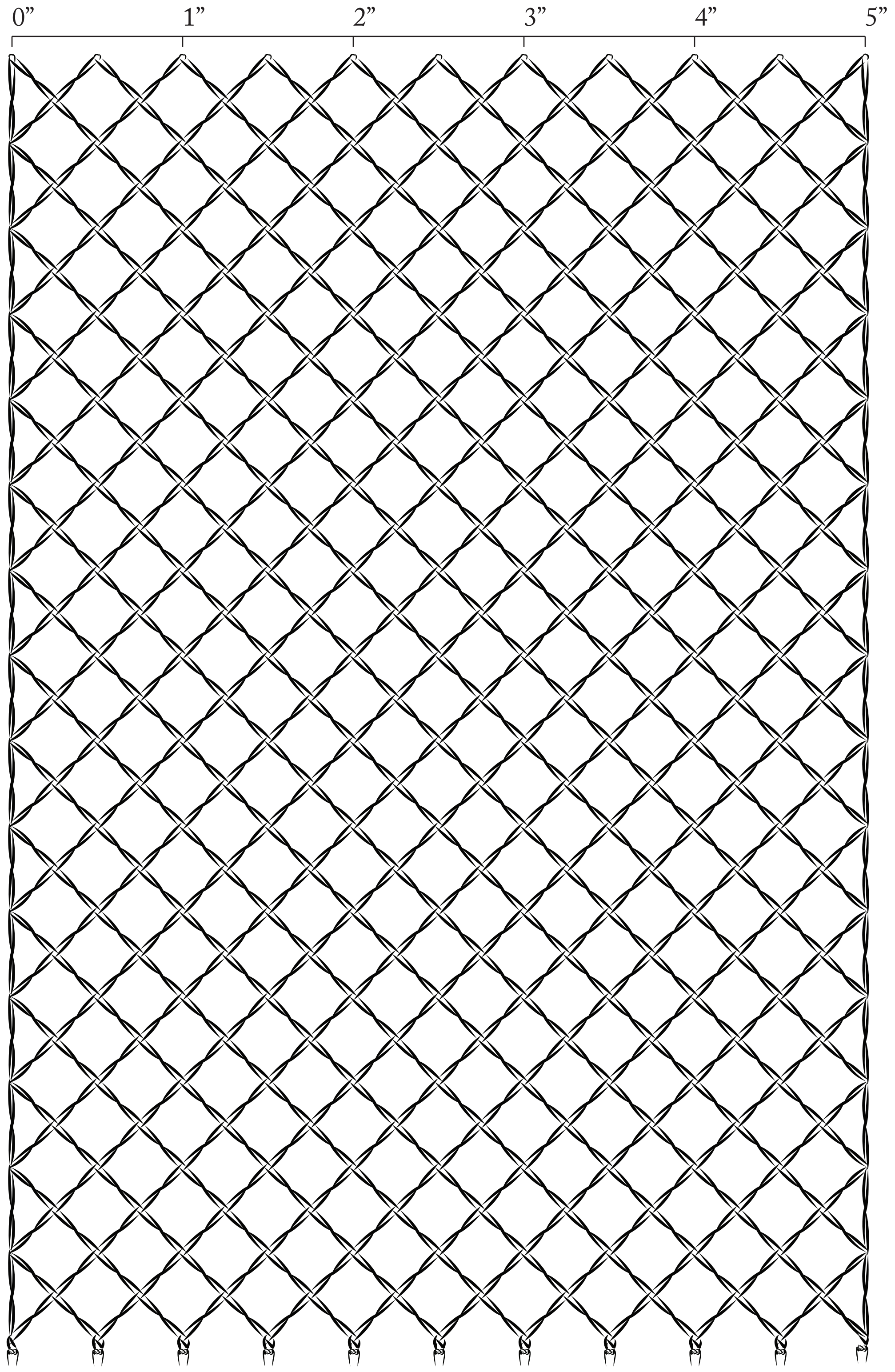


Figure 3.14: *A lace pattern worked three ways and with different levels of expertise.*

3.4 (b) Different Designs with Equivalent Effort

Given the large design space of lace and the difficulty of making it, it is an ambition of this thesis to find lace designs that are visually distinct but require nearly equivalent expenditures of effort. In calculating the embodied effort for visually distinct designs, the expertise coefficients are held constant, as are all effort component values. The lace designs are worked in diagonal rows.

Designs are grouped in effort-based equivalence classes requiring high, medium-high, medium-low, or low effort. A total effort score is given for each design as well as a graph showing the amount of total embodied effort in the design through production.

High Effort

Figure 3.15

Figure 3.16

Medium-High Effort

Figure 3.17

Figure 3.18

Figure 3.19

Medium-Low Effort

Figure 3.20

Figure 3.21

Figure 3.22

Low Effort

Figure 3.23

Figure 3.24

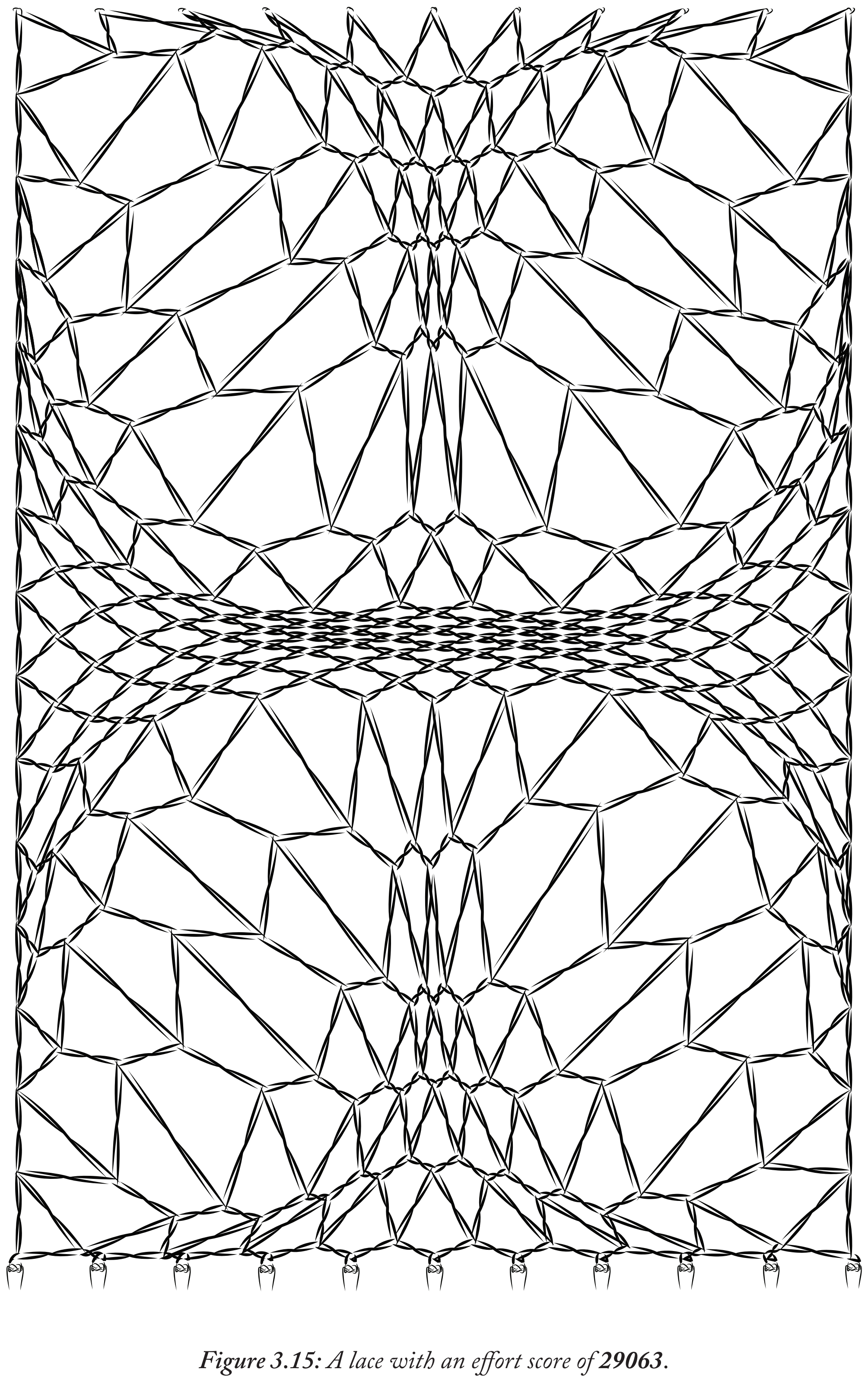


*Figure 3.15: A lace with an effort score of **29063**.*

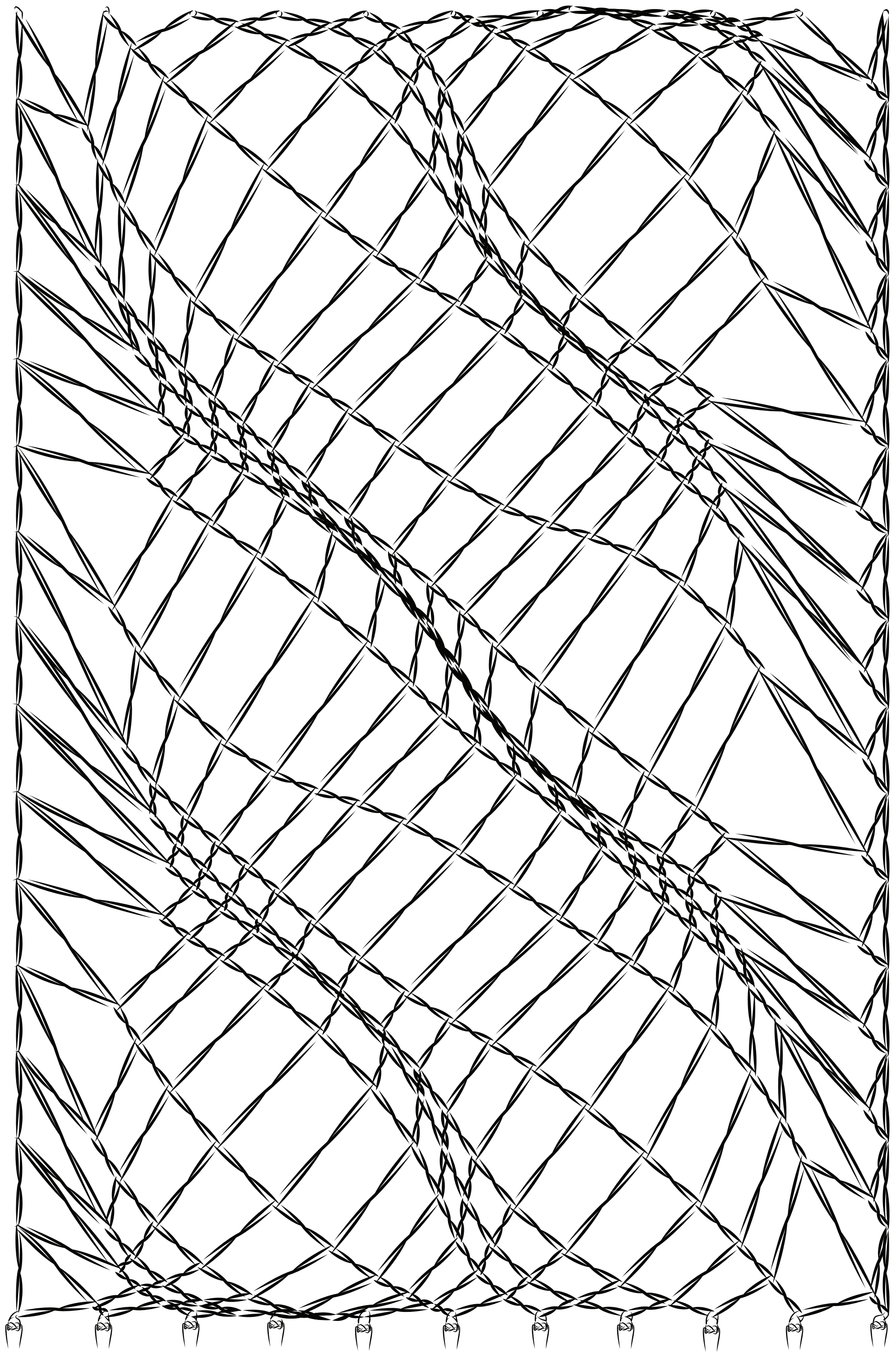


*Figure 3.16: A lace with an effort score of **32194**.*

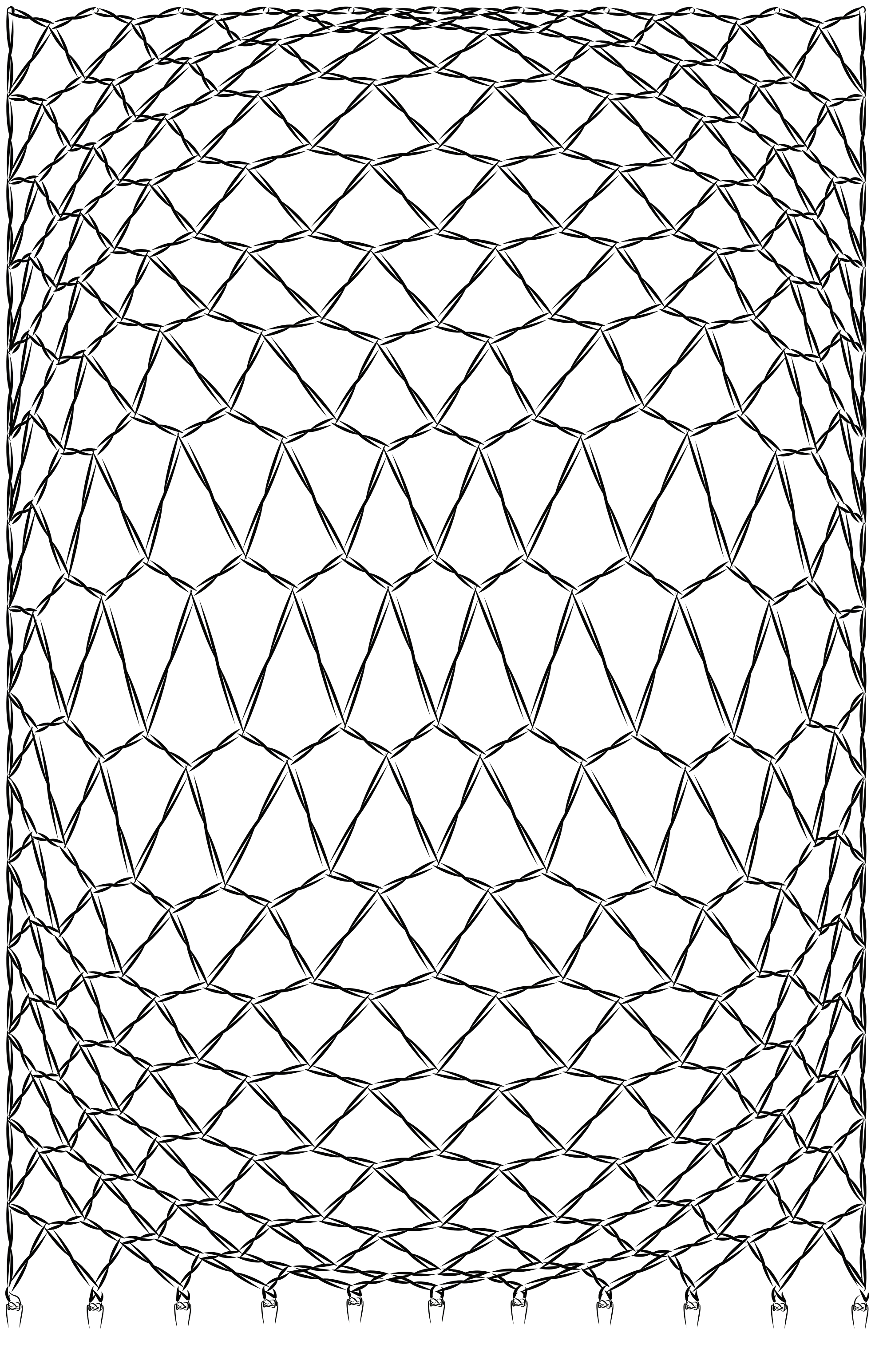


*Figure 3.17: A lace with an effort score of **23967**.*

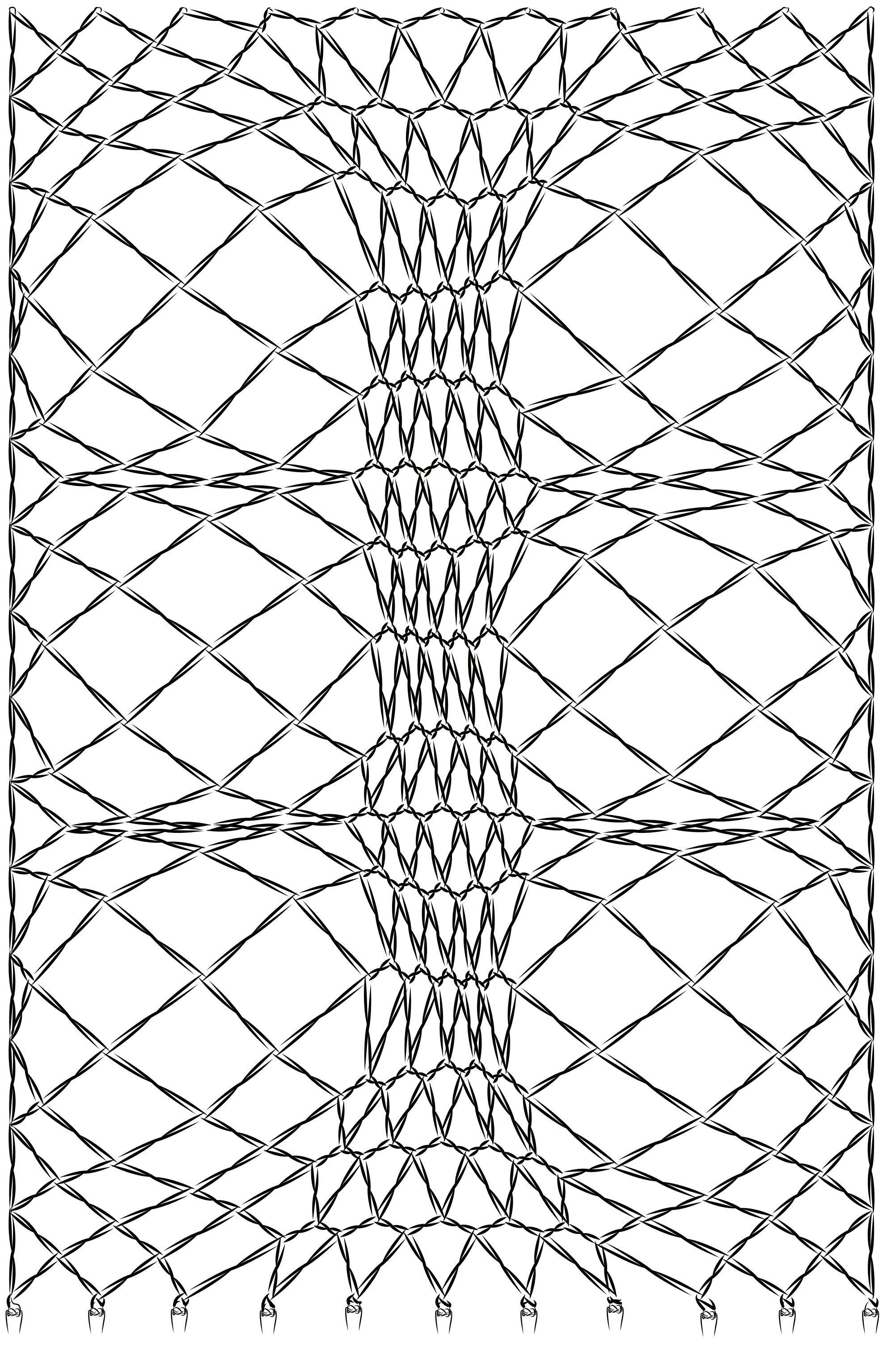


Figure 3.18: *A lace with an effort score of* ***26020***.

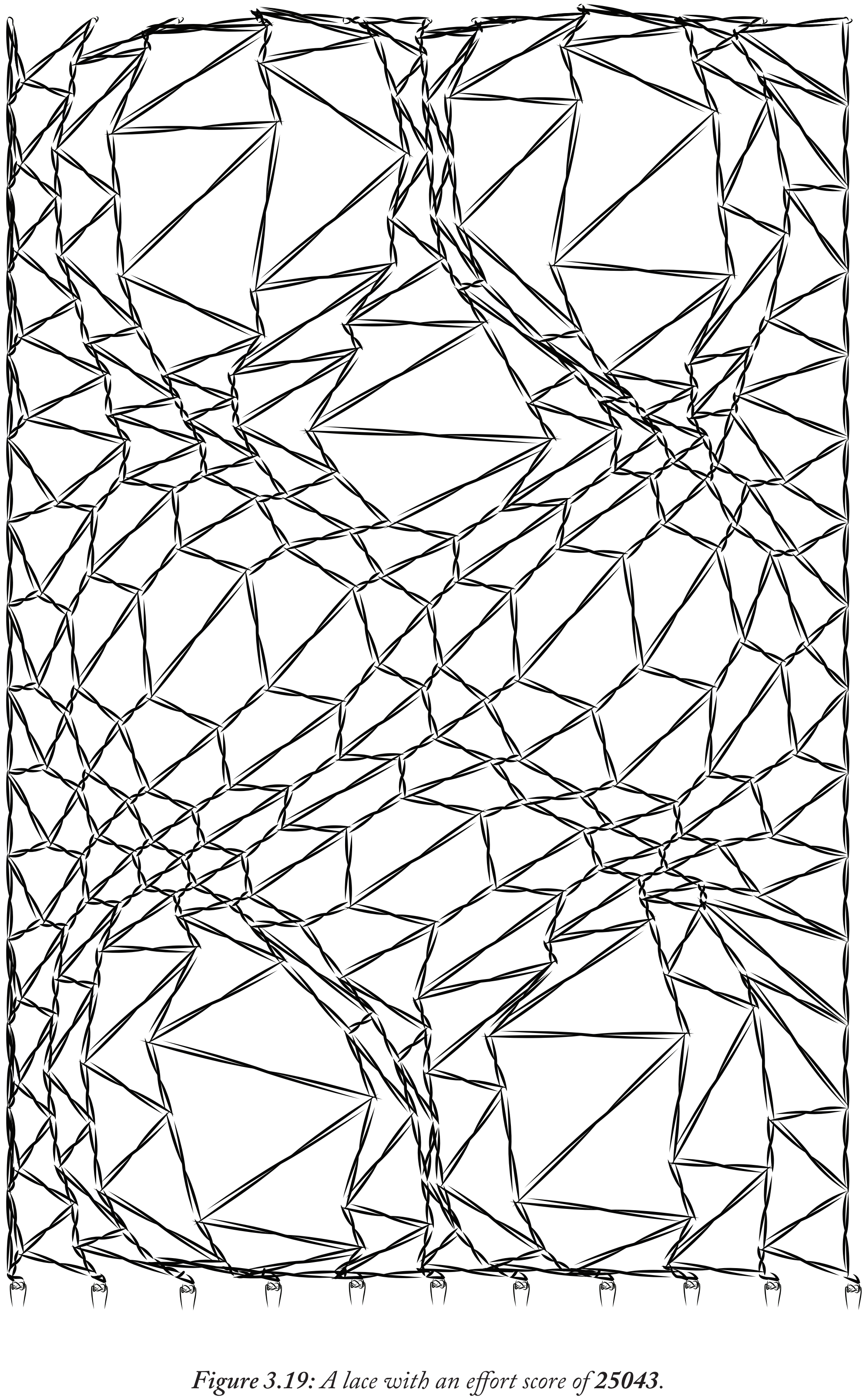


Figure 3.19: *A lace with an effort score of* ***25043***.

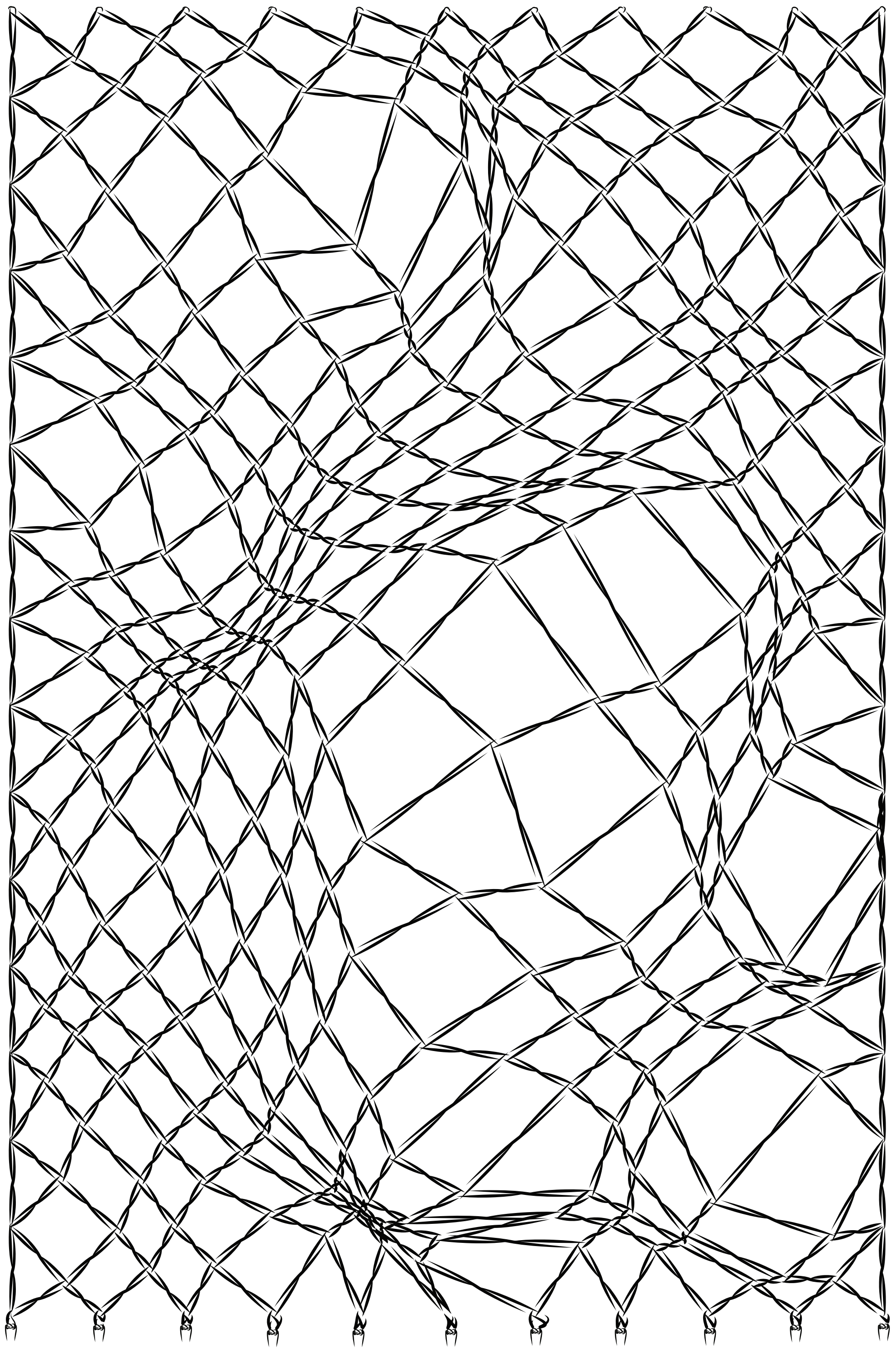


Figure 3.20: *A lace with an effort score of* ***25125***.

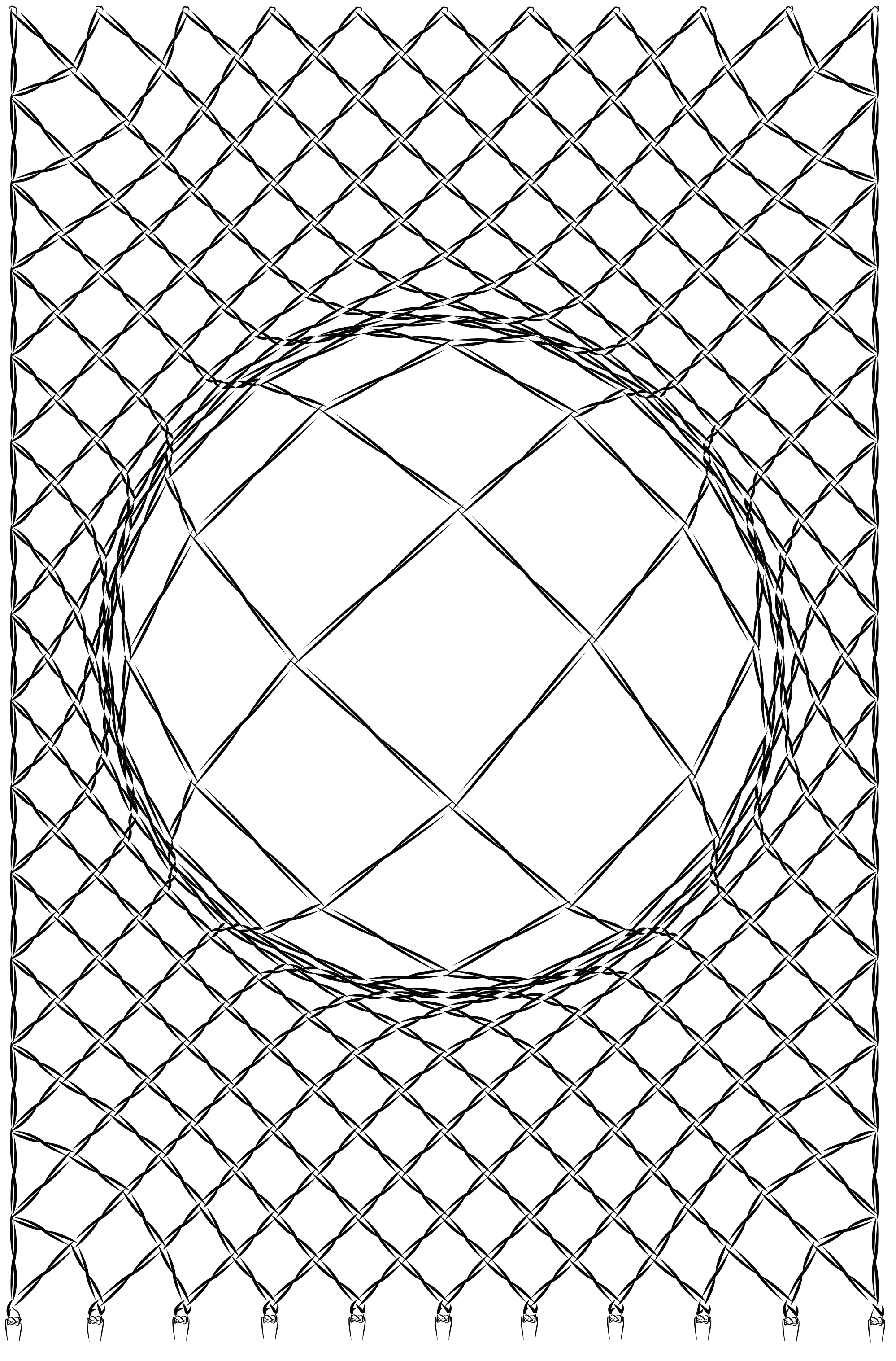


Figure 3.21: A lace with an effort score of ***20962***.

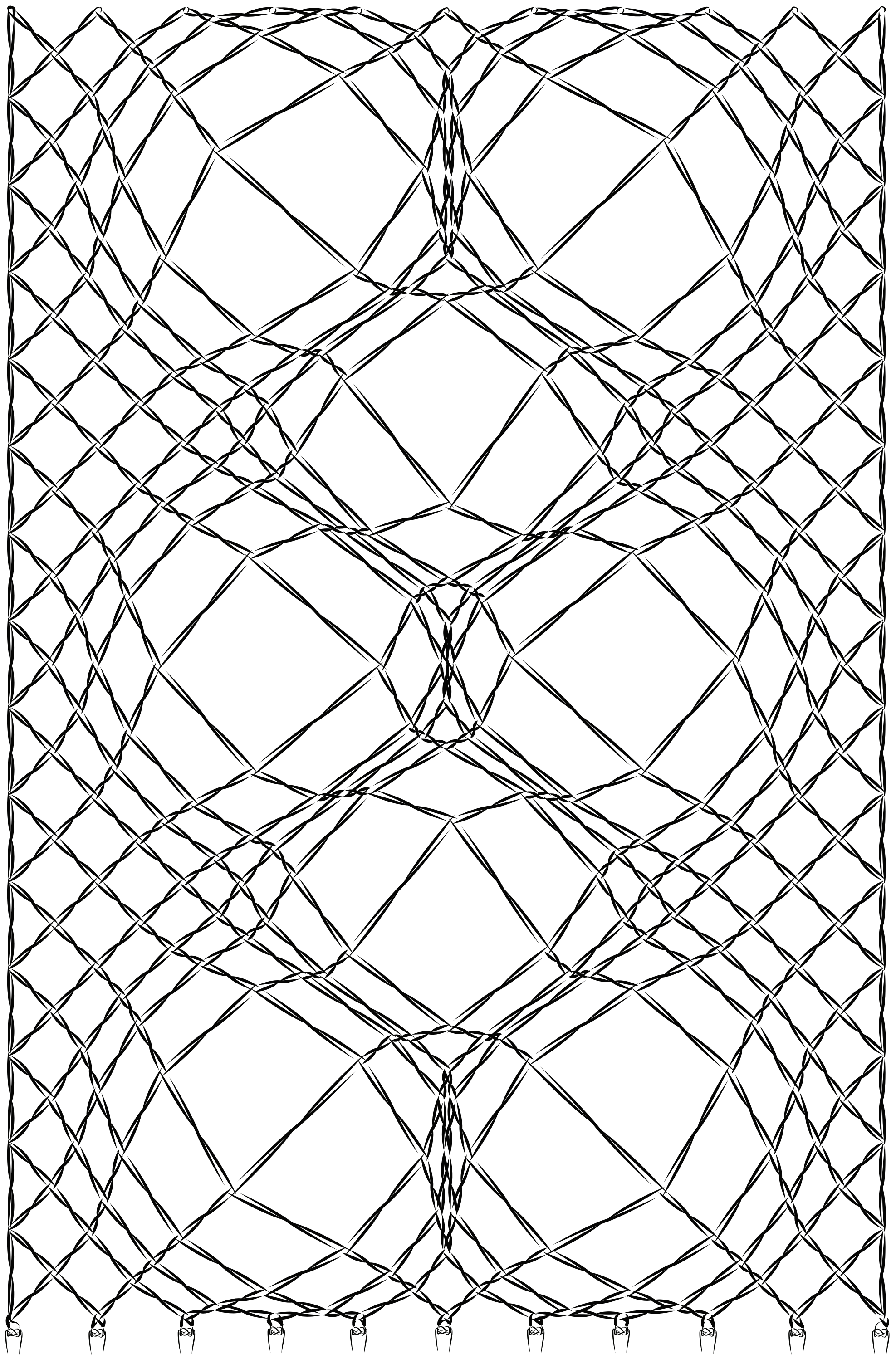


Figure 3.22: A lace with an effort score of ***20000****.*

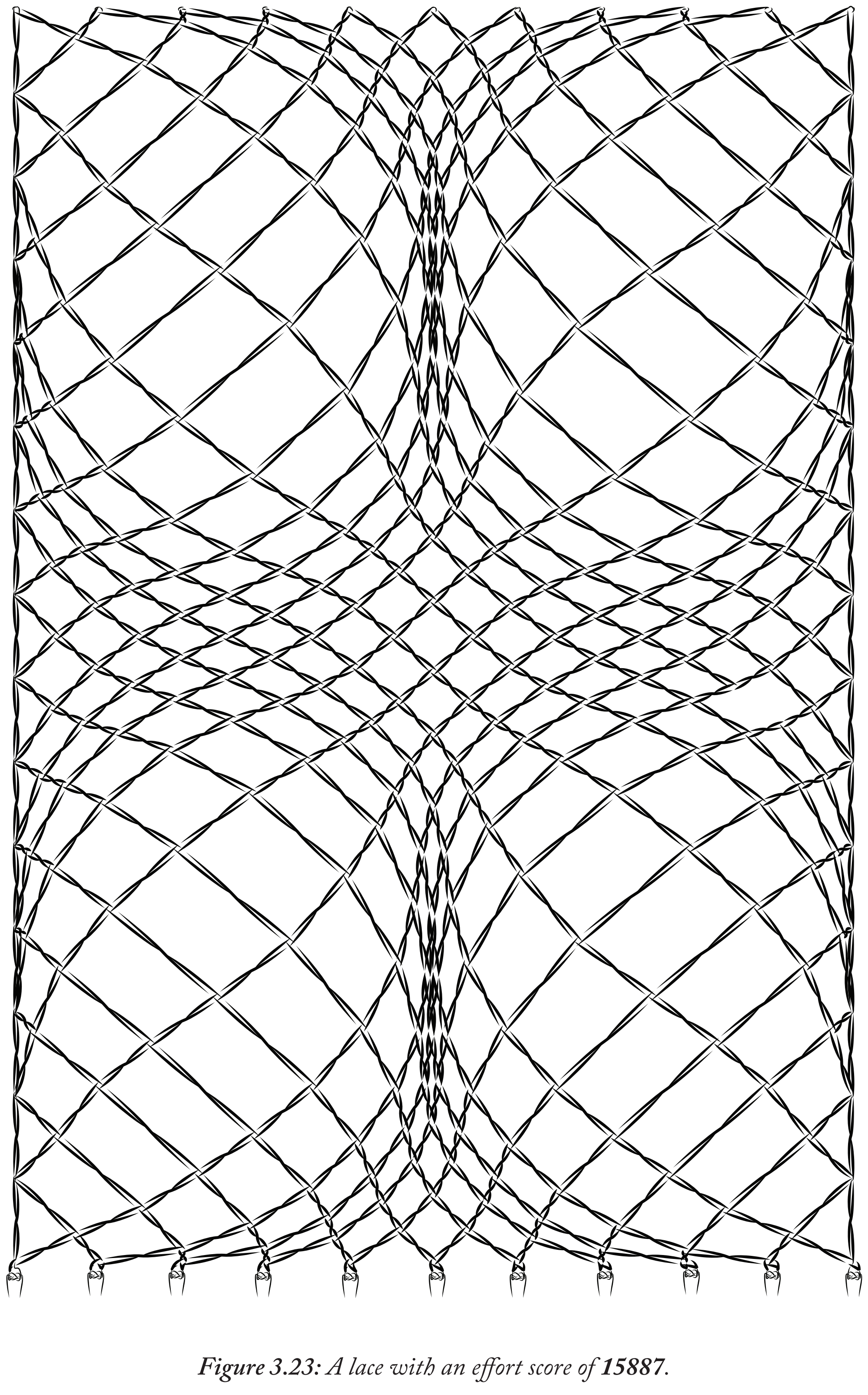


Figure 3.23: A lace with an effort score of ***15887****.*

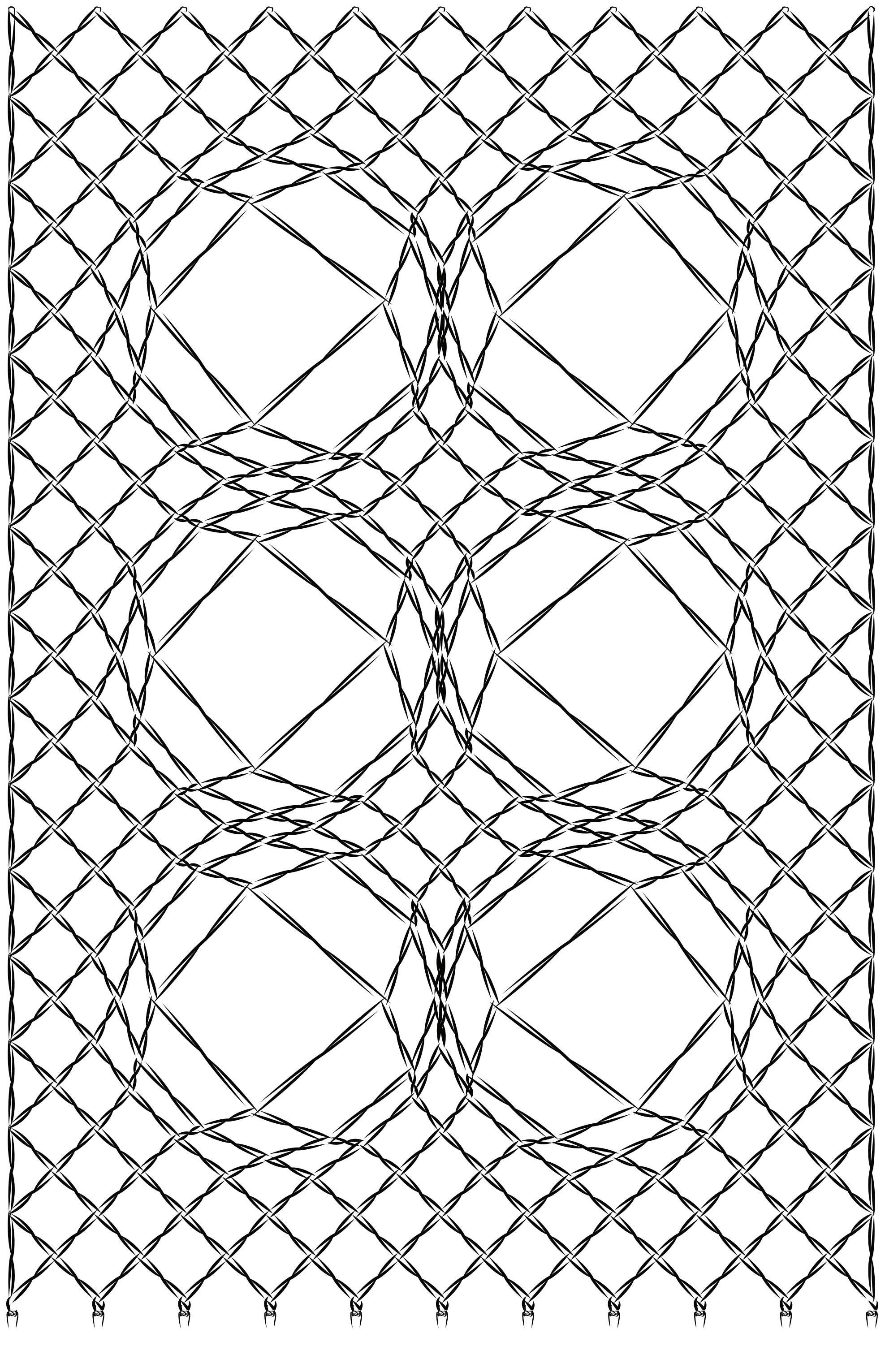


***Figure 3.24:** A lace with an effort score of* ***17999**.*

3.4 (c) Similar Designs with Different Effortt

Another means by which to explore the design space is to find visually similar designs that require different effort aside from how the lace is worked. Here, three patterns share a similar clustering design. Each instance of the design requires different effort.

Different Effort

Figure 3.25

Figure 3.26

Figure 3.27

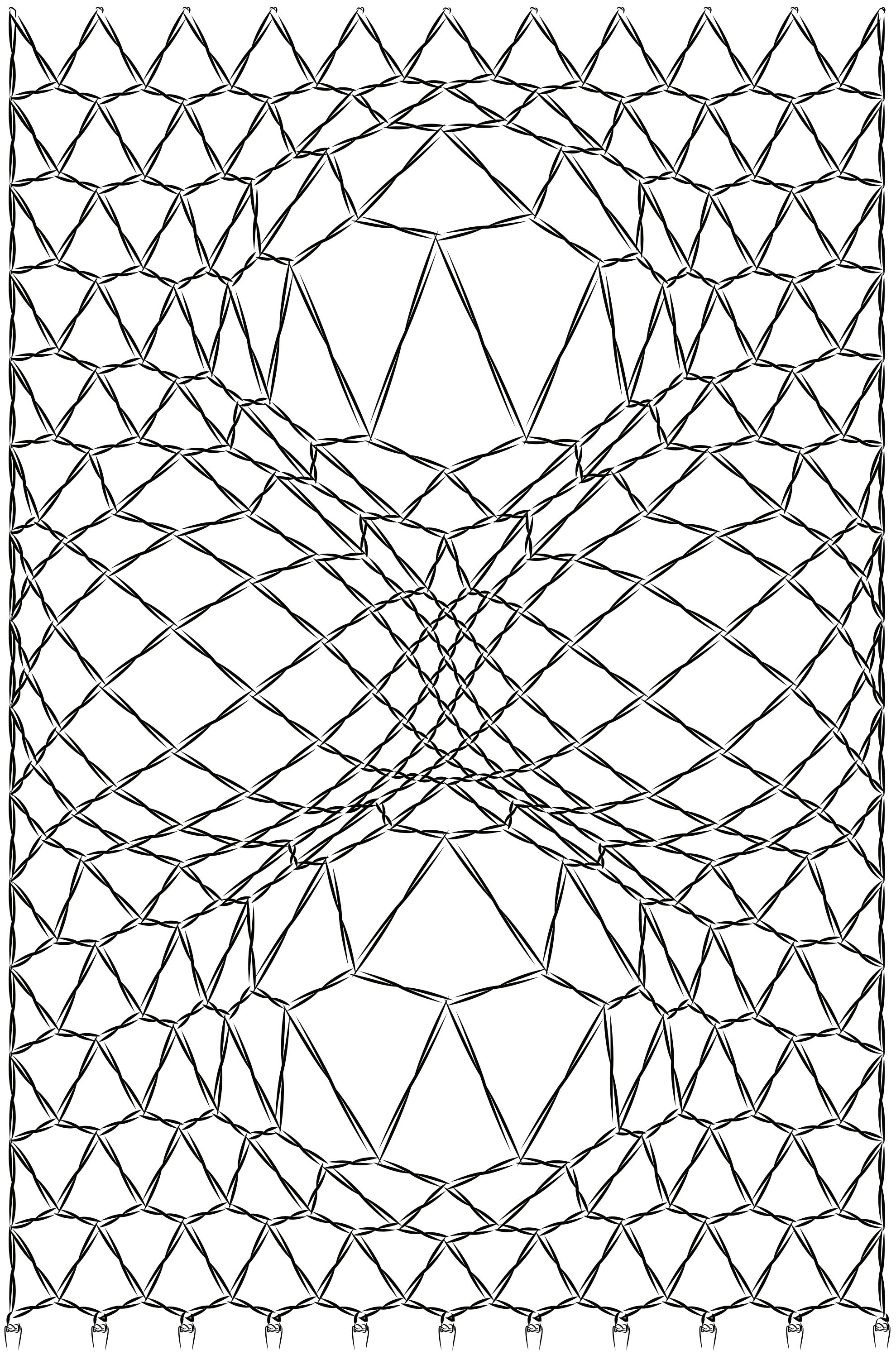


Figure 3.25: A lace with an effort score of ***28801****.*

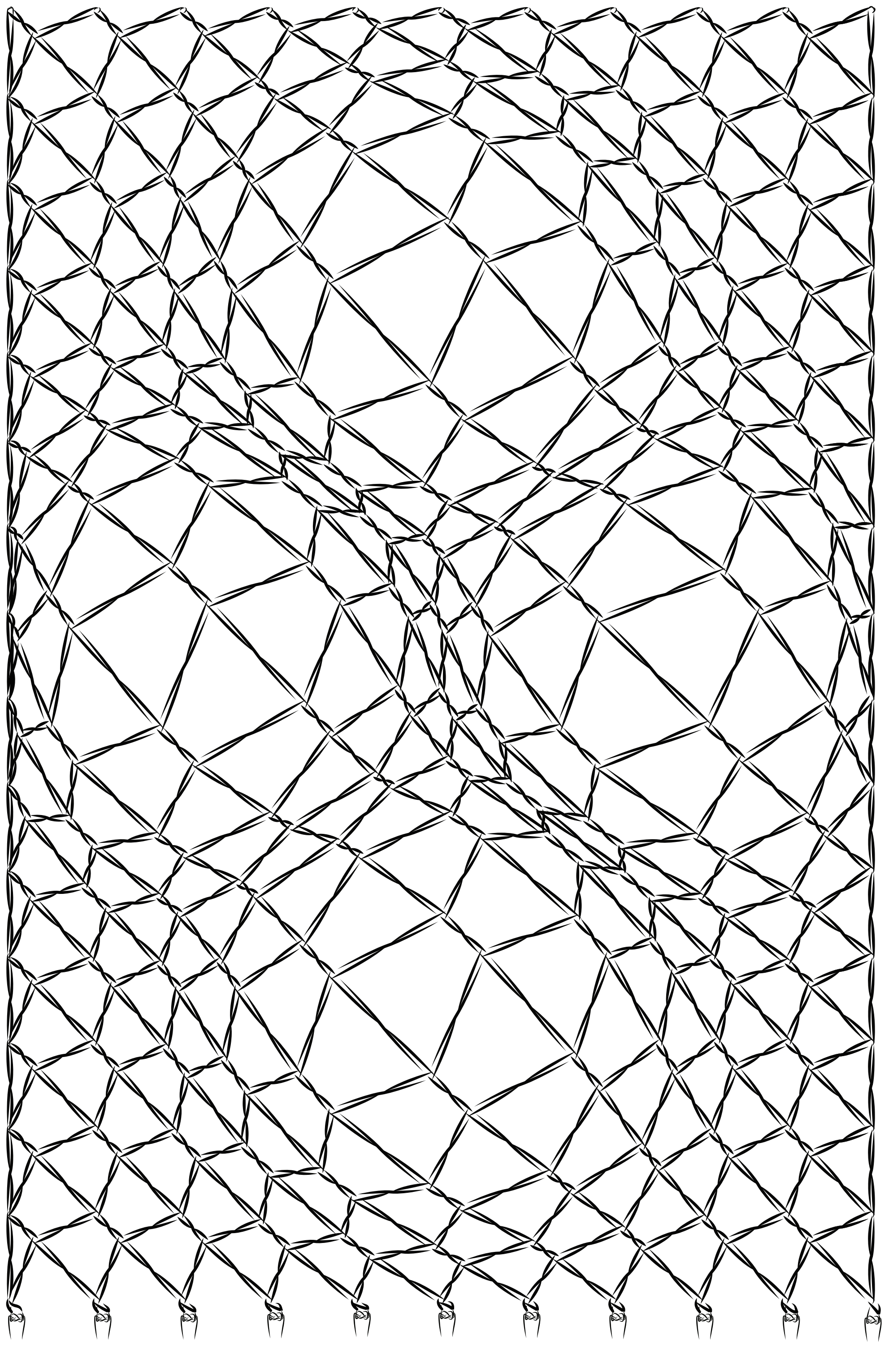


*Figure 3.26: A lace with an effort score of **17418**.*

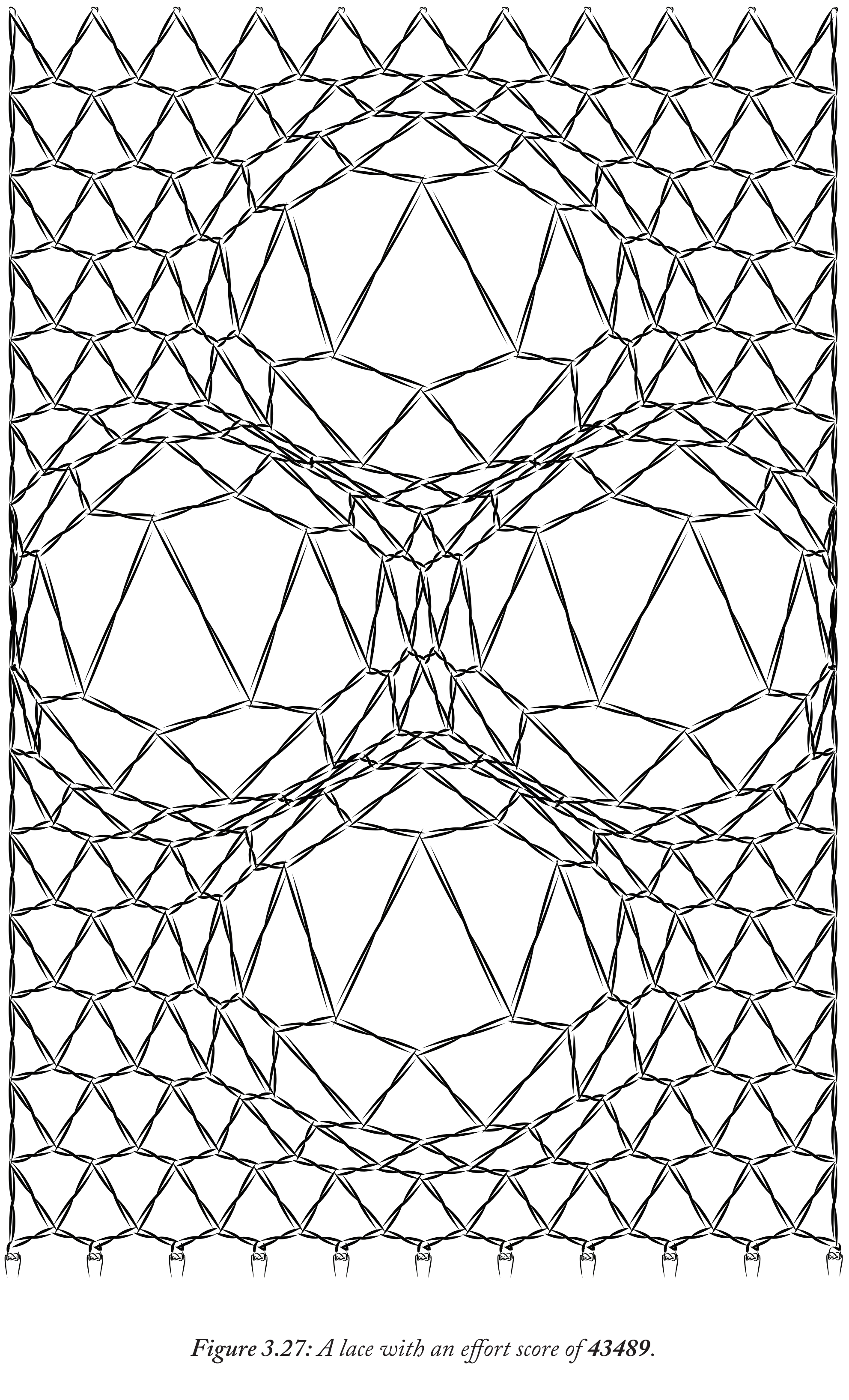


*Figure 3.27: A lace with an effort score of **43489**.*

PART IV

Discussion

4.1 Contributions & Future Work

Using bobbin lace as a case study, I expanded the formalism of making grammars to include an effort-cost tabulation that corresponds to moments of making. I developed a making grammar for bobbin lace at two resolutions: a high-resolution grammar of a cloth-stich lace to elucidate the steps, both physical and cognitive, required to make a design and a low-resolution grammar to generalize the steps for any torchon lace topology. These effort grammars provided the underpinning logic and flow of a computer program that can simulate and automate the production of lace while ensuring that designs generated are valid and makeable. With effort-costs assigned to each moment of making, the design space can be explored through visual inspection relative to embodied effort. An initial exploration of the design space was given and designs were grouped according to effort equivalence classes. I showed that identical lace designs can be made in different ways, and that there is an effort-cost associated with different ways of making. I found designs in the design space that were effort-equivalent but visually distinct as well as designs that were visually similar but required different expenditures of effort.

There is a richer design space yet to be explored, as the scope of this thesis was limited to a study of torchon ground in the production of designs. Additional grounds can be worked with the same ground embedding and would share an identical directed graph representation of torchon ground as, for example, Gravenmoer ground mentioned in Section 3.1(b). Interaction sequences for different grounds would contribute different levels of effort to a design and add to the possible visual complexity. Making rules for motifs such as spiders, triangles, and zigzags are needed to explore the contributions of effort in traditional lace design. A further broadening of possible designs includes addressing other ground embedding representations, such as those discovered by Veronika Irvine and Frank Ruskey.

Effort grammars may be expanded to other craft techniques in which a relatively simple set of rules produces a generous variety of designs. Further work is needed to broaden effort grammars to larger, more complex design processes.

4.2 Concluding Remarks

While the scope of this thesis has been limited to modelling and calculating effort in the production of bobbin lace, a broader consideration of this thesis is to introduce a framework for understanding the landscape of designs that is possible from certain expenditures of effort. In connecting expenditures of effort with the grammar formalism, an open but makeable design space is formulated that can be explored relative to the cost of production. This approach is a reversal of a typical design process in which form is retroactively analyzed for constructability, and frequently changed accordingly. At the same time, designers across many disciplines are under increased pressure from the climactic, social, and economic demands of this century to "perform" to quantitative metrics. Such demands require symbolic representation for numerical analysis and may, through established workflows and software, limit or exclude improvisational creative processes. Rather than addressing the constraints at the end of design, effort grammars proposes a direct engagement with them at the outset. Effort grammar computations are, like shape and making grammars, memoryless and visual. They encode an emergent design with knowledge about how to make it and yet avoid a reductive synthesis of predefined modules.

In fact, effort grammars *are* modular to the extent that they are deployed in context. As the computations in this thesis have shown, the interaction nodes are in a felicitous state to be worked when the corresponding nodes sharing edges in the directed graph representation have already been worked. That is, each node in the directed graph represents a module of effort, but the application of the effort leads to a non-deterministic and visual change of state for the design. While the instances of effort are modular, laying the foundation precedes erecting the walls, the visual change of state is not predetermined. It is therefore possible to propagate through the computation towards makeable designs without sacrificing the improvisational nature of design itself.

At the beginning of this thesis, I introduced the term *embodied effort* as the summation of work, steps, routines, applied skill, cognitive processing, or other forms of output to broadly capture the human or machine processes that directly contribute to the physical production of a design. Irrespective of the means of production, effort is a limited resource and subject to minimization by less than imaginative mechanisms; standardization and mass production decrease cost at a cost. To use economist Herbert Simon's definition of design as "courses of action aimed at changing existing situations into preferred ones,"(1969) there is an opportunity for reconciliation with the economic forces that can lead to predictable outcomes and instead engage the imagination to create designs that do more than solve a problem. The grammar formalism keeps this design space open, while computing effort grounds this search in the context of our current global crises.

In an essay first published in the *T-Square Club Journal* in 1931, architect Louis Kahn distinguished the measurable aspects of design from the "unmeasurable" spirit of a building: "A great building must, in my opinion, begin with the unmeasurable, must go through the measurable in the process of design, but must again in the end be unmeasurable" (1931). He also speculated that "the capacity to see comes from persistently analyzing our reactions to what we look at" (1931). From the computational perspective of this thesis, these claims proved prescient.

Bibliography

Agarwal, Manish, Jonathan Cagan, and Katherine G Constantine. 1999. “Influencing Generative Design through Continuous Evaluation: Associating Costs with the Coffeemaker Shape Grammar.” *AI EDAM* 13 (4): 253–275.

Agarwal, Manish, Jonathan Cagan, and George Stiny. 2000. “A Micro Language: Generating MEMS Resonators by Using a Coupled Form—Function Shape Grammar.” *Environment and Planning B: Planning and Design* 27 (4): 615–626.

Cagan, J, and WJ Mitchell. 1993. “Optimally Directed Shape Generation by Shape Annealing.” *Environment and Planning B: Planning and Design* 20 (1): 5–12.

Charidis, Alexandros. 2017. “Improvisational Specification of Design Spaces.” SM Thesis, Massachusetts Institute of Technology.

Conner, Brett P., Guha P. Manogharan, Ashley N. Martof, Lauren M. Rodomsky, Caitlyn M. Rodomsky, Dakesha C. Jordan, and James W. Limperos. 2014. “Making Sense of 3-D Printing: Creating a Map of Additive Manufacturing Products and Services.” *Additive Manufacturing*, Inaugural Issue, 1–4 (October): 64–76. https://doi.org/10.1016/j.addma.2014.08.005.

Cook, Bridget M., and Geraldine Stott. 2002. *The Book of Bobbin Lace Stitches*. Courier Corporation.

Dillmont, T de. 1886. “Encyclopedia of Needlework, Original Title: Encyclopédie Des Ouvrages Des Dames.”

Edkins, Jo. 2008a. "Making Lace." 2008. http://www.gwydir.demon.co.uk/jo/laceold/make.htm#bobbins.

———. 2008b. "Starting and Finishing Lace." 2008. http://www.gwydir.demon.co.uk/jo/laceold/start.htm.

———. 2010. "Variable Grids." 2010. http://www.gwydir.demon.co.uk/jo/laceold/var.htm.

———. 2014. "Tightening Thread." 2014. http://www.gwydir.demon.co.uk/jo/laceold/tighten.htm.

———. 2016a. "Starting Lace." 2016. http://www.gwydir.demon.co.uk/jo/lace/eqstart.htm.

———. 2016b. "Torchon Ground." 2016. http://www.gwydir.demon.co.uk/jo/lace/grtorchon.htm.

———. 2016c. "Winding Bobbins." Jo Edkins' Bobbin Lace School. 2016. http://www.gwydir.demon.co.uk/jo/lace/eqwind.htm.

———. 2016d. "Working Lace." Jo Edkins' Bobbin Lace School. 2016. http://www.gwydir.demon.co.uk/jo/lace/eqwork.htm.

———. 2017. "Jo Edkins' Bobbin Lace School." Jo Edkins' Bobbin Lace School. 2017. http://www.theedkins.co.uk/jo/lace/.

Elberfeld, Nathaniel, Lavender Tessmer, and Jason Butz. 2017. *Hedge*. Resin-harded, braided carbon fiber panels, CNC cut plastic of varying types.

Elberfeld, Nathaniel, Lavender Tessmer, and Alexandra Waller. 2018. *Concrete Tapestry*. Concrete, carbon fiber.

Grishanov, Sergei, Vadim Meshkov, and Alexander Omelchenko. 2009. "A Topological Study of Textile Structures. Part I: An Introduction to Topological Methods." *Textile Research Journal* 79 (8): 702–713.

Halley, Lorelei. 2009a. "Bobbin Lace History - Overview." 2009. http://lynxlace.com/bobbinlacehistoryoverview.html.

———. 2009b. "Two Structural Classes of Bobbin Lace." 2009. https://www.lynxlace.com/bobbinlace2structuralclasses.html.

Irvine, Veronika, and Frank Ruskey. 2014. "Developing a Mathematical Model for Bobbin Lace." *Journal of Mathematics and the Arts* 8 (3–4): 95–110. https://doi.org/10.1080/17513472.2014.982938.

"La Pompe, Opera Nova Nella Quale Si Retrovano Varie..." 1559. http://visualiseur.bnf.fr/CadresFenetre?O=IFN-8622058&I=34&M=notice.

Kahn, Louis I. 1931. *The Value and Aim in Sketching*.

Knight, Terry. 1989. "Color Grammars: Designing with Lines and Colors." *Environment and Planning B: Planning and Design* 16 (4): 417–449.

———. 2015. "Shapes and Other Things." *Nexus Network Journal* 17 (3): 963–980. https://doi.org/10.1007/s00004-015-0267-3.

Knight, Terry, and George Stiny. 2015. "Making Grammars: From Computing with Shapes to Computing with Things." *Design Studies* 41: 8–28.

Königseder, Corinna, Tino Stanković, and Kristina Shea. 2016. "Improving Design Grammar Development and Application through Network-Based Analysis of Transition Graphs." *Design Science* 2.

Leader, Jean. 2019. "The Lace Guild and Museum." TheLaceGuild. Updated 2019. https://www.laceguild.org.

Lee, Junbok, Young-Jin Park, Chang-Hoon Choi, and Choong-Hee Han. 2017. "BIM-Assisted Labor Productivity Measurement Method for Structural Formwork." *Automation in Construction* 84: 121–32. https://doi.org/10.1016/j.autcon.2017.08.009.

McKnelly, Carrie Lee. 2015. "Knitting Behavior: A Material-Centric Design Process." SM Thesis, Massachusetts Institute of Technology.

Mitchell, William J. 1991. "Functional Grammars: An Introduction."

Mueller, Caitlin T. 2014. "Computational Exploration of the Structural Design Space." PhD Thesis, Massachusetts Institute of Technology.

Muslimin, Rizal. 2014. "EthnoComputation: On Weaving Grammars for Architectural Design." PhD Thesis, Massachusetts Institute of Technology.

Nykamp, Duane Q. 2020. "Math Insight." Accessed 2020. https://mathinsight.org/definition/directed_graph.

Ramos, Diana. 2017. "Construction Cost Estimating: The Basics and Beyond." May 26, 2017. https://www.smartsheet.com/construction-cost-estimating.

Semper, Gottfried. 1851. *The Four Elements of Architecture, Trans. Harry Francis Mallgrave and Wolfgang Hermann*. Cambridge: Cambridge University Press.

Simon, Herbert A. 1969. "The Sciences of the Artificial." *Cambridge, MA*.

Stiny, George. 1981. "A Note on the Description of Designs." *Environment and Planning B: Planning and Design* 8 (3): 257–267.

———. 1991. "The Algebras of Design." *Research in Engineering Design* 2 (3): 171–181.

Ulrich, Uta. 2009. *Gründe Mit System*. Barbara Fay Verlag.

Wallner, Johannes, and Helmut Pottmann. 2011. "Geometric Computing for Freeform Architecture." *Journal of Mathematics in Industry* 1 (1): 4.

Woodbury, Robert F, and Andrew L Burrow. 2006. "Whither Design Space?" *Ai Edam* 20 (2): 63–82.

Appendix

Rule Applied	***Number of Instances***
Rule 1	1
Rule 2	10
Rule 3	10
Rule 4	5
Rule 5	5
Rule 6	48
Rule 7	22
Rule 8	12
Rule 9A	28
Rule 9B	28
Rule 10	5
Rule 11	1
Rule 12	1

Table A.1: *Effort tabulation of example high-resolution making grammar computation.*

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[5,0]	0, 0, 0, 0	20, 21, 22, 23	Different	Rule 2	2	2	1
[4,0]	0, 0, 0, 0	16, 17, 18, 19	Different	Rule 2	2	2	1
[3,0]	0, 0, 0, 0	12, 13, 14, 15	Different	Rule 2	2	2	1
[2,0]	0, 0, 0, 0	8, 9, 10, 11	Different	Rule 2	2	2	1
[1,0]	0, 0, 0, 0	4, 5, 6, 7	Different	Rule 2	2	2	1
[0,0]	0, 0, 0, 0	0, 1, 2, 3	Different	Rule 2	2	2	1
[0,1]	2, 3, 4, 5	5, 4, 3, 2	Shared	Rule 3	2	2	1
[1,1]	6, 7, 8, 9	9, 8, 7, 6	Different	Rule 4	2	2	1
[2,1]	10, 11, 12, 13	13, 12, 11, 10	Different	Rule 4	2	2	1
[3,1]	14, 15, 16, 17	17, 16, 15, 14	Different	Rule 4	2	2	1
[4,1]	18, 19, 20, 21	21, 20, 19, 18	Different	Rule 4	2	2	1
[5,2]	19, 18, 22, 23	23, 22, 18, 19	Shared	Rule 10	2	2	1
[4,2]	15, 14, 21, 20	20, 21, 14, 15	Different	Rule 5	2	2	1
[3,2]	11, 10, 17, 16	16, 17, 10, 11	Different	Rule 5	2	2	1
[2,2]	7, 6, 13, 12	12, 13, 6, 7	Different	Rule 5	2	2	1
[1,2]	3, 2, 9, 8	8, 9, 2, 3	Different	Rule 5	2	2	1
[0,2]	0, 1, 5, 4	4, 5, 1, 0	Different	Rule 10	2	2	1
[0,3]	1, 0, 8, 9	9, 8, 0, 1	Shared	Rule 5	2	2	1
[1,3]	2, 3, 12, 13	13, 12, 3, 2	Different	Rule 5	2	2	1
[2,3]	6, 7, 16, 17	17, 16, 7, 6	Different	Rule 5	2	2	1
[3,3]	10, 11, 20, 21	21, 20, 11, 10	Different	Rule 5	2	2	1
[4,3]	14, 15, 23, 22	22, 23, 15, 14	Different	Rule 5	2	2	1
[5,4]	15, 14, 18, 19	19, 18, 14, 15	Shared	Rule 10	2	2	1
[4,4]	11, 10, 22, 23	23, 22, 10, 11	Different	Rule 5	2	2	1
[3,4]	7, 6, 21, 20	20, 21, 6, 7	Different	Rule 5	2	2	1
[2,4]	3, 2, 17, 16	16, 17, 2, 3	Different	Rule 5	2	2	1
[1,4]	0, 1, 13, 12	12, 13, 1, 0	Different	Rule 5	2	2	1
[0,4]	4, 5, 9, 8	8, 9, 5, 4	Different	Rule 10	2	2	1
[0,5]	5, 4, 12, 13	13, 12, 4, 5	Shared	Rule 5	2	2	1
[1,5]	1, 0, 16, 17	17, 16, 0, 1	Different	Rule 5	2	2	1
[2,5]	2, 3, 20, 21	21, 20, 3, 2	Different	Rule 5	2	2	1
[3,5]	6, 7, 23, 22	22, 23, 7, 6	Different	Rule 5	2	2	1

Table A.2: *Complete effort-tabulation for a lace worked in back-and-forth rows.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	4	2	1	6	2	20	18	11	57	57
1	4	2	1	6	2	20	18	11	57	114
1	4	2	1	26	2	20	18	11	77	191
1	4	2	1	26	2	20	18	11	77	268
1	4	2	1	26	2	20	18	11	77	345
1	4	2	4	113	2	20	18	11	164	509
1	2	0	1	16	2	0	18	1	37	546
1	4	0	1	26	2	0	18	1	47	593
1	4	0	1	26	2	0	18	1	47	640
1	4	0	1	26	2	0	18	1	47	687
1	4	0	1	26	2	0	18	1	47	734
1	2	0	0	12	2	0	18	1	33	767
1	4	0	1	32	2	0	18	1	53	820
1	4	0	3	81	2	0	18	1	102	922
1	4	0	3	80	2	0	18	1	101	1023
1	4	0	1	32	2	0	18	1	53	1076
1	4	0	0	19	2	0	18	1	40	1116
1	2	0	0	14	2	0	18	1	35	1151
1	4	0	2	53	2	0	18	1	74	1225
1	4	0	13	329	2	0	18	1	350	1575
1	4	0	2	50	2	0	18	1	71	1646
1	4	0	0	23	2	0	18	1	44	1690
1	2	0	0	11	2	0	18	1	32	1722
1	4	0	1	29	2	0	18	1	50	1772
1	4	0	3	65	2	0	18	1	86	1858
1	4	0	2	64	2	0	18	1	85	1943
1	4	0	1	31	2	0	18	1	52	1995
1	4	0	0	18	2	0	18	1	39	2034
1	2	0	0	14	2	0	18	1	35	2069
1	4	0	1	34	2	0	18	1	55	2124
1	4	0	2	48	2	0	18	1	69	2192
1	4	0	1	32	2	0	18	1	53	2245

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[4,5]	10, 11, 19, 18	18, 19, 11, 10	Different	Rule 5	2	2	1
[5,6]	11, 10, 14, 15	15, 14, 10, 11	Shared	Rule 10	2	2	1
[4,6]	7, 6, 18, 19	19, 18, 6, 7	Different	Rule 5	2	2	1
[3,6]	3, 2, 22, 23	23, 22, 2, 3	Different	Rule 5	2	2	1
[2,6]	0, 1, 21, 20	20, 21, 1, 0	Different	Rule 5	2	2	1
[1,6]	4, 5, 17, 16	16, 17, 5, 4	Different	Rule 5	2	2	1
[0,6]	8, 9, 13, 12	12, 13, 9, 8	Different	Rule 10	2	2	1
[0,7]	9, 8, 16, 17	17, 16, 8, 9	Shared	Rule 5	2	2	1
[1,7]	5, 4, 20, 21	21, 20, 4, 5	Different	Rule 5	2	2	1
[2,7]	1, 0, 23, 22	22, 23, 0, 1	Different	Rule 5	2	2	1
[3,7]	2, 3, 19, 18	18, 19, 3, 2	Different	Rule 5	2	2	1
[4,7]	6, 7, 15, 14	14, 15, 7, 6	Different	Rule 5	2	2	1
[5,8]	7, 6, 10, 11	11, 10, 6, 7	Shared	Rule 10	2	2	1
[4,8]	3, 2, 14, 15	15, 14, 2, 3	Different	Rule 5	2	2	1
[3,8]	0, 1, 18, 19	19, 18, 1, 0	Different	Rule 5	2	2	1
[2,8]	4, 5, 22, 23	23, 22, 5, 4	Different	Rule 5	2	2	1
[1,8]	8, 9, 21, 20	20, 21, 9, 8	Different	Rule 5	2	2	1
[0,8]	12, 13, 17, 16	16, 17, 13, 12	Different	Rule 10	2	2	1
[0,9]	13, 12, 20, 21	21, 20, 12, 13	Shared	Rule 5	2	2	1
[1,9]	9, 8, 23, 22	22, 23, 8, 9	Different	Rule 5	2	2	1
[2,9]	5, 4, 19, 18	18, 19, 4, 5	Different	Rule 5	2	2	1
[3,9]	1, 0, 15, 14	14, 15, 0, 1	Different	Rule 5	2	2	1
[4,9]	2, 3, 11, 10	10, 11, 3, 2	Different	Rule 5	2	2	1
[5,10]	3, 2, 6, 7	7, 6, 2, 3	Shared	Rule 10	2	2	1
[4,10]	0, 1, 10, 11	11, 10, 1, 0	Different	Rule 5	2	2	1
[3,10]	4, 5, 14, 15	15, 14, 5, 4	Different	Rule 5	2	2	1
[2,10]	8, 9, 18, 19	19, 18, 9, 8	Different	Rule 5	2	2	1
[1,10]	12, 13, 22, 23	23, 22, 13, 12	Different	Rule 5	2	2	1
[0,10]	16, 17, 21, 20	20, 21, 17, 16	Different	Rule 10	2	2	1
[0,11]	17, 16, 23, 22	22, 23, 16, 17	Shared	Rule 5	2	2	1
[1,11]	13, 12, 19, 18	18, 19, 12, 13	Different	Rule 5	2	2	1
[2,11]	9, 8, 15, 14	14, 15, 8, 9	Different	Rule 5	2	2	1

Table A.2(continued): *Complete effort-tabulation for a lace worked in back-and-forth rows.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	4	0	0	22	2	0	18	1	43	2289
1	2	0	0	13	2	0	18	1	34	2322
1	4	0	0	24	2	0	18	1	45	2367
1	4	0	1	31	2	0	18	1	52	2419
1	4	0	1	32	2	0	18	1	53	2473
1	4	0	0	25	2	0	18	1	46	2518
1	4	0	0	21	2	0	18	1	42	2560
1	2	0	0	13	2	0	18	1	34	2594
1	4	0	0	25	2	0	18	1	46	2640
1	4	0	0	26	2	0	18	1	47	2687
1	4	0	0	25	2	0	18	1	46	2733
1	4	0	0	22	2	0	18	1	43	2775
1	2	0	0	16	2	0	18	1	37	2812
1	4	0	0	22	2	0	18	1	43	2855
1	4	0	0	21	2	0	18	1	42	2897
1	4	0	0	21	2	0	18	1	42	2939
1	4	0	0	22	2	0	18	1	43	2983
1	4	0	0	26	2	0	18	1	47	3029
1	2	0	0	15	2	0	18	1	36	3065
1	4	0	0	20	2	0	18	1	41	3106
1	4	0	0	20	2	0	18	1	41	3147
1	4	0	0	20	2	0	18	1	41	3188
1	4	0	0	26	2	0	18	1	47	3235
1	2	0	1	16	2	0	18	1	37	3272
1	4	0	0	23	2	0	18	1	44	3316
1	4	0	0	23	2	0	18	1	44	3360
1	4	0	0	22	2	0	18	1	43	3403
1	4	0	0	22	2	0	18	1	43	3446
1	4	0	0	26	2	0	18	1	47	3493
1	2	0	0	14	2	0	18	1	35	3529
1	4	0	0	25	2	0	18	1	46	3575
1	4	0	1	27	2	0	18	1	48	3623

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[3,11]	5, 4, 11, 10	10, 11, 4, 5	Different	Rule 5	2	2	1
[4,11]	1, 0, 7, 6	6, 7, 0, 1	Different	Rule 5	2	2	1
[5,12]	0, 1, 2, 3	3, 2, 1, 0	Shared	Rule 10	2	2	1
[4,12]	4, 5, 6, 7	7, 6, 5, 4	Different	Rule 5	2	2	1
[3,12]	8, 9, 10, 11	11, 10, 9, 8	Different	Rule 5	2	2	1
[2,12]	12, 13, 14, 15	15, 14, 13, 12	Different	Rule 5	2	2	1
[1,12]	16, 17, 18, 19	19, 18, 17, 16	Different	Rule 5	2	2	1
[0,12]	20, 21, 22, 23	23, 22, 21, 20	Different	Rule 10	2	2	1
[0,13]	21, 20, 19, 18	18, 19, 20, 21	Shared	Rule 5	2	2	1
[1,13]	17, 16, 15, 14	14, 15, 16, 17	Different	Rule 5	2	2	1
[2,13]	13, 12, 11, 10	10, 11, 12, 13	Different	Rule 5	2	2	1
[3,13]	9, 8, 7, 6	6, 7, 8, 9	Different	Rule 5	2	2	1
[4,13]	5, 4, 3, 2	2, 3, 4, 5	Different	Rule 5	2	2	1
[5,14]	4, 5, 1, 0	0, 1, 5, 4	Shared	Rule 10	2	2	1
[4,14]	8, 9, 2, 3	3, 2, 9, 8	Different	Rule 5	2	2	1
[3,14]	12, 13, 6, 7	7, 6, 13, 12	Different	Rule 5	2	2	1
[2,14]	16, 17, 10, 11	11, 10, 17, 16	Different	Rule 5	2	2	1
[1,14]	20, 21, 14, 15	15, 14, 21, 20	Different	Rule 5	2	2	1
[0,14]	23, 22, 18, 19	19, 18, 22, 23	Different	Rule 10	2	2	1
[0,15]	22, 23, 15, 14	14, 15, 23, 22	Shared	Rule 5	2	2	1
[1,15]	21, 20, 11, 10	10, 11, 20, 21	Different	Rule 5	2	2	1
[2,15]	17, 16, 7, 6	6, 7, 16, 17	Different	Rule 5	2	2	1
[3,15]	13, 12, 3, 2	2, 3, 12, 13	Different	Rule 5	2	2	1
[4,15]	9, 8, 0, 1	1, 0, 8, 9	Different	Rule 5	2	2	1
[5,16]	8, 9, 5, 4	4, 5, 9, 8	Shared	Rule 10	2	2	1
[4,16]	12, 13, 1, 0	0, 1, 13, 12	Different	Rule 5	2	2	1
[3,16]	16, 17, 2, 3	3, 2, 17, 16	Different	Rule 5	2	2	1
[2,16]	20, 21, 6, 7	7, 6, 21, 20	Different	Rule 5	2	2	1
[1,16]	23, 22, 10, 11	11, 10, 22, 23	Different	Rule 5	2	2	1
[0,16]	19, 18, 14, 15	15, 14, 18, 19	Different	Rule 10	2	2	1

Table A.2(continued): *Complete effort-tabulation for a lace worked in back-and-forth rows.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	4	0	0	26	2	0	18	1	47	3669
1	4	0	0	24	2	0	18	1	45	3714
1	2	0	0	13	2	0	18	1	34	3748
1	4	0	1	28	2	0	18	1	49	3797
1	4	0	1	34	2	0	18	1	55	3852
1	4	0	1	32	2	0	18	1	53	3905
1	4	0	1	29	2	0	18	1	50	3955
1	4	0	0	21	2	0	18	1	42	3996
1	2	0	0	15	2	0	18	1	36	4032
1	4	0	2	43	2	0	18	1	64	4096
1	4	0	2	46	2	0	18	1	67	4163
1	4	0	2	43	2	0	18	1	64	4227
1	4	0	0	24	2	0	18	1	45	4272
1	2	0	0	11	2	0	18	1	32	4304
1	4	0	1	36	2	0	18	1	57	4361
1	4	0	5	118	2	0	18	1	139	4500
1	4	0	4	112	2	0	18	1	133	4633
1	4	0	1	35	2	0	18	1	56	4689
1	4	0	0	18	2	0	18	1	39	4728
1	2	0	0	14	2	0	18	1	35	4763
1	4	0	2	41	2	0	18	1	62	4825
1	4	0	8	204	2	0	18	1	225	5050
1	4	0	2	41	2	0	18	1	62	5112
1	4	0	0	23	2	0	18	1	44	5156
1	2	0	0	12	2	0	18	1	33	5188
1	4	0	0	23	2	0	18	1	44	5232
1	4	0	1	32	2	0	18	1	53	5284
1	4	0	1	33	2	0	18	1	54	5338
1	4	0	0	23	2	0	18	1	44	5382
1	4	0	0	19	2	0	18	1	40	5422
									5422	

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[3,0]	0, 0, 0, 0	12, 13, 14, 15	Different	Rule 2	2	2	1
[0,0]	0, 0, 0, 0	0, 1, 2, 3	Different	Rule 2	2	2	1
[2,0]	0, 0, 0, 0	8, 9, 10, 11	Shared	Rule 2	2	2	1
[4,0]	0, 0, 0, 0	16, 17, 18, 19	Different	Rule 2	2	2	1
[1,0]	0, 0, 0, 0	4, 5, 6, 7	Different	Rule 2	2	2	1
[2,1]	10, 11, 12, 13	13, 12, 11, 10	Different	Rule 3	2	2	1
[5,0]	0, 0, 0, 0	20, 21, 22, 23	Different	Rule 2	2	2	1
[3,1]	14, 15, 16, 17	17, 16, 15, 14	Different	Rule 4	2	2	1
[0,1]	2, 3, 4, 5	5, 4, 3, 2	Different	Rule 3	2	2	1
[3,2]	11, 10, 17, 16	16, 17, 10, 11	Different	Rule 5	2	2	1
[1,1]	6, 7, 8, 9	9, 8, 7, 6	Different	Rule 4	2	2	1
[1,2]	3, 2, 9, 8	8, 9, 2, 3	Shared	Rule 5	2	2	1
[4,1]	18, 19, 20, 21	21, 20, 19, 18	Different	Rule 4	2	2	1
[4,2]	15, 14, 21, 20	20, 21, 14, 15	Shared	Rule 5	2	2	1
[5,2]	19, 18, 22, 23	23, 22, 18, 19	Different	Rule 10	2	2	1
[4,3]	14, 15, 23, 22	22, 23, 15, 14	Shared	Rule 5	2	2	1
[3,3]	10, 11, 20, 21	21, 20, 11, 10	Different	Rule 5	2	2	1
[2,2]	7, 6, 13, 12	12, 13, 6, 7	Different	Rule 5	2	2	1
[2,3]	6, 7, 16, 17	17, 16, 7, 6	Shared	Rule 5	2	2	1
[5,4]	15, 14, 18, 19	19, 18, 14, 15	Different	Rule 10	2	2	1
[0,2]	0, 1, 5, 4	4, 5, 1, 0	Different	Rule 10	2	2	1
[4,4]	11, 10, 22, 23	23, 22, 10, 11	Different	Rule 5	2	2	1
[0,3]	1, 0, 8, 9	9, 8, 0, 1	Different	Rule 5	2	2	1
[4,5]	10, 11, 19, 18	18, 19, 11, 10	Different	Rule 5	2	2	1
[3,4]	7, 6, 21, 20	20, 21, 6, 7	Different	Rule 5	2	2	1
[0,4]	4, 5, 9, 8	8, 9, 5, 4	Different	Rule 10	2	2	1
[1,3]	2, 3, 12, 13	13, 12, 3, 2	Different	Rule 5	2	2	1
[5,6]	11, 10, 14, 15	15, 14, 10, 11	Different	Rule 10	2	2	1
[2,4]	3, 2, 17, 16	16, 17, 2, 3	Different	Rule 5	2	2	1
[3,5]	6, 7, 23, 22	22, 23, 7, 6	Different	Rule 5	2	2	1
[4,6]	7, 6, 18, 19	19, 18, 6, 7	Shared	Rule 5	2	2	1
[4,7]	6, 7, 15, 14	14, 15, 7, 6	Shared	Rule 5	2	2	1

Table A.3: *Complete effort-tabulation for a lace worked at random nodes.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	8	2	1	6	2	20	18	11	57	57
1	8	2	4	26	2	20	18	11	77	134
1	2	2	1	16	2	20	18	11	67	201
1	8	2	1	46	2	20	18	11	97	298
1	8	2	1	46	2	20	18	11	97	395
1	8	0	1	46	2	0	18	1	67	462
1	8	2	1	46	2	20	18	11	97	559
1	8	0	1	46	2	0	18	1	67	626
1	8	0	1	46	2	0	18	1	67	693
1	8	0	3	124	2	0	18	1	145	838
1	8	0	1	46	2	0	18	1	67	905
1	2	0	1	19	2	0	18	1	40	945
1	8	0	1	46	2	0	18	1	67	1012
1	2	0	1	20	2	0	18	1	41	1053
1	8	0	0	35	2	0	18	1	56	1109
1	2	0	0	14	2	0	18	1	35	1144
1	8	0	2	73	2	0	18	1	94	1238
1	8	0	3	131	2	0	18	1	152	1390
1	2	0	6	100	2	0	18	1	121	1511
1	8	0	0	33	2	0	18	1	54	1566
1	8	0	0	35	2	0	18	1	56	1622
1	8	0	1	50	2	0	18	1	71	1692
1	8	0	0	41	2	0	18	1	62	1755
1	8	0	0	40	2	0	18	1	61	1816
1	8	0	2	92	2	0	18	1	113	1929
1	8	0	0	34	2	0	18	1	55	1984
1	8	0	2	76	2	0	18	1	97	2081
1	8	0	0	38	2	0	18	1	59	2141
1	8	0	2	93	2	0	18	1	114	2255
1	8	0	1	57	2	0	18	1	78	2332
1	2	0	0	16	2	0	18	1	37	2369
1	2	0	0	14	2	0	18	1	35	2404

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[5,8]	7, 6, 10, 11	11, 10, 6, 7	Shared	Rule 10	2	2	1
[1,4]	0, 1, 13, 12	12, 13, 1, 0	Different	Rule 5	2	2	1
[2,5]	2, 3, 20, 21	21, 20, 3, 2	Different	Rule 5	2	2	1
[0,5]	5, 4, 12, 13	13, 12, 4, 5	Different	Rule 5	2	2	1
[3,6]	3, 2, 22, 23	23, 22, 2, 3	Different	Rule 5	2	2	1
[0,6]	8, 9, 13, 12	12, 13, 9, 8	Different	Rule 10	2	2	1
[1,5]	1, 0, 16, 17	17, 16, 0, 1	Different	Rule 5	2	2	1
[2,6]	0, 1, 21, 20	20, 21, 1, 0	Shared	Rule 5	2	2	1
[2,7]	1, 0, 23, 22	22, 23, 0, 1	Shared	Rule 5	2	2	1
[3,7]	2, 3, 19, 18	18, 19, 3, 2	Different	Rule 5	2	2	1
[3,8]	0, 1, 18, 19	19, 18, 1, 0	Shared	Rule 5	2	2	1
[4,8]	3, 2, 14, 15	15, 14, 2, 3	Different	Rule 5	2	2	1
[1,6]	4, 5, 17, 16	16, 17, 5, 4	Different	Rule 5	2	2	1
[0,7]	9, 8, 16, 17	17, 16, 8, 9	Shared	Rule 5	2	2	1
[4,9]	2, 3, 11, 10	10, 11, 3, 2	Different	Rule 5	2	2	1
[5,10]	3, 2, 6, 7	7, 6, 2, 3	Shared	Rule 10	2	2	1
[0,8]	12, 13, 17, 16	16, 17, 13, 12	Different	Rule 10	2	2	1
[1,7]	5, 4, 20, 21	21, 20, 4, 5	Different	Rule 5	2	2	1
[2,8]	4, 5, 22, 23	23, 22, 5, 4	Shared	Rule 5	2	2	1
[1,8]	8, 9, 21, 20	20, 21, 9, 8	Different	Rule 5	2	2	1
[0,9]	13, 12, 20, 21	21, 20, 12, 13	Shared	Rule 5	2	2	1
[2,9]	5, 4, 19, 18	18, 19, 4, 5	Different	Rule 5	2	2	1
[0,10]	16, 17, 21, 20	20, 21, 17, 16	Different	Rule 10	2	2	1
[1,9]	9, 8, 23, 22	22, 23, 8, 9	Different	Rule 5	2	2	1
[1,10]	12, 13, 22, 23	23, 22, 13, 12	Shared	Rule 5	2	2	1
[0,11]	17, 16, 23, 22	22, 23, 16, 17	Shared	Rule 5	2	2	1
[2,10]	8, 9, 18, 19	19, 18, 9, 8	Different	Rule 5	2	2	1
[0,12]	20, 21, 22, 23	23, 22, 21, 20	Different	Rule 10	2	2	1
[1,11]	13, 12, 19, 18	18, 19, 12, 13	Different	Rule 5	2	2	1
[1,12]	16, 17, 18, 19	19, 18, 17, 16	Shared	Rule 5	2	2	1
[0,13]	21, 20, 19, 18	18, 19, 20, 21	Shared	Rule 5	2	2	1
[3,9]	1, 0, 15, 14	14, 15, 0, 1	Different	Rule 5	2	2	1

Table A.3(continued): *Complete effort-tabulation for a lace worked at random nodes.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	2	0	0	16	2	0	18	1	37	2441
1	8	0	1	50	2	0	18	1	71	2512
1	8	0	2	72	2	0	18	1	93	2605
1	8	0	0	41	2	0	18	1	62	2666
1	8	0	1	54	2	0	18	1	75	2742
1	8	0	0	39	2	0	18	1	60	2801
1	8	0	1	56	2	0	18	1	77	2879
1	2	0	1	19	2	0	18	1	40	2919
1	2	0	1	17	2	0	18	1	38	2956
1	8	0	0	44	2	0	18	1	65	3022
1	2	0	0	13	2	0	18	1	34	3056
1	8	0	0	41	2	0	18	1	62	3118
1	8	0	0	46	2	0	18	1	67	3184
1	2	0	0	14	2	0	18	1	35	3219
1	8	0	0	45	2	0	18	1	66	3286
1	2	0	1	16	2	0	18	1	37	3323
1	8	0	1	46	2	0	18	1	67	3390
1	8	0	0	45	2	0	18	1	66	3456
1	2	0	0	13	2	0	18	1	34	3490
1	8	0	0	41	2	0	18	1	62	3552
1	2	0	0	16	2	0	18	1	37	3589
1	8	0	0	37	2	0	18	1	58	3648
1	8	0	0	46	2	0	18	1	67	3714
1	8	0	0	37	2	0	18	1	58	3772
1	2	0	0	14	2	0	18	1	35	3808
1	2	0	0	14	2	0	18	1	35	3843
1	8	0	0	42	2	0	18	1	63	3906
1	8	0	0	38	2	0	18	1	59	3965
1	8	0	1	48	2	0	18	1	69	4034
1	2	0	1	18	2	0	18	1	39	4072
1	2	0	0	15	2	0	18	1	36	4108
1	8	0	0	37	2	0	18	1	58	4167

Node	ThreadsIN	ThreadsOUT	Pairs	Rule Applied	$C(s_i)$	$T(s_i)$	$p(s_i)$
[0,14]	23, 22, 18, 19	19, 18, 22, 23	Different	Rule 10	2	2	1
[4,10]	0, 1, 10, 11	11, 10, 1, 0	Different	Rule 5	2	2	1
[4,11]	1, 0, 7, 6	6, 7, 0, 1	Shared	Rule 5	2	2	1
[3,10]	4, 5, 14, 15	15, 14, 5, 4	Different	Rule 5	2	2	1
[2,11]	9, 8, 15, 14	14, 15, 8, 9	Shared	Rule 5	2	2	1
[5,12]	0, 1, 2, 3	3, 2, 1, 0	Different	Rule 10	2	2	1
[3,11]	5, 4, 11, 10	10, 11, 4, 5	Different	Rule 5	2	2	1
[4,12]	4, 5, 6, 7	7, 6, 5, 4	Shared	Rule 5	2	2	1
[4,13]	5, 4, 3, 2	2, 3, 4, 5	Shared	Rule 5	2	2	1
[3,12]	8, 9, 10, 11	11, 10, 9, 8	Different	Rule 5	2	2	1
[3,13]	9, 8, 7, 6	6, 7, 8, 9	Shared	Rule 5	2	2	1
[2,12]	12, 13, 14, 15	15, 14, 13, 12	Different	Rule 5	2	2	1
[1,13]	17, 16, 15, 14	14, 15, 16, 17	Shared	Rule 5	2	2	1
[4,14]	8, 9, 2, 3	3, 2, 9, 8	Different	Rule 5	2	2	1
[2,13]	13, 12, 11, 10	10, 11, 12, 13	Different	Rule 5	2	2	1
[3,14]	12, 13, 6, 7	7, 6, 13, 12	Shared	Rule 5	2	2	1
[1,14]	20, 21, 14, 15	15, 14, 21, 20	Different	Rule 5	2	2	1
[2,14]	16, 17, 10, 11	11, 10, 17, 16	Different	Rule 5	2	2	1
[5,14]	4, 5, 1, 0	0, 1, 5, 4	Different	Rule 10	2	2	1
[4,15]	9, 8, 0, 1	1, 0, 8, 9	Shared	Rule 5	2	2	1
[3,15]	13, 12, 3, 2	2, 3, 12, 13	Different	Rule 5	2	2	1
[2,15]	17, 16, 7, 6	6, 7, 16, 17	Different	Rule 5	2	2	1
[1,15]	21, 20, 11, 10	10, 11, 20, 21	Different	Rule 5	2	2	1
[0,15]	22, 23, 15, 14	14, 15, 23, 22	Different	Rule 5	2	2	1
[4,16]	12, 13, 1, 0	0, 1, 13, 12	Different	Rule 5	2	2	1
[1,16]	23, 22, 10, 11	11, 10, 22, 23	Different	Rule 5	2	2	1
[3,16]	16, 17, 2, 3	3, 2, 17, 16	Different	Rule 5	2	2	1
[2,16]	20, 21, 6, 7	7, 6, 21, 20	Different	Rule 5	2	2	1
[0,16]	19, 18, 14, 15	15, 14, 18, 19	Different	Rule 10	2	2	1

Table A.3(continued): *Complete effort-tabulation for a lace worked at random nodes.*

$m(s_i)$	$\delta(s_i)$	$b(l_i)$	$\rho(s_i)$	$\varepsilon_{\text{perceiving}}$	$\varepsilon_{\text{recalling}}$	$\varepsilon_{\text{preparing}}$	$\varepsilon_{\text{working}}$	$\varepsilon_{\text{finishing}}$	$\varepsilon_{\text{total}}$	$\varepsilon_{\text{cumulative}}$
1	8	0	0	34	2	0	18	1	55	4221
1	8	0	0	41	2	0	18	1	62	4284
1	2	0	0	15	2	0	18	1	36	4320
1	8	0	0	42	2	0	18	1	63	4382
1	2	0	1	17	2	0	18	1	38	4420
1	8	0	0	38	2	0	18	1	59	4479
1	8	0	1	47	2	0	18	1	68	4547
1	2	0	1	18	2	0	18	1	39	4586
1	2	0	0	15	2	0	18	1	36	4622
1	8	0	1	58	2	0	18	1	79	4701
1	2	0	2	25	2	0	18	1	46	4747
1	8	0	1	57	2	0	18	1	78	4826
1	2	0	2	24	2	0	18	1	45	4871
1	8	0	1	60	2	0	18	1	81	4952
1	8	0	2	78	2	0	18	1	99	5051
1	2	0	4	63	2	0	18	1	84	5135
1	8	0	1	59	2	0	18	1	80	5216
1	8	0	4	169	2	0	18	1	190	5405
1	8	0	0	33	2	0	18	1	54	5460
1	2	0	0	15	2	0	18	1	36	5495
1	8	0	1	65	2	0	18	1	86	5581
1	8	0	5	249	2	0	18	1	270	5850
1	8	0	1	64	2	0	18	1	85	5935
1	8	0	0	42	2	0	18	1	63	5998
1	8	0	0	41	2	0	18	1	62	6059
1	8	0	0	41	2	0	18	1	62	6121
1	8	0	1	51	2	0	18	1	72	6193
1	8	0	1	51	2	0	18	1	72	6266
1	8	0	0	35	2	0	18	1	56	6322
									6322	

```
//GENERATE TORCHON LACE DESIGNS AND COMPUTE EMBODIED EFFORT



//THANKS TO DIEGO PINOCHET AND LUKAS DEBIASI FOR HELP WITH CODE
//THANKS TO DANIEL SHIFFMAN FOR EXCELLENT ONLINE TUTORIALS
//https://www.youtube.com/channel/UCvjgXvBlbQiydffZU7m1_aw



//IMPORT LIBRARIES


import toxi.geom.*;
import toxi.physics2d.*;
import toxi.physics2d.behaviors.*;




//REFERENCE TO THE PHYSICS ENGINE
VerletPhysics2D physics;




/// LACE SIZE & IDENTIFICATION ///

int cols = 6; //11 x 32 full size
int rows = 18;
int cellsize = 60;

String laceID = str(cols)+str(rows);

float dia = 10;
```

```
/// EFFORT VARIABLES ///

int e_cross = 2;
int e_twist = 4;
int e_pin = 2;
int e_unpin = 1;
int e_tight = 4;
int e_perceive_pairs = 5;
int e_perceive_tight = 6;
int e_recall = 2;
int e_bobbin_prep = 10;
int e_bobbin_finish = 5;
float rho_i; // Density at stitch
int delta; // Coefficient of identification
int bli; // pairs of bobbins at stitch l in columns

int rp = 1;
int rr = 1;
int ra = 1;
int rw = 1;
int rf = 1;

float EFFORT, E_PERC, E_RECALL, E_PREP, E_WORK, E_FINISH;

/// SEARCH PROTOCOL ////

boolean raster = false;
boolean diagonal = true;
boolean random = false;
int rs = 1; // random seed value

IntList q  = new IntList();
int r;
int stitch_loc;
```

```
/// ATTRACTOR POINT ///
boolean Particle_lock =  false;
boolean lace = false;


Vec2D mousePos;

float even_i;
float odd_i;


/// COUNTING VALUES ///

int h = 0; //STITCH INDEX
//int counter  = 0;
int c; //COLOR RELATED



/// PRINTING AND OUTPUT ///

PrintWriter output;
import processing.svg.*;
import processing.pdf.*;
boolean pdf = false;

Table table;

boolean record;

/// STITCH SETUP FOR SEARCH

ArrayList<Float[][]> stitches = new ArrayList<Float[][]>();
ArrayList<PVector> stitchesMixed = new ArrayList<PVector>();
```

```
/// INDIVIDUAL THREAD IDENTIFICATION ///

StringList history_in = new StringList();
StringList history_out = new StringList();

/// LACE VERTEX INITIALIZATION /////


boolean initialize = false;
Points[][] p = new Points[cols][rows];

//ARRAY CONTAINING THE VERTEXPARTICLES
VerletParticle2D[][] v = new VerletParticle2D[cols][rows];

AttractionBehavior2D randAttractor;



void settings() {
  if (!pdf) {
    size((cols)*cellsize, rows*cellsize/2, P3D, "filename.svg");
  }
  if (pdf) {
    size((cols)*cellsize, rows*cellsize/2, PDF, "test.pdf");
  }
}
void setup() {

  physics = new VerletPhysics2D();
  physics.setDrag(0.05f);
  physics.setWorldBounds(new Rect(0, 0, width, height));
  randomSeed(rs);
```

```
///////////////////////////////////////////////
///////////////////////////////////////////////
//// INITIALIZE POINTS GRID ///////////////////

for (int i = 0; i < cols; i++) {
  for (int j = 0; j < rows; j++) {

    float i_space_even = cellsize/2+i*cellsize;
    float i_space_odd = cellsize+i*cellsize;
    float j_space = cellsize/2+cellsize/2*j;

    ///////////////////////////////////////


    even_i = (i_space_even);
    odd_i = (i_space_odd);

    float even_j =(j_space);
    float odd_j = (j_space);

    if (j%2 == 0) { //EVEN ROWS
      VerletParticle2D newParticle= addParticle(even_i, even_j);
      v[i][j] = newParticle;

      if (j==0) {
        v[i][j].lock();
      }
      if (j==rows-2) {
        v[i][j].lock();
      }
      if (i==0) {
        v[i][j].lock();
      }
      if (i==cols-1) {
          v[i][j].lock();
        }
      } else { //ODD ROWS
        VerletParticle2D newParticle= addParticle(odd_i, odd_j);
        v[i][j] = newParticle;
      }
    }
  }

  //////SPRING CONNECTIONS FROM EVEN ROWS //////////////
  for (int i = 0; i < cols; i++) { //for all columns
    for (int j = 0; j <rows-2; j++) { //for all rows
      if (j%2 == 0) { // if the row index is even

        if ( i > cols/6-1 && i < cols-cols/5 ) {

          if (i < cols - 1  ) {
            VerletSpring2D springdiag1=new VerletSpring2D(v[i][j], 
v[i][j+1], 1, 0.001); //line \
            physics.addSpring(springdiag1);
          }

          if (i >0 ) {
            // if not the first column
            VerletSpring2D springdiag2=new VerletSpring2D(v[i][j], 
v[i-1][j+1], 1, 0.001); // line /
            physics.addSpring(springdiag2);
          }
        } else {
          if (i< cols - 1) {
            VerletSpring2D springdiag1=new VerletSpring2D(v[i][j], 
v[i][j+1], 1, 0.001); //line \
            physics.addSpring(springdiag1);
          }
          if (i >0 ) {
            // if not the first column
            VerletSpring2D springdiag2=new VerletSpring2D(v[i][j],
v[i-1][j+1], 1, 0.001); // line /
            physics.addSpring(springdiag2);
          }
        }

        if (i == cols -1  ) {
          // if the last column
          VerletSpring2D springvert1=new VerletSpring2D(v[i][j], v[i]
[j+2], 1, 0.001); // line |
          physics.addSpring(springvert1);
        }
      }
    }
  }

  //////SPRING CONNECTIONS FROM ODD ROWS //////////////
  for (int i = 0; i < cols-1; i++) {
    for (int j = 0; j <rows-2; j++) {
      if (j%2 != 0 ) {

        VerletSpring2D springdiag5=new VerletSpring2D(v[i][j], v[i+1]
[j+1], 1, 0.001); // line \
        physics.addSpring(springdiag5);

        //if (j%2 != 0 ) {
        VerletSpring2D springdiag6=new VerletSpring2D(v[i][j], v[i]
[j+1], 1, 0.001); // line /
        physics.addSpring(springdiag6);
      }
    }
  }
  //UPDATE PHYSICS ENGINE
  physics.update();

  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  //////SET UP STITCHES /////////////////////////
  for (int i = 0; i < rows-1; i++) { //scan all rows
    if (i %2 == 0) {
      Float[][] temp_row = new Float[cols][]; //construct 2D float array
      for (int j = 0; j < cols; j++) { //scan all columns

        Float[] row_location = {float(j), float(i), 0.};
        temp_row[j] = row_location;
      }
      stitches.add(temp_row);
    } else {
      Float[][] temp_row = new Float[cols-1][]; //construct 2D float
array
      for (int j = 0; j < cols-1; j++) { //scan all columns

        Float[] row_location = {float(j), float(i), 0.};
        temp_row[j] = row_location;
      }
      stitches.add(temp_row);
    }
  }

  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  //////SET UP RASTER STITCH ORDER//////////////
  if (rasterScan()) {
    for (int i = 0; i < stitches.size(); i++ ) {
      if ( i % 2 == 0) {
        //println(stitches.get(i).getClass().getName());
        Float[][] stitch_row = stitches.get(i);
        Float[][] rev_stitch_row = (Float[][]) reverse(stitch_row);
        for ( Float[] vec_nr : rev_stitch_row) {
          PVector curr_vec = new PVector(vec_nr[0], vec_nr[1], vec_
nr[2]);
          stitchesMixed.add(curr_vec);
        }
      } else {
        Float[][] stitch_row = stitches.get(i);
        for ( Float[] vec_nr : stitch_row) {
          PVector curr_vec = new PVector(vec_nr[0], vec_nr[1], vec_
nr[2]);
          stitchesMixed.add(curr_vec);
        }
      }
    }
  }

  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  //////SET UP DIAGONAL STITCH ORDER////////////
  if (diagonalScan()) {
    int s = 0;
    int r = 0;
    int c = 0;
    int t = 1;
    for ( r = 0; r <= rows - 2; ) {
      //for ( c = 0; c <= cols; ) {
      if (r %2 == 0 && s == 0) { // even row, first column
        Float[][] stitch_row =  stitches.get(r);
        Float[] element = stitch_row[s];
        PVector curr_vec = new PVector(element[0], element[1],
element[2]);
        stitchesMixed.add(curr_vec);
        c = c + 1; // increment c by 1
        r = 0;     // return to first row
        s = c;     // set s = c;
      } else if ( s == cols && r == 0) { // when columns run out
        c = cols -1;
        r = 2*t;
        s = c;
        Float[][] stitch_row =  stitches.get(r);
        Float[] element = stitch_row[s];
        PVector curr_vec = new PVector(element[0], element[1],
element[2]);
        stitchesMixed.add(curr_vec);
        r++;
        s = s - 1;
        t++;
      } else if ( r == rows-2 && s != 0) { // when rows run out
        Float[][] stitch_row =  stitches.get(r);
        Float[] element = stitch_row[s];
        PVector curr_vec = new PVector(element[0], element[1],
element[2]);
        stitchesMixed.add(curr_vec);
        c = cols -1;
        r = 0;
        r = 2*t;
        s = c;
        t++;
      } else if (r == 0 && s > 0 && s < cols) { // first row, not first
column
        Float[][] stitch_row = stitches.get(r);
        Float[] element = stitch_row[s];
        PVector curr_vec = new PVector(element[0], element[1],
element[2]);
        stitchesMixed.add(curr_vec);
        r++;
        s = s-1;
      } else if (r %2 != 0 && s < cols && r < rows -2) { //odd row
        Float[][] stitch_row =  stitches.get(r);
        Float[] element = stitch_row[s];
        PVector curr_vec = new PVector(element[0], element[1], 
element[2]);
        stitchesMixed.add(curr_vec);
        r++;
      } else if (r %2 == 0 && s <= c  && s !=0 && r < rows -2 && r 
!=0) { //even row, not first row, not first column
        Float[][] stitch_row =  stitches.get(r);
        Float[] element = stitch_row[s];
        PVector curr_vec = new PVector(element[0], element[1], 
element[2]);
        stitchesMixed.add(curr_vec);
        r++;
        s = s-1;
      }
    }
  }
  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  //////SET UP RANDOM STITCH ORDER//////////////
  if (randomScan()) {
    for (int i = 0; i < stitches.size(); i++ ) {
      Float[][] stitch_row = stitches.get(i);
      for ( Float[] vec_nr : stitch_row) {
        PVector curr_vec = new PVector(vec_nr[0], vec_nr[1], vec_
nr[2]);
        stitchesMixed.add(curr_vec);
      }
    }
  }
  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  /////////// TABLE SETUP ////////////////////////
  table = new Table();
  table.addColumn("Node");
  table.addColumn("ThreadsIN");
  table.addColumn("ThreadsOUT");
  table.addColumn("Pairs");
  table.addColumn("Rule Applied");
  table.addColumn("C(S_i)");
  table.addColumn("T(S_i)");
  table.addColumn("p(S_i)");

  table.addColumn("blank"); // FOR TABLE SPANNING FULL SPREAD

  table.addColumn("m(S_i)");
  table.addColumn("delta(S_i)");
  table.addColumn("b(l_i)");
  table.addColumn("rho(S_i)");
  table.addColumn("E_PERC");
  table.addColumn("E_RECALL");
  table.addColumn("E_PREP");
  table.addColumn("E_WORK");
  table.addColumn("E_FINISH");
  table.addColumn("EFFORT");
}
```

```
void draw() {
  if (record) {
    beginRecord(SVG, "frame-####.svg");
  }

  background(255); // (RGB, 150, 35, 0)


  for (int i = 0; i < cols; i++) { //for all columns
    for (int j = 0; j <rows-2; j++) { //for all rows

      if (j%2 == 0) { // if the row index is even
        ellipse(v[i][j].x, v[i][j].y, 5, 5);
      }
    }
  }

  //////SPRING CONNECTIONS FROM ODD ROWS //////////////
  for (int i = 0; i < cols-1; i++) {
    for (int j = 0; j <rows-2; j++) {
      if (j%2 != 0 ) {
        ellipse(v[i][j].x, v[i][j].y, 5, 5);
      }
    }
  }

  //UPDATE PHYSICS IN DRAW ELSE THE SIMULATION WILL NOT RUN
  physics.update();
  if (Particle_lock==true) {
    lockall(v);
  }

  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  ///////////////////////////////////////////////
  //////BACK AND FORTH SCAN ////////////////////
  if (initialize) {

    for (int i = 0; i < cols; i++) {
      for (int j = 0; j<rows; j++) {
        ellipse(v[i][j].x, v[i][j].y, 50, 50);

        p[i][j] = new Points(v[i][j].x, v[i][j].y, dia);
        if (j == 0) {
          p[i][j].initialRow();
        }
      }
    }
    initialize = false;
  }
  if (lace) {
    TableRow newRow = table.addRow();
    String[] pos_in = new String [4];
    String[] pos_out = new String [4];
    String joinedPos_in;
    String joinedPos_out;

    PVector position = new PVector();
    if (randomScan()) {
      r = int(random(stitchesMixed.size()-1)); // select random
stitches
      position = stitchesMixed.get(r);
    }
    if (!randomScan()) {
      position = stitchesMixed.get(h); // do not change h
    }
    int posX = int(position.x);
    int posY = int(position.y);


    if (p[posX][posY].isLegal() &! p[posX][posY].isWorked()) {
      p[posX][posY].select();




      //////TRACKING THREADS //////////////////////
      if (posY == 0) { // INITIAL ROW
        p[posX][0].threadsInit(posX);
        p[posX][0].threadsOut();
      }

      if (posX>=0 && posX < cols-1 && posY > 0 && posY %2 != 0 ) { //
ODD ROWS
        arrayCopy(p[posX][posY-1].threadsOut(), 2, p[posX][posY].
threadsL(), 0, 2); //put threadsOut into threads L
        arrayCopy(p[posX+1][posY-1].threadsOut(), 0, p[posX][posY].
threadsR(), 0, 2); //put threadsOut into threads R
        arrayCopy(concat(p[posX][posY].threadsL(), p[posX][posY].
threadsR()), p[posX][posY].threadsIn()); //threads In = threads L + R
        p[posX][posY].threadsOut();
      }
      if (posX > 0 && posX < cols-1 && posY > 0 && posY %2 == 0 ) { //
EVEN ROWS ABOVE posY = 0

        arrayCopy(p[posX-1][posY-1].threadsOut(), 2, p[posX][posY].
threadsL(), 0, 2); //put threadsOut into threads L
        arrayCopy(p[posX][posY-1].threadsOut(), 0, p[posX][posY].
threadsR(), 0, 2); //put threadsOut into threads R
        arrayCopy(concat(p[posX][posY].threadsL(), p[posX][posY].
threadsR()), p[posX][posY].threadsIn()); //threads In = threads L + R
```

```
        p[posX][posY].threadsOut();
      }

      if (posX == 0 && posY >= 2 && posY%2 == 0) { //left edge
        //p[posX][posY].threadsIn();
        arrayCopy(p[posX][posY-2].threadsOut(), 0, p[posX][posY].
threadsL(), 0, 2); //put threadsOut into threads L
        arrayCopy(p[posX][posY-1].threadsOut(), 0, p[posX][posY].
threadsR(), 0, 2); //put threadsOut into threads R
        arrayCopy(concat(p[posX][posY].threadsL(), p[posX][posY].
threadsR()), p[posX][posY].threadsIn());
        p[posX][posY].threadsOut();
      }

      if (posX == cols-1 && posY>=2 && posY%2 == 0) { //right edge

        arrayCopy(p[posX-1][posY-1].threadsOut(), 2, p[posX][posY].
threadsL(), 0, 2); //put threadsOut into threads L
        arrayCopy(p[posX][posY-2].threadsOut(), 2, p[posX][posY].
threadsR(), 0, 2); //put threadsOut into threads R
        arrayCopy(concat(p[posX][posY].threadsL(), p[posX][posY].
threadsR()), p[posX][posY].threadsIn());
        p[posX][posY].threadsOut();
      }
    }
    if (h < (stitchesMixed.size()-1) && !randomScan()) {
      h++;
    }

    if (p[posX][posY].isLegal() &! p[posX][posY].isWorked()) {

      for (int i = 0; i < 4; i++) {
        pos_in[i] = str(p[posX][posY].threadsIn[i]);
        pos_out[i] = str(p[posX][posY].threadsOut[i]);
      }

      joinedPos_in = join(pos_in, ", ");
      joinedPos_out = join(pos_out, ", ");


      history_in.append(joinedPos_in);
      history_out.append(joinedPos_out);

      q.append(r);

      if (history_in.size() > 2) {

        if (random) {
          stitch_loc = q.size();
        } else if (!randomScan()) {
          stitch_loc = h;
        }
        String[] last_out = split(history_out.get(stitch_loc-2), ",
");
        String[] this_in = split(history_in.get(stitch_loc-1), ", ");

        newRow.setString("Pairs", "Different");
        delta = 8;

        for (int i = 0; i < 4; i++) {
          for (int j = 0; j < 4; j++) {
            if (int(last_out[i]) == int(this_in[j])) {
              newRow.setString("Pairs", "Shared");
              delta = 2;
            }
          }
        }
        newRow.setString("ThreadsIN", joinedPos_in);
```

```
    newRow.setString("ThreadsOUT", joinedPos_out);
    newRow.setString("Node", "["+str(posX)+","+str(posY)+"]");
    newRow.setInt("C(S_i)", 2);
    newRow.setInt("T(S_i)", 2);
    newRow.setInt("p(S_i)", 1);
    newRow.setInt("m(S_i)", 1);
    newRow.setInt("delta(S_i)", delta);
  }
}




///// DISPLAY TORCHON POINTS ////////////////////////
for (int i = 0; i < cols; i++) {
  for (int j = 0; j <rows-1; j++) {
    if (j%2 == 0) {
      p[i][j].display();
    }
  }
}
for (int i = 0; i < cols-1; i++) {
  for (int j = 0; j <rows-1; j++) {
    if (j%2 != 0) {
      p[i][j].display();
    }
  }
}


////// DISPLAY NEXT LEGAL STITCHES ///////////////////
for (int i = 0; i < cols-1; i++) {
  for (int j = 0; j <rows-1; j++) {
    if (j%2 != 0) { //ODD ROWS
```

```
          if (p[i][j].isWorked() && p[i+1][j].isWorked()) {
            p[i+1][j+1].nextLegal();
          }
        }
        if (j%2 == 0) { //EVEN ROWS
          if (p[i][j].isWorked() && p[i+1][j].isWorked()) {
            p[i][j+1].nextLegal();
          }
        }
      }
    }

    ///// EDGE STITCHES ///////////////////////////////////
    for (int i = 0; i < cols-1; i++) {
      for (int j = 2; j <rows-1; j++) {
        if (j%2 == 0) {
          if (p[0][j-2].isWorked() && p[0][j-1].isWorked()) {
            p[0][j].nextLegal();
          }
          if (p[cols-1][j-2].isWorked() && p[cols-2][j-1].isWorked())
{
            p[cols-1][j].nextLegal();
          }
        }
      }
    }

    //////DISPLAY CONNECTIONS FROM EVEN ROWS /////////////
    for (int i = 0; i < cols; i++) { //for all columns
      for (int j = 0; j <rows-2; j++) { //for all rows
        if (j%2 == 0) { // if the row index is even

          if (i< cols - 1 && p[i][j].isWorked() && p[i][j+1].
isWorked()) {
```

```
            // if not the last column and the vertex is worked and the
related vertex is worked in the next row
            line(p[i][j].coordx(), p[i][j].coordy(), p[i][j+1].
coordx(), p[i][j+1].coordy()); //line \
          }

          if (i >0 && p[i][j].isWorked() && p[i-1][j+1].isWorked()) {
// line \
            // if not the first column and the related vertex is worked
in the next row
            line(p[i][j].coordx(), p[i][j].coordy(), p[i-1][j+1].
coordx(), p[i-1][j+1].coordy() ); // line /
          }

          if (i == cols -1 && j < rows -1 && p[i][j].isWorked() &&
p[i-1][j+1].isWorked()) {
            // if the last column and the vertex is worked and the
related vertex is worked in the next row
            line(p[i][j].coordx(), p[i][j].coordy(), p[i-1][j+1].
coordx(), p[i-1][j+1].coordy() ); // line /
          }

          if (i == cols -1 && j < rows -1 && p[i][j].isWorked() &&
p[i][j+2].isWorked()) {
            // if the last column and the vertex is worked and the
related vertex is worked in the next row
            line(p[i][j].coordx(), p[i][j].coordy(), p[i][j+2].
coordx(), p[i][j+2].coordy() ); // line |
          }

          if (i == 0  && j < rows -1 && p[i][j].isWorked() && p[i]
[j+1].isWorked()) {
            // if the first column and the vertex is worked and the
related vertex is worked in the next row
```

```
            line(p[i][j].coordx(), p[i][j].coordy(), p[i][j+1].
coordx(), p[i][j+1].coordy() ); // line \
          }

          if (i == 0  && j < rows -1 && p[i][j].isWorked() && p[i]
[j+2].isWorked()) {
            // if the first column and the vertex is worked and the
related vertex is worked in the next row
            line(p[i][j].coordx(), p[i][j].coordy(), p[i][j+2].
coordx(), p[i][j+2].coordy() ); // line |
          }
        }
      }
    }
    //////DISPLAY CONNECTIONS FROM ODD ROWS //////////////
    for (int i = 0; i < cols-1; i++) {
      for (int j = 0; j <rows-2; j++) {
        if (j%2 != 0 && p[i][j].isWorked() && p[i+1][j+1].isWorked())
{
          // if the vertex is worked and the related vertex is worked
in the next row
          line(p[i][j].coordx(), p[i][j].coordy(), p[i+1][j+1].
coordx(), p[i+1][j+1].coordy() ); // line \
        }
        if (j%2 != 0 && p[i][j].isWorked() && p[i][j+1].isWorked()) {
          // if the vertex is worked and the related vertex is worked
in the next row
          line(p[i][j].coordx(), p[i][j].coordy(), p[i][j+1].coordx(),
p[i][j+1].coordy() ); // line /
        }
      }
    }

    ////// WORK DENSITY TRIANGLES //////////////
    for (int i = 0; i < cols; i++) {
      for (int j = 0; j <rows-1; j++) { // j = 1

        if (p[i][j].Selected() ) { //Selected = !selected in Points
class; or every triangle shows.
          color c = p[i][j].colors(i, j);

          if (j == 0 ) {

            newRow.setString("Rule Applied", "Rule 2");

            bli = 2;
            newRow.setInt("b(l_i)", bli);
            //int ebp = e_bobbin_prep;
            //E_PREP = ra*bli*ebp;
            //newRow.setFloat("E_PREP", E_PREP);

            //int eupn = e_unpin;
            //int psi = 1;
            //int ebf = e_bobbin_finish;
            //E_FINISH = rf*(bli*ebf + psi*eupn);
            //newRow.setFloat("E_FINISH", E_FINISH);
          }

          if (j == 1 ) { // i >0 && i < cols && ( !p[i-1][j].isWorked()
|| !p[i+1][j].isWorked())) { // INITIAL ROW

            newRow.setString("Rule Applied", "Rule 3");
          }

          if (j == 1 && i >0 && i < cols && ( p[i-1][j].isWorked() ||
p[i+1][j].isWorked())) {

            newRow.setString("Rule Applied", "Rule 4");
          }

          if (j%2 != 0 && i < cols-1 && j != 1) { //ODD ROWS

            triangle(p[i][j].coordx(), p[i][j].coordy(),
              p[i][j-1].coordx(), p[i][j-1].coordy(),
              p[i+1][j-1].coordx(), p[i+1][j-1].coordy());

            newRow.setString("Rule Applied", "Rule 5");
          }

          if (j%2 == 0 && i > 0 && i < cols-1 && j>0) { //EVEN ROWS //
exclude first row

            triangle(p[i][j].coordx(), p[i][j].coordy(),
              p[i-1][j-1].coordx(), p[i-1][j-1].coordy(),
              p[i][j-1].coordx(), p[i][j-1].coordy());

            newRow.setString("Rule Applied", "Rule 5");
          }

          if (j%2 == 0 && i == 0 && j > 0 ) { //LEFT EDGE

            triangle(p[i][j].coordx(), p[i][j].coordy(),
              p[i][j-2].coordx(), p[i][j-2].coordy(),
              p[i][j-1].coordx(), p[i][j-1].coordy());

            newRow.setString("Rule Applied", "Rule 10");
          }

          if (j%2 == 0 && i == cols-1 && j > 0 ) { //RIGHT EDGE

            triangle(p[i][j].coordx(), p[i][j].coordy(),
              p[i][j-2].coordx(), p[i][j-2].coordy(),
              p[i-1][j-1].coordx(), p[i-1][j-1].coordy());

            newRow.setString("Rule Applied", "Rule 10");
          }

          //newRow.setInt("Work Area", p[i][j].density(c));
          float norm = 784;
          rho_i = norm/p[i][j].density(c);
          if (p[i][j].density(c) == 0) {
            rho_i = 1;
          }
          newRow.setFloat("rho(S_i)", rho_i);
        }
      }
    }
    int csi = 2;
    int tsi = 2;
    int psi = 1;
    int msi = 1;
    int ebp = e_bobbin_prep;
    int ept = e_perceive_tight;
    int epp = e_perceive_pairs;
    int ecr = e_cross;
    int etw = e_twist;
    int epn = e_pin;
    int eupn = e_unpin;
    int etgt = e_tight;
    int ercl = e_recall;
    int ebf = e_bobbin_finish;
    E_PERC = rp*(rho_i*(ept+delta*epp));
    E_RECALL = rr*ercl;
    E_PREP = ra*bli*ebp;
    E_WORK = rw*(csi*ecr+tsi*etw+psi*epn+msi*etgt);
    E_FINISH = rf*(bli*ebf + psi*eupn);
```

```
    EFFORT = E_PERC+E_RECALL+E_PREP+E_WORK+E_FINISH;

    newRow.setFloat("E_PERC", E_PERC);
    newRow.setFloat("E_RECALL", E_RECALL);
    newRow.setFloat("E_PREP", E_PREP);
    newRow.setFloat("E_WORK", E_WORK);
    newRow.setFloat("E_FINISH", E_FINISH);
    newRow.setFloat("EFFORT", EFFORT);
    saveTable(table, "data/new.csv");

    bli = 0; // Reset initial bobbins to 0
  }
  if (pdf) {
    PGraphicsPDF pdf = (PGraphicsPDF) g;
    if (frameCount == 100) {
      exit();
    } else {
      pdf.nextPage();
    }
  }
  if (record) {
    endRecord();
    record = false;
  }
} ////// END OF DRAW /////////////

void lockall(VerletParticle2D[][] particles) {
  for (int i = 0; i < cols; i++) {
    for (int j = 0; j < rows; j++) {
      particles[i][j].lock();
    }
  }
  physics.update();
}
```

```
void keyPressed() {
  if (key == 'q') {
    //SVG FINISH
    endRecord();
    println("Finished");
    exit();
  }
  if (key == 's') {
    if (Particle_lock== false) {
      Particle_lock= true;
    } else {
      Particle_lock= false;
    }
    println("stopped");
  }
  if (key == 'p') {
    initialize = true;
    println("initialized");
  }
  if (key == 'w') {
    lace = true;
    println("lacemaking");
  }
  if (key == 'r') {
    record = true;
    println("svg saved");
  }
}
void mousePressed() {
  //record = true;
  addAttractor();
  mousePos = new Vec2D(mouseX, mouseY);
}
boolean rasterScan() {
  if (raster) {
    return true;
  } else {
    return false;
  }
}

boolean diagonalScan() {
  if (diagonal) {
    return true;
  } else {
    return false;
  }
}

boolean randomScan() {
  if (random) {
    return true;
  } else {
    return false;
  }
}
//FUNCTION THAT CREATES THE PARTICLE
VerletParticle2D addParticle(float xPos, float yPos) {
  VerletParticle2D v = new VerletParticle2D(xPos, yPos);
  physics.addParticle(v);
  return v;
}
void addAttractor() {

  ///DIFFERENT METHODS TO CREATE A GROUP OF ATTRACTORS////

  //float symX =mouseX;
  //float symY = height - mouseY;
  //Vec2D randLoc = Vec2D.randomVector().scale(7).addSelf(mouseX,
mouseY);
  //Vec2D randLocSym = Vec2D.randomVector().scale(5).addSelf(symX,
symY);
  //VerletParticle2D a = new VerletParticle2D(randLoc);
  //VerletParticle2D b = new VerletParticle2D(randLocSym);
  //fill(0);
  //ellipse(mouseX, mouseY, 10, 10);
  //ellipse(symX, symY, 10, 10);
  //fill(255);
  //physics.addParticle(a);
  //physics.addParticle(b);
  //physics.addBehavior(new AttractionBehavior2D(a, width/2, .2, -.3));
  //physics.addBehavior(new AttractionBehavior2D(b, width/2, -.3, .2));
  //float symX =width - mouseX;
  //float symY = mouseY;
  //Vec2D click = Vec2D.randomVector().scale(5).addSelf(mouseX,
mouseY);
  //Vec2D clickSym = Vec2D.randomVector().scale(5).addSelf(symX, symY);
  //VerletParticle2D a = new VerletParticle2D(click);
  //VerletParticle2D b = new VerletParticle2D(clickSym);
  //fill(0);
  //ellipse(mouseX, mouseY, 10, 10);
  //ellipse(symX, symY, 10, 10);
  //fill(255);
  //physics.addParticle(a);
  //physics.addParticle(b);
  //physics.addBehavior(new AttractionBehavior2D(a, 400, .3, .1));
  //physics.addBehavior(new AttractionBehavior2D(b, 400, .3, .1));


  //float symX = random(width);
  //float symY = random(height);
  //for (int i = 0; i < 5; i++) {
  //  Vec2D test = Vec2D.randomVector().scale(5).addSelf(width-
i*(width/50), (height-i*(height/4)));

  //  VerletParticle2D a = new VerletParticle2D(test);
  //  physics.addParticle(a);
  //  physics.addBehavior(new AttractionBehavior2D(a, 100, .1, .1));
  //  ellipse(a.x, a.y, 20, 20);
  //  fill(255);
  //}

  //Vec2D click = Vec2D.randomVector().scale(5).addSelf(symX, symY);

  ////Vec2D click = Vec2D.randomVector().scale(5).addSelf(mouseX,
mouseY);
  //Vec2D click = Vec2D.randomVector().scale(5).addSelf(symX, symY);
  //Vec2D clickSym = Vec2D.randomVector().scale(5).addSelf(width-symX,
symY);

  //VerletParticle2D a = new VerletParticle2D(click);
  //VerletParticle2D b = new VerletParticle2D(clickSym);
  //fill(0);
  ////ellipse(mouseX, mouseY, 20, 20);
  //ellipse(symX, symY, 20, 20);
  //ellipse(width-symX, symY, 20, 20);
  //fill(255);
  //physics.addParticle(a);
  //physics.addParticle(b);
  //physics.addBehavior(new AttractionBehavior2D(a, 100, .1, .1));
  //physics.addBehavior(new AttractionBehavior2D(b, 100, .1, .1));

  //float symX = random(width);
  //float symY = random(height);
  for (int i = 1; i < 10; i+=2) {

    float symX = (width - width/2);
    float symY = (height - height/6);

    float symX1 = (width/2);
    float symY1 = (height/6);

    //float symX2 = (width-width/3);
    //float symY2 = (height/2);

    //float symX3 = (width/3);
    //float symY3 = (height/2);

    //float symX4 = (width/2);
    //float symY4 = (height/3);

    //float symX5 = (width - width/2);
    //float symY5 = (height - height/3);
    //float symX6 = (width/3);
    //float symY6 = (height/6);

    //float symX7 = (width-width/3);
    //float symY7 = (height/6);

    Vec2D test  = Vec2D.randomVector().scale(4).addSelf( symX, symY );
    Vec2D test2 = Vec2D.randomVector().scale(4).addSelf( symX1, symY1
);
    //Vec2D test3 = Vec2D.randomVector().scale(4).addSelf( symX2,
symY2 );
    //Vec2D test4 = Vec2D.randomVector().scale(4).addSelf( symX3,
symY3 );

    //Vec2D test5 = Vec2D.randomVector().scale(4).addSelf( symX4,
symY4 );
    //Vec2D test6 = Vec2D.randomVector().scale(4).addSelf( symX5,
symY5 );

    //Vec2D test7 = Vec2D.randomVector().scale(4).addSelf( symX6,
symY6 );
    //Vec2D test8 = Vec2D.randomVector().scale(4).addSelf( symX7,
symY7 );
    VerletParticle2D a = new VerletParticle2D(test);
    VerletParticle2D b = new VerletParticle2D(test2);
    //VerletParticle2D c = new VerletParticle2D(test3);
    //VerletParticle2D d = new VerletParticle2D(test4);
    //VerletParticle2D e = new VerletParticle2D(test5);
    //VerletParticle2D f = new VerletParticle2D(test6);
    //VerletParticle2D g = new VerletParticle2D(test7);
    //VerletParticle2D h = new VerletParticle2D(test8);
    physics.addParticle(a);
    physics.addParticle(b);
    //physics.addParticle(c);
    //physics.addParticle(d);
    //physics.addParticle(e);
    //physics.addParticle(f);
    //physics.addParticle(g);
    //physics.addParticle(h);
    for (int j = 0; j < 10; j++) {
      Float[] anchors = new Float[10];
      anchors[j] = i*10.;

      physics.addBehavior(new AttractionBehavior2D(a, 200, .02, 0));
      physics.addBehavior(new AttractionBehavior2D(b, 200, .02, 0));

      //physics.addBehavior(new AttractionBehavior2D(c, 150, .2, 0));
      //physics.addBehavior(new AttractionBehavior2D(d, 150, .2, 0));
      //physics.addBehavior(new AttractionBehavior2D(e, 150, .2, 0));
      //physics.addBehavior(new AttractionBehavior2D(f, 150, .2, 0));

      //physics.addBehavior(new AttractionBehavior2D(g, 150, .2, 0));
      //physics.addBehavior(new AttractionBehavior2D(h, 150, .2, 0));
    }

    fill(0);
    ellipse(a.x, a.y, 20, 20);
    ellipse(b.x, b.y, 20, 20);
    //ellipse(c.x, c.y, 20, 20);
    //ellipse(d.x, d.y, 20, 20);
    //ellipse(e.x, e.y, 20, 20);
    //ellipse(f.x, f.y, 20, 20);
    ////fill(0,255,0);
    //ellipse(g.x, g.y, 20, 20);
    //ellipse(h.x, h.y, 20, 20);
  }
}


class Attractor {
  PVector pos;
  float charge;
  boolean selected;
  Attractor (float x, float y, float charge) {
    pos = new PVector(x, y);
    this.charge = charge;
  }
  void display() {
    pushMatrix();
    translate(pos.x, pos.y);
    //noStroke();
    fill(0);
    ellipse(0, 0, charge, charge);
    popMatrix();
  }
}
class Points {
  float xpos;
  float ypos;
  float dia;
  boolean worked;
  boolean legal;
  boolean initial;
  boolean selected;
  boolean wasWorked;
  boolean wasSelected;
  int val;
  int[] threadsInit = new int[4];
  int[] threadsL = new int[2];
  int[] threadsR = new int[2];
  int[] threadsIn = new int[4];
  int[] threadsOut = new int[4];
  Points(float xpos_, float ypos_, float dia_) {
    xpos = xpos_;
    ypos = ypos_;
    dia = dia_;
    worked = false;
    legal = false;
    wasSelected = false;
    val =0;
  }
  float coordx() {
    return(xpos);
  }
  float coordy() {
    return(ypos);
```

```
}
//////////////////////////////////
//////////////////////////////////
int[] threadsInit(int val) {
  int g = (val*4);
  if (initial) {
    for (int i = 0; i < 4; i ++) {
      threadsInit[i] = (g+i);
    }
  }
  return threadsInit;
}
int[] threadsL() {
  return threadsL;
}
int[] threadsR() {
  return threadsR;
}
int[] threadsIn() {
  return threadsIn;
}
int[] threadsOut() {
  for (int i = 0; i<4; i++) {
    if (initial) {
      threadsOut[i] = threadsInit[i];
    } else {
      threadsOut[i] = threadsIn[3-i];
    }
  }
  return threadsOut;
}
color colors(int i, int j) {
  float g = map(i, 0, cols, 0, 255);
  float b = map(j, 0, rows -1, 0, 255);
```

```
    color c = color(255, g, b);
    fill(c);
    return c;
  }
  int density(int c) {
    loadPixels();
    int counter = 0;
    for (int q = 0; q < width*height; q++) {
      if (pixels[q] == c) {
        counter = counter + 1;
      }
    }
    return counter;
  }
  //////////////////////////////////
  //////////////////////////////////
  void select() {
    selected = true;
    fill(255, 153, 52);
    rect(xpos-15, ypos-15, 30, 30);
  }
  boolean Selected() {
    if (selected) {
      selected = !selected;
      return true;
    } else {
      return false;
    }
  }
  boolean isWorked() {
    if (worked) {
      return true;
    } else {
      return false;
    }
  }
  boolean isLegal() {
    if (legal) {
      return true;
    } else {
      return false;
    }
  }
  void initialRow() {
    initial = true;
    legal = true;
  }
  void nextLegal() {
    legal = true;
  }
  void display() {
    if (legal) {
      if (selected) {
        fill(255, 234, 21);
        worked = true;
      } else if (initial) {
        fill(255, 0, 0); // Initial positions are Red until worked
      } else {
        fill(127); //Legal and unworked stitches are grey.
      }
    } else {
      noFill(); // vertices are just outlines until legal or worked
      worked = false;
    }
    ellipse(xpos, ypos, dia, dia);
  }
}
///// END OF THESIS /////
```