UNIVERSITY OF CALIFORNIA,
IRVINE

Constructing, Counting and Matching Combinatorial and Geometric Shapes

DISSERTATION

submitted in partial satisfaction of the requirements
for the degree of

DOCTOR OF PHILOSOPHY

in Computer Science

by

Martha Carolina Osegueda Escobar

Dissertation Committee:
Distinguished Professor Michael T. Goodrich, Chair
Distinguished Professor David Eppstein
Professor Sandy Irani

2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

First I'd like to thank my advisor, Michael Goodrich, for his guidance, support and encouragement these last five years. Although I might have felt uncertain at times he would always help drive me forward and connected me to opportunities that allowed me to collaborate with numerous different people. I'd also like to thank Amelia Regan, my first contact at UCI, for her encouragement and support my first year at UCI. I would also like to thank Gill Barequet, for all of his words of encouragement, his collaboration and support ever since we first collaborated during my visit to the Technion. I would also like to thank David Eppstein and Sandy Irani for serving in my defense and advancement committees, and to thank Vijay Vazirani and John Avise for serving in my advancement committee.

I was lucky to be introduced to the field of computer science in high-school by Mrs. Serrano, although my introduction to algorithms was limited I immediately fell in love with the drafting and optimizing of algorithms. Through my undergraduate studies I was introduced more formally to algorithms and our ability to characterize their time and space complexities through proofs. I am grateful I was be able to peer-lead and, later, TA during my undergraduate studies and grateful for the opportunity to learn from Claudia Casas who was always so devoted to serving her community, may she rest in peace. I would also like to thank Dr. Vladik Kreinovich who introduced me to research and collaborated with on my first publications. Before coming to UCI I was unfamiliar with the fields of Computational Geometry and Graph Algorithms, all I knew was I really liked beautiful algorithms and elegant proofs. Ultimately these two fields would represent the majority of my work throughout this program and I am grateful for the opportunity to both take and TA for these courses when they were offered by Michael Goodrich and David Eppstein, respectively. My publications have covered a wide breadth of topics in these fields and for this I am grateful to my advisor, Michael Goodrich. Through this freedom I was able to collaborate with a wide array of people and learn about a wide array of topics and this is something I will always be grateful for.

On that note, I would like to thank my co-authors and collaborators. For their collaboration in the work included in Chapter 2, I would like to thank Ramtin Afshar, Michael Goodrich and Pedro Matias. For their collaboration in the work included in Chapter 3, I would like to thank Gill Barequet and Gil Ben-Shachar, additionally I'd like to thank Günter Rote and Vuong Bui for helpful comments on preliminary drafts of this work. For their collaboration in the work included in Chapter 4, I would like to thank Juan Besa, Timothy Johnson, Nil Mamano and Parker Williams. For their collaboration in the work included in Chapter 5, I would like to thank Gill Barequet, Shion Fukuzawa, Michael Goodrich, David Mount and Evrim Ozel. I would also like to thank co-authors in works not included in this dissertation, including David Eppstein, Daniel Frishberg, Andrei Asinowski and Günter Rote. I would also like to thank current and former theory graduate students, for their engaging discussions, friendship and support.

Lastly, I would like to thank people in my life that have supported and encouraged me throughout this journey. I would like to begin by thanking my partner, David, who has

consistently extended his love and support the entire way and without whom I likely would not have the strength to get here. I would also like to thank my family for always encouraging me in my learning, especially my mom who has sacrificed so much to support me and my siblings. I would like to thank my siblings for motivating me, my sister Andrea through her perseverance and my brother Alex through his youthful excitement and warmth. I would also like to thank my dad for instilling curiousity in me throughout my childhood. It would be impossible for me not to extend the largest of thanks to my aunt and uncle, Lizeth and Roberto Osegueda, for hosting me through my undergraduate studies, and enabling and encouraging me to continue my studies. Not only did they host me but they also provided me with a surrogate family away from home and they will always hold a special place in my heart. Finally I would also like to thank members of my extended families Osegueda, Escobar and Baires for their warmth and support.

When I was little I would tell anyone who would listen that I wanted to be a "mad scientist" when I grew up. This answer has changed very little throughout my life, however I eventually realized it would be a "mad *computer* scientist" instead.

VITA

Martha Carolina Osegueda Escobar

EDUCATION

Doctor of Philosophy in Computer Science — **2022**
University of California, Irvine — ***Irvine, CA***

Masters of Science in Computer Science — **2019**
University of California, Irvine — ***Irvine, CA***

Bachelor of Science in Computer Science — **2017**
University of Texas at El Paso — ***El Paso, TX***

RESEARCH EXPERIENCE

Graduate Research Assistant — **2017–2022**
University of California, Irvine — ***Irvine, CA***

Visiting Researcher — **2019**
Technion — Israel Institute of Technology — ***Haifa, Israel***

Undergraduate Student Researcher — **2015–2017**
University of Texas at El Paso — ***El Paso, TX***

TEACHING EXPERIENCE

Teaching Assistant — **2017–2022**
University of California, Irvine — ***Irvine, CA***

Undergraduate Teaching Assistant — **2016–2017**
University of Texas at El Paso — ***El Paso, TX***

Undergraduate Peer-Leader — **2014–2015**
University of Texas at El Paso — ***El Paso, TX***

REFEREED JOURNAL PUBLICATIONS

Angles of Arc-Polygons and Lombardi Drawings of Cacti — **Under Review**
David Eppstein, Daniel Frishberg, and Martha C. Osegueda

On the Number of Compositions of Two Polycubes — **Under Review**
Andrei Asinowski, Gill Barequet, Gil Ben-Shachar, and Martha C. Osegueda, Günter Rote

Taming the knight's tour: Minimizing turns and crossings — **January 2022**
Juan Jose Besa, Timothy Johnson, Nil Mamano, Martha C. Osegueda, Parker Williams

Theoretical Computer Science 902

Concatenation arguments and their applications to polyominoes and polycubes — **October 2021**
Gill Barequet, Gil Ben-Shachar, and Martha C. Osegueda

Computational Geometry 98

REFEREED CONFERENCE PUBLICATIONS

Diamonds are Forever in the Blockchain: Geometric Polyhedral Point-Set Pattern Matching — **Under Review**
Gill Barequet, Shion Fukuzawa, Michael T. Goodrich, David M. Mount, Martha C. Osegueda, Evrim Ozel

Mapping Networks via Parallel kth-Hop Traceroute Queries — **March 2022**
Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda

39th International Symposium on Theoretical Aspects of Computer Science (STACS 2022)

Angles of Arc-Polygons and Lombardi Drawings of Cacti — **August 2021**
David Eppstein, Daniel Frishberg, and Martha C. Osegueda

33rd Canadian Conference on Computational Geometry (CCCG 2021)

Taming the knight's tour: Minimizing turns and crossings **September 2020**

Juan Jose Besa, Timothy Johnson, Nil Mamano, Martha C. Osegueda

10th International Conference on Fun with Algorithms (FUN 2021)

Reconstructing Biological and Digital Phylogenetic Trees in Parallel **August 2020**

Ramtin Afshar, Michael T. Goodrich, Pedro Matias, and Martha C. Osegueda

28th Annual European Symposium on Algorithms (ESA 2020)

Minimum-Width Drawings of Phylogenetic Trees **December 2019**

Juan Jose Besa, Michael T. Goodrich, Timothy Johnson, and Martha C. Osegueda

13th International Conference on Combinatorial Optimization and Applications (COCOA 2019)

Fuzzy-inspired hierarchical version of the von Neumann-Morgenstern solutions as a natural way to resolve collaboration-related conflicts **October 2016**

Olga Kosheleva, Vladik Kreinovich, Martha Osegueda Escobar

2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)

How to transform partial order between degrees into numerical values **October 2016**

Olga Kosheleva, Vladik Kreinovich, Joe Lorkowski, Martha Osegueda

2016 IEEE International Conference on Systems, Man, and Cybernetics (SMC)

ABSTRACT OF THE DISSERTATION

Constructing, Counting and Matching Combinatorial and Geometric Shapes

By

Martha Carolina Osegueda Escobar

Doctor of Philosophy in Computer Science

University of California, Irvine, 2022

Distinguished Professor Michael T. Goodrich, Chair

A shape can be defined as the representation of an object or its external boundary so as to characterize the information remaining when describing an object absent manipulations such as translation, rotation, reflection, or scaling. This includes combinatorial shapes, such as for structures defined in terms of relationships such as "parent-child", "sibling", or "predecessor-successor", as well as geometric shapes, such as those defined by polygons or polytopes. In this thesis, we provide novel methods for constructing, counting, and matching combinatorial and geometric shapes. For example, we study learning and constructing phylogenetic trees, which describe evolutionary relationships among a group of objects. Reconstructing phylogenetic trees is an important problem in computational biology, data protection and computer security. With respect to counting, we focus on characterizing the growth in the number of polycubes and families of polycubes, where a polycube is the shape defined by a connected set of n copies of d-dimensional hypercubes. Counting polycubes of a given size has been long-studied in statistical physics, given their applications to percolation processes. We also consider constructing knight's tour shapes, for chess board grids, optimizing two new metrics of "simplicity" in knight's tours: the number of turns and the number of crossings. Finally, we focus on matching geometric shapes to real-world objects. In particular, motivated by the tracing of ethically-sourced diamonds, we develop

approximation algorithms for geometric polyhedral point-set pattern matching as a minimum width polyhedral annulus problem under translations and rotations.

Chapter 1

Introduction

A shape is the geometric representation of an object or its external boundary. These encompass a wide breadth of shapes which are commonly used across a variety of applications. In particular, we follow precedent set in [76] and define a shape as the geometric information remaining when removing any manipulations from its description, such as translation, rotation and uniform scaling. Some examples of shapes include lines, squares, cubes, polyhedra, polytopes and any combination of them (see Figure 1.1). However, this also includes combinatoral shapes such as those defined in terms of relationships such as "parent-child", "sibling", or "predecessor-successor" as well as those defined as combinations of geometric shapes like polycubes.

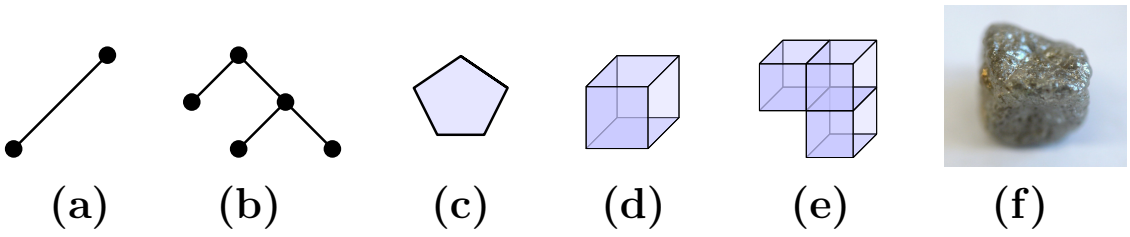


Figure 1.1: Some examples of shapes: **(a)** a line, **(b)** a tree, **(c)** an irregular pentagon, **(d)** a cube, **(e)** a polycube, and **(f)** the shape of a rough diamond.

1.1 Combinatorial Shapes

Let us consider a shape to be **combinatorial** if it can be described as a combination of other "simple" shapes. For example, in Figure 1.1, (b) and (e) are composed of multiple copies of (b) and (d), respectively. Although combinatorial shapes can be the result of combining different simple shape, this disseration focuses on combinatorial shapes that result from combining multiple copies of the same shape.

Combinatorial shapes lend themselves to solving a variety of problems, since combinations of simple shapes define different resulting combinatorial shapes. For example, the tree in Figure 1.1 (b) does not have the same shape as a tree with four leaves even if it uses the same number of lines. Similarly, the polycube in Figure 1.1 (e) is comprised of three cubes but does not have the same shape as the polycube where the three cubes are attached in a straight line.

This also becomes more involved whenever certain copies of the comprising simple shapes describe distinct **relationships**. For example, in Figure 1.1(b) it is implied by graph drawing convention that it is an upward drawing of a tree, where the top node represents the root and tree edges are directed away from it to represent a "parent-child" relationship. If one of the implied root's neighbors became the new root, these relationships would change and the tree's shape would be different.

Knowing the general form of a combinatorial shape, creates two natural dual problems:

1. Constructing a given combinatorial shape by finding the required combination, and
2. Counting, or finding, all possible combinatorial shapes.

In this dissertation, Chapter 2 studies constructing phylogenetic trees, Chapter 3 focuses on counting polycubes and families of polycubes, and Chapter 4 focuses on constructing knight's tours that minimize turns and crossings.

1.1.1 Learning or Constructing Combinatorial Shapes

We refer to the process of learning the combination achieving a combinatorial shape as learning or constructing the shape. In general, if the shape is unknown we can rely on "asking" a series of questions about the structure. The answers impose constraints that restrict the valid combinations that could form our combinatorial shape. Ultimately the goal becomes to ask the questions which are the most helpful in isolating the target combination. Conventionally, the process of asking a question is considered a **query** and algorithms tend to focus on minimizing the **query-complexity**, $Q(n)$.

More specifically we focus on phylogenetic applications where phylogenetic trees are trees whose shape represents the evolutionary relationships between a group of species. The branching of the tree describes how the relevant biological entities have evolved through common ancestors [73, 31, 92]. Learning the shape of the phylogenetic trees involve very expensive ***queries*** (*i.e.* experiments) that test species' evolutionary similarity [73, 48, 59, 95, 70, 87, 105, 26, 45, 44, 43]. Similarly, phylogenetic trees have also been used to represent the evolution of digital objects, and how they have been edited and transformed. Phylogenetic trees of digital objects have applications for data protection, computer security, privacy, copyright disputes, and plagiarism checking [59, 95, 70, 87, 105, 26, 45, 44, 43].

Given the biological applications we consider only queries involving the leaves of our tree. Leaves will usually represent living biological entities and internal nodes will correspond to evolutionary ancestors that are likely to have become extinct. More specifically we focus on ***relative-distance*** queries, which given three leaf nodes x, y, and z, responds

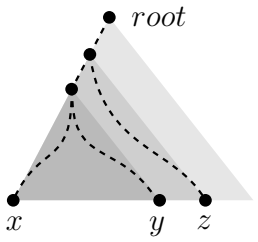


Figure 1.2: An example of a relative distance query where a query with leaves x, y and z would return (x, y)

with the pair of species (x, y),(x, z) or (y, z) that has the most-recent common ancestor, *i.e.* the two "closest" leaves [73] (as shown in Figure 1.2). Previous work on phylogenetic tree reconstruction through relative-distance queries, to our knowledge, focused solely on sequential reconstruction and proven $O(n \log n)$ queries as optimal [73, 48]. Reconstructions using a more powerful version of relative-distance queries have been proven to require an asymptotically optimal $O(dn \log_d n)$ queries when the tree has maximum degree d [115, 61].

Our Results. In Chapter 2, we propose a randomized divide-and-conquer parallel algorithm with $O(n \log n)$ query complexity and $O(\log n)$ rounds, with high probability (w.h.p.)[1]. We also provide experimental analysis to compare its performance against the two known sequential algorithms: the deterministic algorithm introduced by Kannan, Lawler and Warnow [73] which is reminiscent of insertion sort, and the randomized algorithm introduced by Emamjomeh-Zadeh and Kempe [48] inspired by hierarchical clustering.

1.1.2 Counting Combinatorial Shapes

As the number of components in a combinatorial shape increase, it becomes increasingly more computationally expensive to exhaustively enumerate all possible composite shapes. For this reason work tends to focus on counting the number of possible shapes of a given size, or deriving formulas approximating the number of possible shapes of a given size.

[1]We say that an event occurs with high probability if it occurs $1 - 1/n^c$, for some constant $c \geq 1$.

Specifically we focus on developing methods for polycubes, which are comprised of n copies of d-dimensional hypercubes. These hypercubes must be connected through $(d-1)$-dimensional facets. Counting polyominoes and polycubes has been a long-standing problem in statistical physics since the 1950s [30, 110], where they are usually referred to by the more general term of **lattice animals**, and play a significant roles in percolation processes and in the collapse transition when branched polymers are heated. Counts of animals and formulae for specific types of animals serve as mathematical models to describe these physical processes. Polycubes are animals of the d-dimensional cubical lattice, but animals of other lattices such as polyiamonds, animals of the triangular lattice, have also been studied.

In specific we concern ourselves with fixed polycubes, which are equivalent only if they can be transformed into the other by translation. Numerous previous work has been devoted to improving the best-known lower [18] and upper [19, 79] bounds on the growth constants of polyominoes. Conventionally, $A_d(n)$ denotes the number of d-dimensional polycubes of size n and λ_d denotes the growth constant. It has been proven [78] that $\lambda_2 := \lim_{n\to\infty} \sqrt[n]{A_2(n)}$ and that $A_2(n+1)/A_2(n)$ converges to λ_2 as $n \to \infty$ [85], however these results can be extended to any dimension.

Our Results. In Chapter 3, we extend the notion of "concatenation argument" and develop methods for deriving lower and upper bounds on the growth constants for families of polyominoes and polycubes whose enumerating sequences are so-called quasi sub- or super-multiplicative. We demonstrate various applications of this technique, set improved bounds on the growth constants of fixed polycubes when $d \geq 3$, and demonstrate how to achieve bounds for tree-like polycubes and convex polyominoes.

1.1.3 Constructing Shapes with Specific Properties

Alternatively, one might consider the problem of finding the specific combination of shapes, which optimize a given criteria in the combinatorial shape. In this process we would construct or realize a shape fitting the relevant criteria. One example criteria could be minimizing features such the number of times they changed the direction in which the shapes attach.

However exhaustively identifying each possible combinatorial shape in order to identify the optimum shape would be equally expensive. Thus, we focus on identifying shapes that **approximate** the optimum. The goal of approximation algorithms is to find a solution within a guaranteed approximation ratio of the optimum, for example, approximating a minimization problem would guarantee the solution is within the approximation ratio times the optimal.

Consider the game of chess, each piece moves in a particular way and a move can be defined in terms of "predecessor-successor" relationships. The movement of a singular piece across a chess board, therefore defines a combinatorial structure of cells connected by sequences of piece movements. In particular, this means connected cells must be a legal move away from eachother with respect to the piece being moved and each visited cell (apart from the start and endpoint) must have a predecessor and successor.

Specifically we focus on the knight's tour, which describe a sequence of **knight moves** visiting every single cell on the board. A knight moves two cells along one axis and one along the other (shown in Figure 1.3). The knight's tour is an old and heavily studied problem, see [15]. A knight's tour is consider closed if the last cell visited is also a knight's move away from the initial cell. We focus on closed tours, which turns this problem into a special case of the Hamiltonian cycle problem.

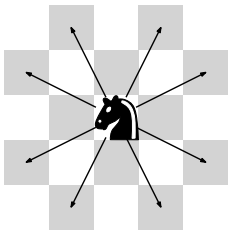


Figure 1.3: A knight moves one unit along one axis and two units along the other.

Our Results. In Chapter 4, we introduce two new metrics of "simplicity" for knight's tours: the number of turns and the number of crossings. We propose a novel algorithm for finding knight's tours on a $n \times n$ board that have $9.25n+O(1)$ turns and $12n+O(1)$ crossings. Consequently we prove that this is a $9.25/6 + o(1)$ approximation on the minimum number of turns and $3 + o(1)$ approximation on the minimum number of crossings. Our algorithm has an optimal running time, linear to the number of cells in an $n \times m$ board, of $O(nm)$. Further, this algorithm is fully parallelizable since it can be computed in constant time for each given index of the board; with $O(nm)$ processors it can be executed in $O(1)$ time. We also generalize for variations of the problem including: high-dimensional boards, symmetric tours under 90° rotation, tours in odd-width boards that skip a corner cell and tours for $(1, 4)$-leapers (also called *giraffes*).

1.2 Real-World and Geometric Shapes

The previous section dealt with combinatorial shapes both geometric and non-geometric. However, the shapes of real-world objects can also be equally as complex. Real-world shapes include infinitely small details in their surface which are impossible to capture exactly in a definition. For this purpose we instead represent the approximate shape of an object by sampling its surface.

However, it is still hard to reason about a point sample, and matching the pointset to more easily characterized geometric shapes, such as convex polyhedra, can help facilitate

comparisons amongst objects. Motivated by applications in the diamond industry, where the shape of diamonds are expected to fit a particular polyhedral shape, we focus on finding the ***minimum width*** polyhedral annulus that contains the sample points (shown in Figure 1.4). This would capture any surface imperfections within the annulus and capture any inaccuracies accrued through sampling. Variations of this problem in the plane have been studied. Some results include constrained versions of polygon annulus placement under polygonal offset [16], maximum point coverage with a polygon [17] and finding the minimum-width annulus for rectangles and squares [58, 12, 14, 13, 90].

Since, samples tend to be noisy we focus on developing efficient approximation schemes that find $(1+\varepsilon)$-approximate minimum width annuli for any fixed d-dimensional polyhedra. Approximation schemes have previously been developed for the minimum-width spherical annulus [32], and generally focus on identifying small subsets of points guaranteed to have $(1+\varepsilon)$-approximate solution [3, 96, 120, 4], these sets are commonly referred to as coresets.

Our Results. In Chapter 5, we provide $(1+\varepsilon)$-approximation algorithms for the minimum width annuli for any fixed polyhedra and polytopes. Given a set of n points in $\mathbf{R}^d$ and a convex polytope, we provide an $O(\varepsilon^{-d}n)$-time algorithm under translations, for $d \geq 3$, and $O(n \log \varepsilon^{-1} + \varepsilon^{-2})$ time for $d = 2$, and provide an $O(f^{d-1}\varepsilon^{1-2d}n)$-time algorithm when also allowing for rotations, parameterized on f, which we define as the slimness of the point set.

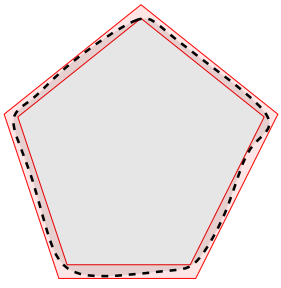


Figure 1.4: An example matching points sampled from the surface of a real-world object (surface is shown dashed and object is shown in gray) with the minimum width polyhedral annulus (shown in red).

Chapter 2

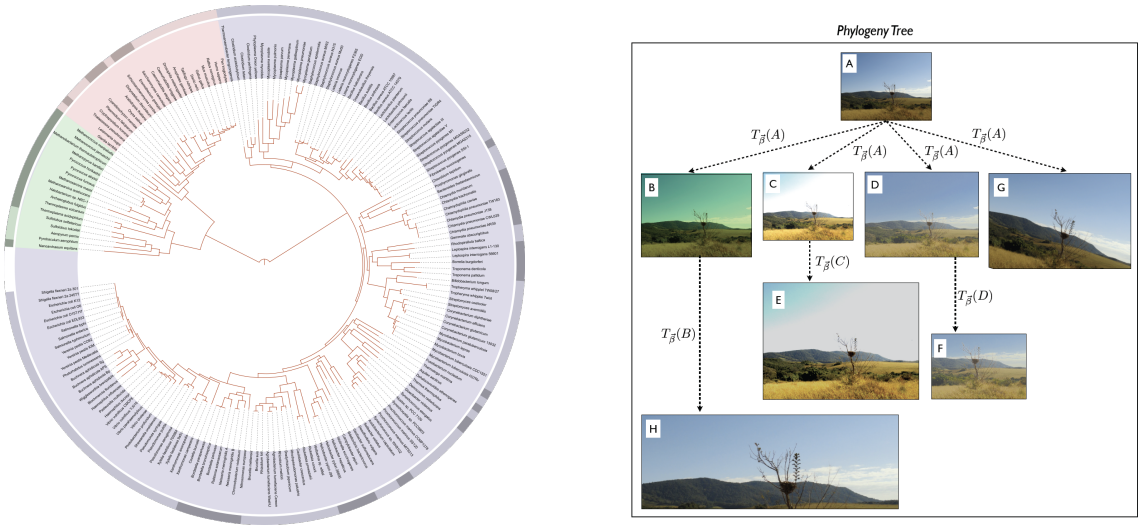
Reconstructing Biological Phylogenetic Trees in Parallel

2.1 Introduction

Phylogenetic trees represent evolutionary relationships among a group of objects. For instance, each node in a biological phylogenetic tree represents a biological entity, such as a species, bacteria, or virus, and the branching represents how the entities are believed to have evolved from common ancestors [73, 31, 92]. (See Figure 2.1a.) In a digital phylogenetic tree, on the other hand, each node represents a data object, such as a computer virus [59, 95], a source-code file [70], a text file or document [87, 105], or a multimedia object (such as an image or video) [26, 45, 44, 43] and the branching represents how these objects are believed to have evolved through edits or data compression/corruption. (See Figure 2.1b.)

In this chapter, we are interested in studying efficient methods for reconstructing phylogenetic trees from queries regarding their structure. In particular, with some exceptions,[1]

[1]One notable exception to this restriction of only being able to ask queries involving leaves in a biological phylogenetic tree is for phylogenetic trees of biological viruses, for which genetic sequencing may be known



(a) (b)

Figure 2.1: Two phylogenetic trees. (a) A biological phylogenetic tree of life, showing relationships between species whose genomes had been sequenced as of 2006; public domain image by Ivica Letunic, retraced by Mariana Ruiz Villarreal. (b) A digital phylogenetic tree of images, from Dias *et al.* [43].

in a biological phylogenetic tree we can only perform queries involving the leaves of the tree, since these typically represent living biological entities and internal nodes represent ancestors that are likely to be extinct. This type of phylogenetic tree has also received attention in the context of *hierarchical clustering*, where the goal is to provide a hierarchical grouping structure of items according to their similarity [48]. therefore, we study both types of querying regimes in this chapter.

More specifically, with respect to biological phylogenetic trees, we focus on ***relative-distance*** queries, where one is given three leaf nodes (corresponding to species), x, y, and z, and the response is a determination of which pair, (x, y), (x, z), or (y, z), is a closest pair, hence, has the most-recent common ancestor [73].

The motivation for reconstructing phylogenetic trees comes from a desire to better understand the evolution of the objects represented in a given phylogenetic tree. For

for all instances; hence, ancestor-descendant path queries might also be appropriate for reconstructing some biological phylogenetic trees.

example, understanding how biological species evolved is useful for understanding and categorizing the fossil record and understanding when species are close relatives [73, 48]. Similarly, understanding how digital objects have been edited and transformed can be useful for data protection, computer security, privacy, copyright disputes, and plariagism detection [59, 95, 70, 87, 105, 26, 45, 44, 43]. For instance, understanding the evolutionary process of a computer virus can provide insights into its ancestry, characteristics of the attacker, and where future attacks might come from and what they might look like [95].

The efficiency of a tree reconstruction algorithm can be characterized in terms of its ***query-complexity*** measure, $Q(n)$, which is the total number of queries of a certain type needed to reconstruct a given tree. This parameter comes from machine-learning and complexity theory, e.g., see [1, 34, 46, 109], where it is also known as "decision-tree complexity," e.g., see [118, 25]. Previous work on tree reconstruction has focused on sequential methods, where queries are issued and answered one at a time. For example, in pioneering work for this research area, Kannan *et al.* [73] show that an n-node biological phylogenetic tree can be reconstructed sequentially from $O(n \log n)$ three-node relative-distance queries. Indeed, their reconstruction algorithms are inherently sequential and involve incrementally inserting leaf nodes into the phylogenetic tree reconstructed for the previously-inserted nodes.

In many tree reconstruction applications, queries are expensive [73, 48, 59, 95, 70, 87, 105, 26, 45, 44, 43], but can be issued in batches. For example, there is nothing preventing the biological experiments [73] that are represented in three-node relative-distance queries from being issued in parallel. Thus, in order to speed up tree reconstruction, in this chapter we are interested in parallel tree reconstruction. To this end, we also use a ***round-complexity*** parameter, $R(n)$, which measures the number of rounds of queries needed to reconstruct a tree such that the queries issued in any round comprise a batch of independent queries. That is, no query issued in a given round can depend on the outcome of another query issued in that round, although both can depend on answers to queries issued in previous rounds.

Roughly speaking, $R(n)$ corresponds to the span of a parallel reconstruction algorithm and $Q(n)$ corresponds to its work. In this chapter, we are interested in studying complexities for $R(n)$ and $Q(n)$ with respect to biological and digital phylogenetic trees with fixed maximum degree, d.

2.1.1 Related Work

The general problem of reconstructing graphs from distance queries was studied by Kannan *et al.* [74], who provide a randomized algorithm for reconstructing a graph of n vertices using $\tilde{O}(n^{3/2})$ distance queries.[2]

Previous parallel work has focused on inferring phylogenetic trees through Bayesian estimation [5]. However, we are not aware of previous parallel work using a similar query models to ours. With respect to previous work on sequential tree reconstruction, Culberson and Rudnicki [37] provide the first sub-quadratic algorithms for reconstructing a weighted undirected tree with n vertices and bounded degree d from additive queries, where each query returns the sum of the weights of the edges of the path between a given pair of vertices. Reyzin and Srivastava [102] show that the Culberson-Rudnicki algorithm uses $O(n^{3/2} \cdot \sqrt{d})$ queries.

Waterman *et al.* [115] introduce the problem of reconstructing biological phylogenetic trees, using additive queries, which are more powerful than relative-distance queries. Hein [61] shows that this problem has a solution that uses $O(dn \log_d n)$ additive queries, when the tree has maximum degree d, which is asymptotically optimal [77]. Kannan *et al.* [73] show that an n-node binary phylogenetic tree can be reconstructed from $O(n \log n)$ three-node relative-distance queries and prove this is optimal through information theoretic bounds. Their method appears inherently sequential, however, as it is based on an incremental

[2]The $\tilde{O}(\cdot)$ notation hides poly-logarithmic factors.

approach that mimics insertion-sort. Similarly, Emamjomeh-Zadeh and Kempe [48] also give a sequential method using relative-distance queries that has a query complexity of $O(n \log n)$. Their algorithm, however, was designed for a different context, namely, hierarchical clustering.

Additionally, there exists some work (e.g. [71, 27, 63]) in an alternative perspective of the problem reconstructing phylogenetic trees, in which the goal is to find the best tree explaining the similarity and the relationship between a given fixed (or dynamic) set of data sequences (e.g. of species), using Maximum Parsimony [51, 54, 103] or Maximum Likelihood [52, 35]. This contrasts with our approach of recovering the "ground truth" tree known only to an oracle, which is consistent with its answers about the tree.

Our Contributions. In this chapter, we study the parallel phylogenetic tree reconstruction problem and show that an n-node rooted biological (binary) phylogenetic tree can be reconstructed from three-node ***relative-distance queries*** with $R(n)$ that is $O(\log n)$ and $Q(n)$ that is $O(n \log n)$, with high probability (w.h.p.)[3]. Both bounds are asymptotically optimal.

Moreover, given the many applications of biological and digital phylogenetic tree reconstruction, we feel that our algorithms have real-world applications. Thus, we have done an extensive experimental analysis of our algorithm, using both real-world and synthetic data for biological and digital phylogenetic trees. Our experimental results provide empirical evidence that our methods achieve significant parallel speedups while also providing improved query complexities in practice.

[3]We say that an event occurs with high probability if it occurs with probability at least $1 - 1/n^c$, for some constant $c \geq 1$.

2.2 Preliminaries

In graph theory, an ***arborescence*** is a directed graph, T, with a distinguished vertex, r, called the ***root***, such that, for any vertex v in T that is not the root, there is exactly one path from r to v, e.g., see Tutte [112]. That is, an arborescence is a graph-theoretic way of describing a rooted tree, so that all the edges are going away from the root. In this chapter, when we refer to a "rooted tree" it should be understood formally to be an arborescence.

We represent a rooted tree as $T = (V, E, r)$, with a vertex set V, edge set E, and root $r \in V$. The ***degree*** of a vertex in such a tree is the sum of its in-degree and out-degree, and the degree of a tree, T, is the maximum degree of all vertices in T. So, an arborescence representing a binary tree would have degree 3. Because of the motivating applications, e.g., from computational biology, we assume in this chapter that the trees we want to reconstruct have maximum degree that is bounded by a fixed constant, d.

Let us review a few terms regarding rooted trees.

Definition 2.1. *(ancestry) Given a rooted tree, $T = (V, E, r)$, we say u is **parent** of v (and v is a **child** of u) if there exists a directed edge (u, v) in E. The **ancestor** relation is the transitive closure of the parent relation, and the **descendant** relation is the transitive closure of the child relation. We denote the number of descendants of vertex s by $D(s)$. A node without any children is called a **leaf**. Given two leaf nodes, u and v in T, their **lowest common ancestor**, $lca(u, v)$, is the node, w in T, that is an ancestor of both u and v and has no child that is also an ancestor of u and v.*

We next define the types of queries we consider in this chapter for reconstructing a rooted tree, $T = (V, E, r)$.

Definition 2.2. *A **relative-distance query** for T is a function, closer, which takes three leaf nodes, u, v, and w in T, as input and returns the pair of nodes from the set, $\{u, v, w\}$,*

that has the lower lowest common ancestor. That is, $closer(u, v, w) = (u, v)$ if $lca(u, v)$ is a descendant of $lca(u, w) = lca(v, w)$. Likewise, we also have that $closer(u, v, w) = (u, w)$ if $lca(u, w)$ is a descendant of $lca(u, v) = lca(v, w)$, and $closer(u, v, w) = (v, w)$ if $lca(v, w)$ is a descendant of $lca(u, v) = lca(u, w)$.

Definition 2.2 assumes T is a binary tree (of degree 3). Note that in this chapter we restrict relative-distance queries to leaves, since these represent, e.g., current species in the application of reconstructing biological phylogenetic trees.

Definition 2.3. *A **path query** for T is a function, path, that takes two nodes, u and v in T, as input and returns 1 if there is a (directed) path from vertex u to v, and otherwise returns 0. Also, for $u \in V$ and $W \subseteq V$, we define $count(u, W) = \sum_{v \in W} path(u, v)$, which is the number of descendants of u in W.*

We next study some preliminaries involving the structure of degree-d rooted trees that will prove useful for our parallel algorithms.

Definition 2.4. *Let $T = (V, E, r)$ be a degree-d rooted tree. We say that an edge $e = (x, y) \in E$ is an **even-edge-separator** if removing e from T partitions it into two rooted trees, $T' = (V', E', y)$ and $T'' = (V'', E'', r)$, such that $\frac{|V|}{d} \leq |V'| \leq \frac{|V|(d-1)}{d}$ and $\frac{|V|}{d} \leq |V''| \leq \frac{|V|(d-1)}{d}$. (See Figure 2.2.)*

Lemma 2.1. *Every rooted tree of degree-d has an even-edge-separator.*

Proof. This follows from a result by Valiant [113, Lemma 2]. □

As we will see, this fact is useful for designing simple parallel divide-and-conquer algorithms. Namely, if we can find an even-edge-separator, then we can cut the tree in two by removing that edge and recurse on the two remaining subtrees in parallel (see Figure 2.2).

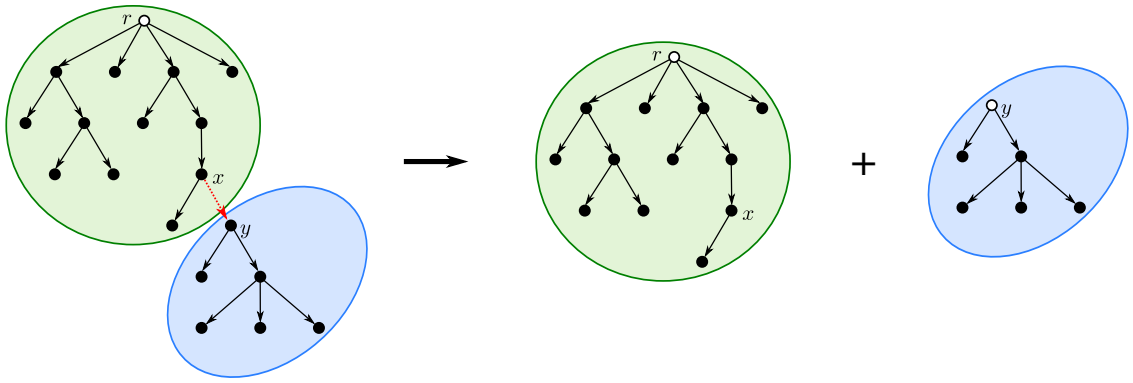


Figure 2.2: Illustration of a divide-conquer approach for trees. The edge (x, y) is an even-edge-separator. Note that the root of T'' is r, while y becomes root of T'.

2.3 Reconstructing Biological Phylogenetic Trees in Parallel

Relative-distance queries model an experimental approach to constructing a biological phylogenetic tree, e.g., where DNA sequences are compared to determine which samples are the most similar [73]. In simple terms, pairs of DNA sequences that are closer to one another than to a third sequence are assumed to be from two species with a common ancestor that is more recent than the common ancestor of all three. In this section, for the sake of tree reconstruction, we assume the ***responder*** has knowledge of the absolute structure of a rooted binary phylogenetic tree; hence, each response to a $closer(u, v, w)$ query is assumed accurate with respect to an unknown rooted binary tree, T. As in the pioneering work of Kannan *et al.* [73], we assume the distance comparisons are accurate and consistent. The novel dimension here is that we consider parallel algorithms for phylogenetic tree reconstruction.

As mentioned above, we consider relative-distance queries to occur between leaves of a rooted binary tree, T. That is, in our query model, the ***querier*** has no knowledge of the internal nodes of T and can only perform queries using leaves. Because T is a binary phylogenetic tree, we may assume it is a ***proper*** binary tree, where each internal node in T has exactly two children.

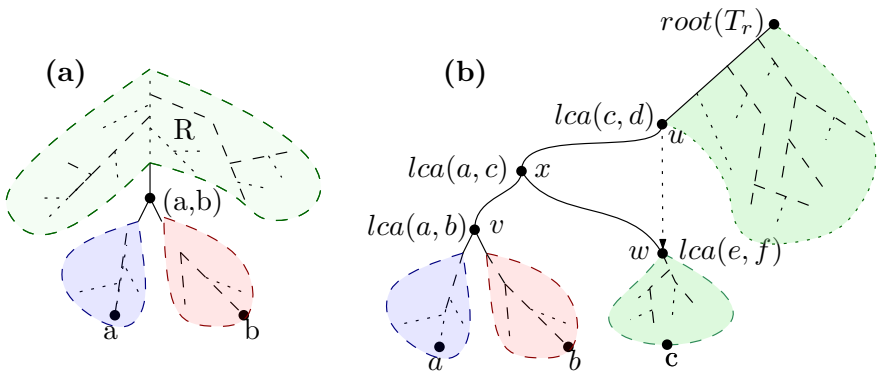


Figure 2.3: **(a)** The subgroups leaves are split into. **(b)** The linking step attaching T_a, T_b and T_r.

2.3.1 Algorithm

At a high level, our parallel reconstruction algorithm (detailed in Algorithm 1) uses a randomized divide-and-conquer approach, similar to Figure 2.2. In our case, however, the division process is random three-way split through a vertex separator, rather than an edge-separator-based binary split. Initially, all leaves belong to a single partition, L. Then two leaves, a and b, are chosen uniformly at random from L and each remaining leaf, c, is queried in parallel against them using relative-distance queries. Notice that the lowest common ancestor of a and b splits the tree into three parts. Given a and b, the other leaves are split into three subsets (R, A, and B) according to their query result (as shown in Figure 2.3(a)):

- A: leaves close to a, i.e., for which $closer(a, b, c) = (a, c)$
- B: leaves close to b, i.e., for which $closer(a, b, c) = (b, c)$
- R: remaining leaves, i.e., for which $closer(a, b, c) = (a, b)$

We then recursively construct the trees in parallel: T_a, for $A \cup \{a\}$; T_b, for $B \cup \{b\}$; and T_r, for R. The remaining challenge, of course, is to merge these trees to reconstruct the complete tree, T. The subtree of T formed by subset $A \cup B$ is rooted at an internal node, $v = lca(a, b)$; hence, we can create a new node, v, label it "$lca(a, b)$" and let T_a and T_b be

Algorithm 1: Reconstruct a binary tree of a set of leaves, L.

```
1  Function reconstruct-phylogenetic(L):
2      if |L| ≤ 3 then return the tree formed by querying L
3      Pick two leaves, a, b ∈ L, uniformly at random
4      for each c ∈ L s.t. c ≠ a, b do in parallel
5          Perform query closer(a, b, c)
6      Split the leaves in L into R, A, and B based on results
7      parallel do
8          T_a ← reconstruct-phylogenetic(A ∪ {a})
9          T_b ← reconstruct-phylogenetic(B ∪ {b})
10         T_r ← reconstruct-phylogenetic(R)
11     Let v be a new node, labeled "lca(a, b)"
12     Set v's left child to root(T_a) and the right to root(T_b)
13     if R = ∅ then return tree, T_v, rooted at v
14     else return link(v, T_r)
```

v's children. If $R = \emptyset$, then we are done. Otherwise, we need to determine the parent of v in T; that is, we need to link v into T_r using function $\mathsf{link}(v, T_r)$ (see Algorithm 2).

To identify the parent of v, in T, let us assume inductively that each internal node $u \in T_r$ has a label "$lca(c, d)$", since we have already recursively labeled each internal node in T. Recall that v is labeled with "$lca(a, b)$". The crucial observation is to note if there exists an edge $(u \to w)$ in T_r, such that u is labeled "$lca(c, d)$" and $closer(a, c, d) = (a, z)$ for $z \in \{c, d\}$, and w is either leaf z or an ancestor of z labeled "$lca(e, f)$" with $closer(a, e, f) = (e, f)$, (See Figure 2.3(b)), then edge $(u \to w)$ must be where the parent of v belongs in T, and if there is no such edge, the parent of v is the root of T and the sibling of v is the root of T_r. We can determine the edge $(u \to w)$ in a single parallel round by performing the query, $closer(a, c, d)$, for each each internal node $u \in T$ (where the label of u is "$lca(c, d)$"). It is also worth noting that if the oracle can identify cases where all three leaves share a single lca, simple modifications to Algorithm 1 would enable it to handle trees of higher degree.

Algorithm 2: Link v into the tree, T_r.

Function link(v, T_r)**:**
  for ***each internal node*** $u \in T_r$ **do in parallel**
    Query $closer(a, c, d)$, where u's label is "$lca(c, d)$"
  Let $(u \to w)$ be the edge in T_r such that:
   u is labeled "$lca(c, d)$" and $closer(a, c, d)$ is (a, z),
   where $z \in \{c, d\}$, and
   w is leaf z, or an ancestor of z with label "$lca(e, f)$" and $closer(a, e, f) = (e, f)$
  if ***no such edge*** $(u \to w)$ ***exists*** **then**
    Let g be a leaf from $root(T_r)$'s lca label
    Create a new root node, labeled "$lca(a, g)$" with v as left child and $root(T_r)$ as right child
    return the tree rooted at this new node
  Remove $(u \to w)$ from T_r
  Create a new node, x, labeled "$lca(a, z)$", with parent u, left child v, and right child w
  return T_r

2.3.2 Analysis

The correctness of our algorithm follows from the way relative-distance queries always return a label for the lowest common ancestor for the two closest leaves among the three input nodes. Furthermore, executing the three recursive calls can be done in parallel, because $A \cup \{a\}$, $B \cup \{b\}$, and R form a partition of the set of leaves, L, and at every stage we only perform relative-distance queries relevant to the respective partition.

Theorem 2.1. *Given a set, L, of n leaves in a proper binary tree, T, such as a biological phylogenetic tree, we can reconstruct T using relative-distance queries with a round complexity, $R(n)$, that is $O(\log n)$ and a query complexity, $Q(n)$, that is $O(n \log n)$, with high probability.*

Proof. Because the recursive calls we perform in each call to the reconstruct algorithm are done in parallel on a partition of the leaf nodes in L, we perform $\Theta(n)$ work per round. Thus, showing that the number of rounds, $R(n)$, is $O(\log n)$ w.h.p. also implies that $Q(n)$ is $O(n \log n)$ w.h.p. To prove this, we show that each round in Algorithm 1 has a constant

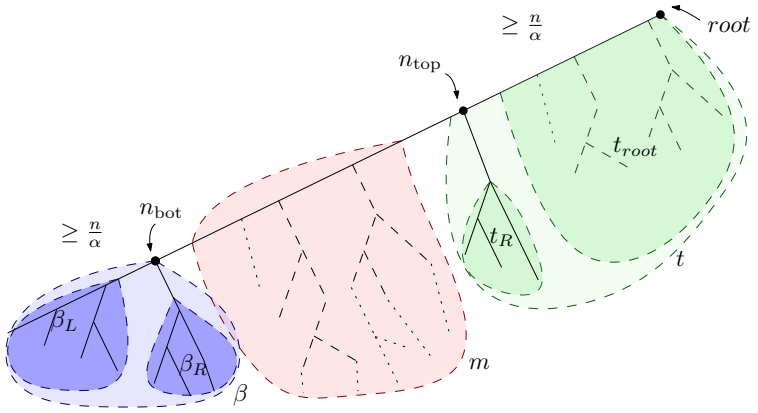


Figure 2.4: A left-heavy tree drawing displaying node n_{top}, node n_{bot} and the relevant partitions

probability of decreasing the problem size by at least a constant factor for each of its recursive calls. For analysis purposes, we consider the left-heavy representation of each tree, in which the tree rooted at the left child of any node is always at least as big as the tree rooted at its right child. (See Figure 2.4.) Using this view, we can characterize when a partition determines a "good split" and provide bounds on the sizes of the partitions, as follows.

Lemma 2.2. *With probability of at least* $\frac{\alpha-5}{2\alpha^2}$*, a round of Algorithm 1 decreases the problem size of any recursive call by at least a factor of* $\frac{\alpha-1}{\alpha}$*, for a constant* $\alpha > 5$*, thus experiencing a* ***good split****.*

Proof. Define the ***spine*** to be all nodes on the path from the root to its left-most leaf in a left-heavy drawing. Let n_{bot} be the bottom-most node on the spine that has at least n/α of the nodes in its sub-tree. Conversely, let n_{top} be the parent of the top-most spine node that has at most $(1 - 1/\alpha) \cdot n$ descendants. (See Figure 2.4.) Consider the three resulting trees obtained from separating at the incoming edge to n_{bot} and the outgoing edge from n_{top} to its left child. As shown in Figure 2.4, let t be the resulting tree retaining the root, β the tree rooted at n_{bot} and m the tree between n_{top} and n_{bot}. Within β let β_L and β_R be the trees rooted at the left and right child of n_{bot}. Similarly, for t, let t_R be the tree rooted at the right child of n_{top}. Finally, let t_{root} be the remaining tree when cut at n_{top}.

Consider the size of tree β, $|\beta|$, since this is the first tree rooted in the spine with over n/α nodes, then β_L must have had strictly under n/α nodes. Since the trees are in left-heavy order, β_R can have at most as many nodes as β_L so $\frac{n}{\alpha} \leq |\beta| < \frac{2n}{\alpha}$. Furthermore, we know that $|\beta|+|m|+|t_R|+|t_{root}|+1 = n$. Due to the left-heavy order, $|t_R| \leq |\beta|+|m|$. By definition of n_{top}, it's necessary that $|t_{root}| < \frac{n}{\alpha}$, thus $2(|\beta|+|m|)+1 > n-\frac{n}{\alpha}$ and $|\beta|+|m| > \left(\frac{\alpha-1}{2\alpha}\right)n - \frac{1}{2}$. Using the previous inequality and $|t| = n - |\beta| - |m|$, we find $|t| < \left(\frac{\alpha+1}{2\alpha}\right)n + \frac{1}{2}$. Also, $|m| = n - |t| - |\beta|$, so $|m| > n - \frac{5n+\alpha n}{2\alpha}$.

$$\frac{n}{\alpha} \leq |\beta| < \frac{2n}{\alpha}, \qquad \frac{n}{\alpha} \leq |t| < \left(\frac{\alpha+1}{2\alpha}\right)n + \frac{1}{2}, \qquad \left(\frac{\alpha-5}{2\alpha}\right)n < |m| \leq \left(\frac{\alpha-2}{\alpha}\right)n \quad (2.1)$$

Picking a leaf from β and another from m guarantees that $\beta \subseteq (A \cup \{a\})$ and $t \subseteq R$. Thus, using Equation (2.1), each of the three sub-problem sizes, $|A \cup \{a\}|$, $|B \cup \{b\}|$, and $|R|$, will be at most $\left(\frac{\alpha-1}{\alpha}\right)n$, when $\alpha > 5$. $Pr[\text{good split}] \geq Pr[\text{leaf in } \beta] \cdot Pr[\text{leaf in } m]$. Asymptotically, $Pr[\text{leaf in } \beta] \approx Pr[\text{node in } \beta]$, thus $Pr[\text{good split}] > \frac{\alpha-5}{2\alpha^2}$, which established the lemma. □

Returning to the proof of Theorem 2.1, let $p = \frac{\alpha-5}{2\alpha^2}$. From Lemma 2.2, we expect it will take $1/p$ rounds to obtain a good split. Every good split will reduce the problem-size by at least a constant factor, $\frac{\alpha-1}{\alpha}$. Thus, we are guaranteed to have just a single node left after we get $N_{\text{SPLITS}} = \log_{\frac{\alpha}{\alpha-1}}(n)$ good splits. Consider the geometric random variable, X_i, describing the number of rounds required to obtain the i-th good split, then $X = X_1 + \ldots + X_{N_{\text{SPLITS}}}$ describes the total number of rounds required by the algorithm. By linearity of expectation, $E[X] = \frac{2\alpha^2}{(\alpha-5)} \cdot N_{\text{SPLITS}} = \frac{2\alpha^2}{(\alpha-5)} \cdot \log_{\frac{\alpha}{\alpha-1}}(n)$. Therefore, since $\alpha > 5$ is a constant, this already implies an expected $O(\log n)$ rounds for Algorithm 1. Moreover, by a Chernoff bound for the sum of independent geometric random variables (see [60, 89]), $Pr\left[X > C \cdot E[X]\right] \leq e^{\frac{-(C-1)\cdot p \cdot N_{\text{SPLITS}}}{5}}$ for any constant $C \geq 3$ and constant $\alpha > 5$. Thus, the probability that we

take over $C \cdot E[X]$ rounds is $O(1/n^{C-1})$. Therefore, by a union bound across the n leaves, our algorithm completes in $O(\log n)$ rounds w.h.p. □

Corollary 2.1. *Algorithm 1 is optimal when asking $\theta(n)$ queries per round.*

The query complexity of Algorithm 1 matches an $\Omega(n \log n)$ lower bound for $Q(n)$, due to Kannan *et al.* [73]. Besides, we need $\Omega(\log n)$ rounds if we have $\theta(n)$ processors; hence, the round complexity of Algorithm 1 is also optimal.

2.4 Experiments

Given that our algorithm is randomized and performs optimally with high probability, we carried out experiments to analyze the practicality of our algorithm and compared their performance with the best known algorithms for reconstructing rooted trees.

In order to assess the practical performance of Algorithm 1, we performed experiments using synthetic phylogenetic trees and real-world biological phylogenetic trees from the phylogenetic library TreeBase [97], which is a database of biological phylogenetic trees, comprising over 100,000 distinct taxa in total.

We implemented an oracle interface, instantiated it with the relevant trees, and implemented our algorithm along with two other phylogenetic tree reconstruction algorithms that use relative-distance queries. The first is by Emamjomeh-Zadeh and Kempe [48], which is a randomized sequential divide-and-conquer algorithm. The second is by Kannan *et al.* [73], where they use a sequential deterministic procedure reminiscent of insertion-sort. All three algorithms achieve the optimal asymptotic query complexity of $\Theta(n \log n)$ in expectation.

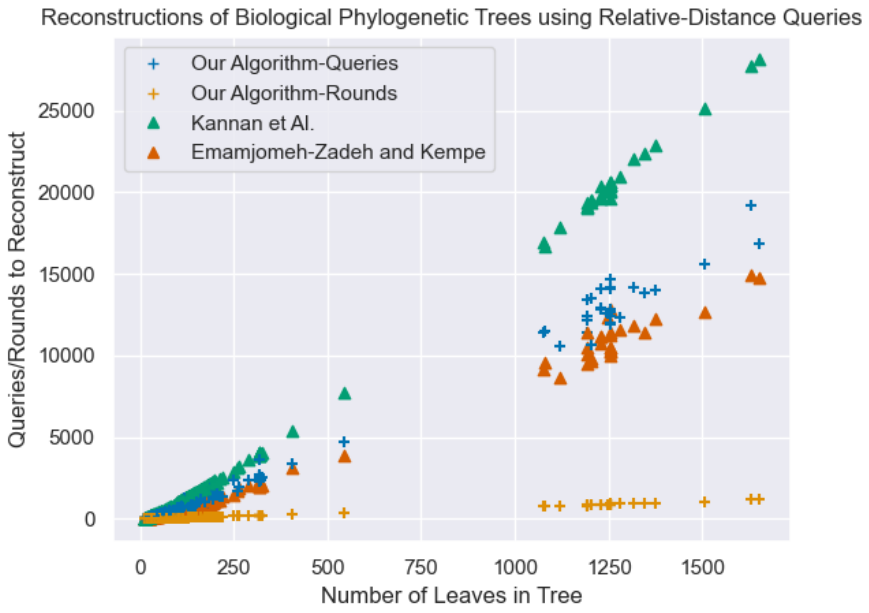


Figure 2.5: A scatter plot showing the number of queries and rounds for each of the three tree reconstruction algorithms for real trees from TreeBase. Since our algorithm is parallel, we include round complexity to serve as a comparison for the sequential complexity.

2.4.1 Real Data.

We instantiated our oracles with 1,220 biological phylogenetic trees from the TreeBase collection and used them to run all three algorithms. The results, shown in Figure 2.5, suggest that our algorithm outperforms the algorithm by Kannan *et al.*, both in terms of its round complexity and query complexity. However our algorithm almost matches Emamjomeh-Zadeh and Kempe's in terms of total queries and we believe the small difference is a direct result of the cost incurred while parallelizing the link step of Algorithm 1. It remains clear that Algorithm 1 outperforms the two other algorithms when considering the parallel speed-up.

2.4.2 Synthetic Data.

We also tested this algorithm using synthetic data and found similar results, detailed in Figure 2.6. In order to generate random instances of trees with maximum degree, d, we

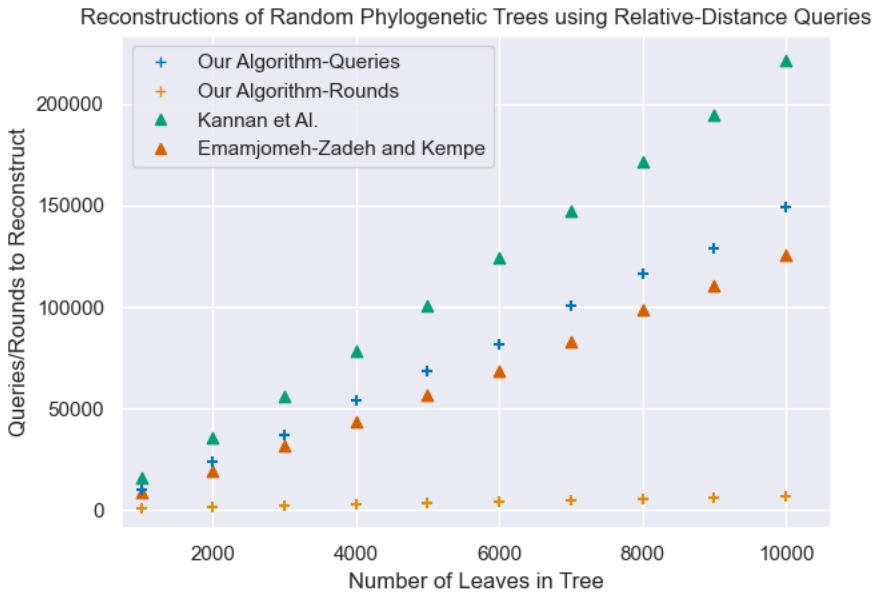


Figure 2.6: A plot showing the average number of queries and rounds for each of the three tree reconstruction algorithms. Each data point represents the average for 10 randomly generated trees.

synthesized a data set of random degree-d trees of n nodes for different values of n and $d = 2$. To generate a random tree, T, for a given n and d, we first generated a random Prüfer sequence [100] of a labeled tree, which defines a unique sequence associated with that tree, and converted it to its associated tree. In particular, each n-node tree $T = (V, E)$ has a unique code sequence $s_1, s_2, \ldots, s_{n-2}$, where $s_i \in V$ for all $1 \leq i \leq n-2$ and every node $v_i \in V$ of degree d_i appears exactly $d_i - 1$ times in this sequence. Therefore, in order to generate random degree-d rooted trees we generate a random Prüfer sequence while imposing conditions that: (i) all vertices appear at most $d-1$ times in the code and (ii) there is at least one node such that it appears exactly $d-1$ times. Converting each such sequence to its associated tree gave us a random degree-d tree instance that we used in our experiments. Given our algorithm's strict focus on biological phylogenetic trees, we use only full binary trees, where each internal node has exactly two children.

Chapter 3

Concatenation Arguments and their Applications on Counting Polyominoes and Polycubes

3.1 Introduction

A ***polycube*** of size n is a connected set of n cells on the cubical lattice $\mathbb{Z}^d$, where connectivity is through $(d-1)$-dimensional facets of the cells. Two-dimensional polycubes are also called ***polyominoes***. Two ***fixed*** polycubes are said to be ***equivalent*** if one can be transformed into the other by a translation (as opposed to ***free*** polycubes, of which two instances are equivalent if one can be transformed into the other by a translation, rotation, and/or mirroring). In the sequel, we consider only fixed polycubes, hence we simply call them "polycubes." Counting polyominoes and polycubes is a long-standing problem: Their study began in the 1950s in statistical physics [30, 110], where they are

usually called ***lattice animals***.[1] In statistical physics, lattice animals play a significant role in percolation processes and in the collapse transition that happens when branched polymers are heated. Counts of animals and formulae for specific types of animals form the basis for the mathematical models that describe these physical processes.

Let $A_d(n)$ denote the number of d-dimensional polycubes of size n. The sequence $(A_2(n))$ is currently known up to $n = 56$ [68]. The growth constants (asymptotic growth rates) of polyominoes and polycubes have also attracted much attention. Klarner [78] showed that $\lambda_2 := \lim_{n\to\infty} \sqrt[n]{A_2(n)}$ exists. The convergence of $A_2(n+1)/A_2(n)$ to λ_2 (as $n\to\infty$) was proven only three decades later by Madras [85], using a novel pattern-frequency argument. (These results easily extend to any dimension.) The currently best-known lower [18] and upper [19] bounds on λ_2 are 4.0025 and 4.5252, respectively. (The previous upper bound, 4.6496 [79], which has been the best known for almost half a century, was due to Klarner and Rivest.) It is widely ***believed*** (see, ***e.g.***, [56, 57]), but has never been proven rigorously, that $\lambda_2 \approx 4.06$. The best estimate, $\lambda_2 = 4.0625696 \pm 0.0000005$, is currently due to Jensen [68].

In this chapter, we extend the notion of a "concatenation argument" and develop methods for deriving lower and upper bounds on the growth constants of families of polyominoes and polycubes whose enumerating sequences are so-called quasi sub- or super-multiplicative. We demonstrate various applications of this technique to setting new bounds on the growth constants of all fixed polyominoes and polycubes, as well as of specific families, such as tree polycubes and convex polyominoes.

[1] More precisely, polycubes are called ***strongly-embedded site animals*** in this literature. This terminology comes from the dual graph of the square lattice (which is also a square lattice). In the dual graph, a polyomino cell becomes a vertex (site), and a neighborhood relation between two polyomino cells becomes an edge (bond). In the dual notation, the size of a "site animal" is the number of its vertices, while the size of a "bond animal" is usually the number of its edges. (Both types of animals should still be represented by connected graphs.) In addition, an edge always belongs to a "strongly-embedded" animal if its two endpoints are vertices of the animal, while this is not necessarily the case for "weakly-embedded" animals. For example, the 2×2 polyomino P=  corresponds to one strongly-embedded site animal of size 4 ( , the dual of P), but to five (5) weakly-embedded site animals of the same size (,  ,  ,  , and ). As bond animals, these animals have different sizes.

The chapter is organized as follows. In Section 3.2, we introduce a few preliminary notions and describe the basic concatenation argument. In Section 3.3, we provide several more complex concatenation methods and analyze the growth constants they imply. Then, we give some simple applications of these methods in Section 3.4. In Section 3.5, we show how to apply our methods in a recursive manner, setting new bounds on the growth constants of polycubes in three and higher dimensions. Finally, in Section 3.6, we show a more involved application to convex polyominoes. We end in Section 3.7 with some concluding remarks.

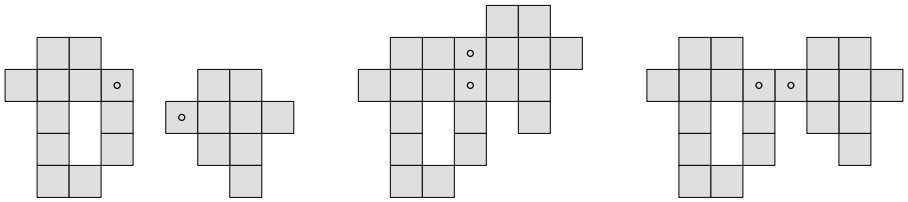
3.2 Preliminaries

3.2.1 Concatenation and Super-/Sub- Multiplicative Sequences

A sequence $(Z(n))$ is called ***super-multiplicative*** (resp., ***sub-multiplicative***) if $Z(m+n) \geq Z(m)Z(n)$ (resp., $Z(m+n) \leq Z(m)Z(n)$) for all $m, n \in \mathbb{N}$.[2] It is known [99, p. 171] (using a lemma attributed to Michael Fekete by Klarner [78]) that a super-multiplicative (resp., sub-multiplicative) sequence $Z(n)$, with the property that the sequence $(Z'(n)) = (\sqrt[n]{Z(n)})$ is bounded from above (resp., below), has a ***growth constant***. That is, the quantity $\lim_{n\to\infty} Z'(n)$ exists.

Let us define a total lexicographic order on cells of the cubical lattice: First by x_1 (x in two dimensions), then by x_2 (y in two dimensions), and so on. Thus, in two dimensions, the ***smallest*** (resp., ***largest***) square of a polyomino P is the lowest (resp., highest) cell in the leftmost (resp., rightmost) column of P. The vertical (resp., horizontal) ***concatenation*** of two polyominoes P_1, P_2 is the positioning of P_2 such that its smallest cell lies immediately ***above*** (resp., ***to the right of***) the largest cell of P_1 (see Figure 3.1). Similarly, two d-dimensional polycubes can be concatenated in d ways. Obviously, concatenating two

[2] Whether or not the relation is strict is insignificant since the asymptotic results are identical.



(a) Two polyominoes (b) Vertical (c) Horizontal

Figure 3.1: Concatenations of two polyominoes.

polycubes always yields a valid polycube (connected and with no overlapping cells), and two different pairs of polycubes of sizes m, n always yield by concatenation two different polycubes of size $m + n$. (The first claim follows at once from the lexicographic order of cells. The second claim can easily be justified by observing the smallest cell in which the two concatenated polycubes differ.) Many polycubes, however, can be represented as the concatenations of several pairs of polycubes (of different sizes), whereas others (***e.g.***, ⊞) cannot be represented at all as the concatenation of smaller polycubes.

The following is a folklore concatenation argument for polyominoes, setting a (rather weak) lower bound on their growth constant. A direct consequence of the discussion above is that $A_2^2(n) < A_2(2n)$. That is, $\sqrt[n]{A_2(n)} < \sqrt[2n]{A_2(2n)}$. Hence, a sequence of the form $A^* = \left(\sqrt[n_0 2^i]{A_2(n_0 2^i)}\right)_{i=0}^{\infty}$ is monotone increasing for any natural number n_0. Since the entire sequence $A = (A_2(n))$ is super-multiplicative, and the sequence $A' = \left(\sqrt[n]{A_2(n)}\right)$ is bounded from above [**?**], the sequence A has a growth constant λ_2. Obviously, every subsequence of A' also converges to λ_2. In addition, since any such subsequence A^* is monotone increasing, any element of it, $\sqrt[n_0]{A_2(n_0)}$, is a lower bound on λ_2. Empirically, the best (largest) lower bound is obtained this way by setting $n_0 = 56$ (the largest value of n for which $A_2(n)$ is known), yielding the lower bound $\lambda_2 > 3.7031$.

Remarks.

1. Although A^* is a subsequence of A', the monotonicity of the former does ***not*** imply the monotonicity of the latter. Nevertheless, the known values of $A_d(n)$ (in all dimensions) ***suggest*** that A' be monotone increasing too. We will later refer to this observed property as "the unproven monotonicity of the root sequence" ("Conjecture $\sqrt{}$UM," in short).

2. A stronger observed phenomenon is the monotonicity of the ***ratio*** sequence, ***i.e.***, $A_d(n)/A_d(n-1) < A_d(n+1)/A_d(n)$ (for $n \geq 2$). Equivalently, $A_d^2(n) < A_d(n+1)A_d(n-1)$, and in a more general form, $A_d^2(n) < A_d(n+k)A_d(n-k)$ (for $1 \leq k < n$).

3. Yet another observation is that the ratio sequence dominates the root sequence, that is, $\sqrt[n]{A_d(n)} < A_d(n)/A_d(n-1)$ for all $n > 1$, and in a more general form (using the convention $A(0) = 1$), $\sqrt[n]{A_d(k)A_d(n-k)} < A_d(n-k)/A_d(n-1-k)$ (for $n > k \geq 0$). We will later refer to this property as "the unproven dominance of the ratio sequence over the root sequence" ("Conjecture $\dot{\sqrt{}}$UD," in short).

4. Property (2) implies both Properties (1) and (3). If all three unproven properties are true, then the ratio sequence $(A_d(n+1)/A_d(n))$ converges to λ_d faster than the root sequence $(\sqrt[n]{A_d(n)})$, which is widely believed to be the case.

At any rate, all these observations were never proven and are stated here as conjectures. In this work, we do not rely on them to be true.

Similarly, for any sub-multiplicative sequence $(B(n))$, for which $B'(n) = \sqrt[n]{B(n)}$ is bounded from below, and for which we can show that $B'(n) \geq B'(2n)$ for any natural number n, we conclude that any known value $B(n_0)$ sets the upper bound $B'(n_0)$ on the growth constant of $(B(n))$.

3.2.2 Quasi Super- and Sub-Multiplicativity

A sequence $(Z(n))$ is ***quasi super-multiplicative*** (resp., ***quasi sub-multiplicative***) if $Z(m+n) \geq P(m+n)Z(m)Z(n)$ (resp., $Z(m+n) \leq P(m+n)Z(m)Z(n)$), for any $m, n \in \mathbb{N}$ and for some positive function $P(x)$ which is sub-exponential with x. (Hereafter, we will consider cases in which this function is a polynomial.) Sometimes, the sequence enumerating a family (subset) of polycubes is either quasi super-multiplicative or quasi sub-multiplicative, or even both (naturally, with different auxiliary functions $P(\cdot)$). In such cases, we can take advantage of the fact that $\lim_{n\to\infty} \sqrt[n]{P(n)} = 1$ and obtain bounds on the growth constant of $Z(n)$ from known values of $Z(n)$. In favorable cases, where we have as many values of the enumerating sequence as we want (either by a closed-form or a recursive formula), this method approaches the growth constant as close as we want.

Theorem 3.1. *Assume that the limit* $\mu := \lim_{n\to\infty} \sqrt[n]{Z(n)}$ *exists for a sequence* $(Z(n))$. *Let* $c_1 \neq 0, c_2, c_3$ *be some constants. Then:*

(a) (multiplicative polynomial) If $c_1 n^{c_2} Z^2(n) \leq Z(2n)$ *(resp.,* $c_1 n^{c_2} Z^2(n) \geq Z(2n)$*)* $\forall n \in \mathbb{N}$, *then* $\sqrt[n]{c_1 (2n)^{c_2} Z(n)} \leq \mu$ *(resp.,* $\sqrt[n]{c_1 (2n)^{c_2} Z(n)} \geq \mu$*)* $\forall n \in \mathbb{N}$.

(b) (index shift) If $c_1 Z^2(n + c_3) \leq Z(2n)$ *(resp.,* $c_1 Z^2(n + c_3) \geq Z(2n)$*)* $\forall n \in \mathbb{N}$, *then* $\sqrt[n]{c_1 Z(n + 2c_3)} \leq \mu$ *(resp.,* $\sqrt[n]{c_1 Z(n + 2c_3)} \geq \mu$*)* $\forall n \in \mathbb{N}$. *Equivalently, if* $c_1 Z^2(n) \leq Z(2n+c_3)$ *(resp.,* $c_1 Z^2(n) \geq Z(2n+c_3)$*)* $\forall n \in \mathbb{N}$, *then* $\sqrt[n]{c_1 Z(n - c_3)} \leq \mu$ *(resp.,* $\sqrt[n]{c_1 Z(n - c_3)} \geq \mu$*)* $\forall n > c_3$.

Proof. Our strategy is to manipulate the given relation, in both cases, and reach a relation of the form $\zeta(n) \leq \zeta(2n)$ or $\zeta(n) \geq \zeta(2n)$. Then, we follow closely the logic of the basic argument described in Section 3.2.1.

(a) By straightforward manipulations of the relation $c_1 n^{c_2} Z^2(n) \leq Z(2n)$ (resp., $c_1 n^{c_2} Z^2(n) \geq Z(2n)$), we obtain that $\sqrt[n]{c_1 (2n)^{c_2} Z(n)} \leq \sqrt[2n]{c_1 (4n)^{c_2} Z(2n)}$ (resp., $\sqrt[n]{c_1 (2n)^{c_2} Z(n)} \geq$

$\sqrt[2n]{c_1(4n)^{c_2}Z(2n)}$). Then, by setting $\zeta(n) = \sqrt[n]{c_1(2n)^{c_2}Z(n)}$, we see that $\zeta(n) \leq \zeta(2n)$ (resp., $\zeta(n) \geq \zeta(2n)$). It follows that the subsequence $(\zeta(2^{i+1}n_0))_{i=0}^{\infty}$ is monotone increasing (resp., decreasing) and converging to μ, for any natural number n_0. The claim follows.

(b) In this case, we substitute $n := m + c_3$ in the relation $c_1Z^2(n + c_3) \leq Z(2n)$ (resp., $c_1Z^2(n + c_3) \geq Z(2n)$) and perform manipulations as in Case (a), obtaining that $c_1^2Z^2(m + 2c_3) \leq c_1Z(2m + 2c_3)$ (resp., $c_1^2Z^2(m + 2c_3) \geq c_1Z(2m + 2c_3)$). Hence, $\sqrt[m]{c_1Z(m + 2c_3)} \leq \sqrt[2m]{c_1Z(2m + 2c_3)}$ (resp., $\sqrt[m]{c_1Z(m + 2c_3)} \geq \sqrt[2m]{c_1Z(2m + 2c_3)}$). Elementary calculus shows that the limits of the two sequences $\left(\sqrt[m]{c_1Z(m)}\right)$ and $\left(\sqrt[m]{c_1Z(m + 2c_3)}\right)$, as $m \to \infty$, are equal. (Indeed, since $\lim_{m\to\infty} \frac{Z(m+1)}{Z(m)} = \mu$, there exists some constant δ, such that $\frac{Z(m+1)}{Z(m)} \leq \delta$, for all $m \in \mathbb{N}$. Hence, $c_1Z(m) \leq c_1Z(m + 2c_3) \leq c_1\delta^{2c_3}Z(m)$, and, thus, by the squeezing theorem, $\lim_{m\to\infty} \sqrt[m]{c_1Z(m + 2c_3)} = \mu$.) Finally, we fix $\zeta(m) = \sqrt[m]{c_1Z(m + 2c_3)}$ and continue as in Case (a). The equivalent case, in which the shift appears in the right side of the relation, is treated analogously by substituting $n := m - c_3$.

□

Remark. In fact, reaching any relation of the form $\zeta(n) \leq \zeta(f(n))$, where $n < f(n)$ for all $n \in \mathbb{N}$ (for some integer function $f(n)$), is sufficient for our purposes. Moreover, the requirement that these relations hold for ***all*** $n \in \mathbb{N}$ can be relaxed to that they hold only for all $n \geq n_0$, for some $n_0 \in \mathbb{N}$, in which case known values of only the n_0th element or later in the sequence can be used for obtaining the bounds.

3.3 Methods of Concatenation

We list below the main concatenation variants we employ. For ease of exposition, we use relations that yield, using Theorem 3.1, ***lower*** bounds on the growth constants of polycubes. However, the same methods can be used for setting upper bounds as well. Consistently with Conjecture $\sqrt{}$UM, we observed that in all dimensions, the best (largest) lower bounds on λ_d were obtained by using the largest known element of $A_d(n)$. Let a_1, a_2 be the lexicographically largest and smallest cells, respectively, of the two concatenated polycubes P_1, P_2. We denote the enumerating sequence by $Z(n)$ and its growth constant by λ_Z.

[E] The most elementary method of concatenation, and surprisingly also the most dominant variant in the literature, attaches cell a_1 to cell a_2 in a ***single*** way. This leads to the relation $Z^2(n) \leq Z(2n)$, which, as detailed above (see Section 3.2.1), implies that $\lambda_Z \geq \sqrt[n]{Z(n)}$ for all $n \in \mathbb{N}$.

[C] A simple improvement on Method [E] is achieved by considering all possible (lattice dependent) c ways of attaching a_1 to a_2, such that a_1 is lexicographically smaller than a_2. (For example, two d-dimensional polycubes can be concatenated in d ways as shown for two dimensions in Figure 3.1.) This leads to the relation $c\,Z^2(n) \leq Z(2n)$, which, by Theorem 3.1(a), implies that $\lambda_Z \geq \sqrt[n]{c\,Z(n)}$ for all $n \in \mathbb{N}$.

[M] A possible improvement on Method [C] could be obtained by considering all possible polycubes of size k for being concatenated in between P_1 and P_2. As in Method [C], assume that there are c ways for attaching two cells lexicographically. This leads to the relation $c^2 Z(k) Z^2(n) \leq Z(2n+k)$, which, by Theorem 3.1(b), implies that $\lambda_Z \geq \sqrt[n]{c^2 Z(k) Z(n-k)}$ for all $k, n \in \mathbb{N}$, such that $n > k$.

Remark. It is initially unknown which values of k, n lead to the best (largest) lower bound on λ_Z. It is ***a priori*** not even known, for a fixed value of k, which

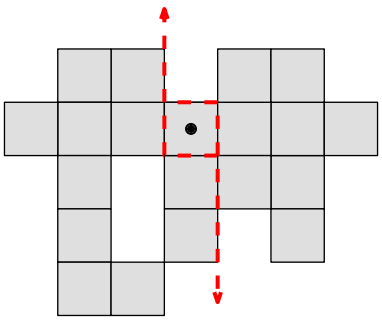


Figure 3.2: Overlapping a cell.

value of n yields the best bound. Empirically, for d-D polycubes, the best bound on λ_d is obtained by choosing the largest k for which $A_d(k)$ is known, and by setting $n := 2k$. This empirical fact is consistent with Conjecture $\vdots\sqrt{\text{UD}}$. Indeed, assume that $\sqrt[n]{A_d(k)A_d(n-k)} < A_d(n-k)/A_d(n-1-k)$. After a few simple manipulations, we see that $A_d(k)A_d(n-1-k) < (A_d(k)A_d(n-k))^{(n-1)/n}$. By taking the $(n-1)$st root on both sides, we conclude that $\sqrt[n-1]{A_d(k)A_d(n-1-k)} < \sqrt[n]{A_d(k)A_d(n-k)}$, which confirms the empirical findings. However, by setting $n = 2k$, we find that $\sqrt[n]{d^2 A_d(k)A_d(n-k)} = \sqrt[2k]{d^2 A_d^2(k)} = \sqrt[k]{d\,A_d(k)}$, resulting in the same bound as in Method [C].

[O] A slightly different method (see Figure 3.2) is based on ***overlapping*** cells a_1 and a_2. This operation always yields a valid polycube because every remaining cell in P_1 is smaller than a_1, and every remaining cell in P_2 is larger than a_2, so there are no other overlapping cells. In addition, all pairs of polycubes necessarily generate unique polycubes by this method of concatenation. This leads to the relation $Z^2(n) \leq Z(2n-1)$, which, by Theorem 3.1(b), implies that $\lambda_Z \geq \sqrt[n]{Z(n+1)}$ for all $n \in \mathbb{N}$.

[MO] A possible improvement on Method [O] can be obtained by concatenating P_1 and P_2 through all possible polycubes of size k, where P_1 overlaps the middle polycube's smallest cell, and P_2 overlaps the middle polycube's largest cell. This leads to the relation $Z(k)Z^2(n) \leq Z(2n+k-2)$, which, by Theorem 3.1(b), implies that $\lambda_Z \geq \sqrt[n]{Z(k)Z(n-k+2)}$ for all $k, n \in \mathbb{N}$, such that $n > k-2$.

Remark. In practice, for d-D polycubes, we observed (as in Method [M]) that the best (largest) lower bound on λ_d was obtained by setting $k = 1$ and by choosing the largest n for which $A_d(n+1)$ is known, yielding the bound $\sqrt[n]{A_d(1)A_d(n+1)}$. For polycubes, we have that $A_d(1) = 1$ (for any d), hence, this bound is precisely the bound obtained by Method [O].

In many cases, one can identify a family F, a ***subset*** of animals on the original lattice, which is closed under concatenation. Clearly, by simple calculus, if the family F has a growth constant λ_F of its own, then $\lambda_Z \geq \lambda_F$. This idea can be combined with any of the methods listed above.

In the following sections, we provide several applications of the listed concatenation methods. In some cases, we apply additional ***ad hoc*** tricks.

3.4 Simple Applications

We now provide a few simple applications of the basic concatenation methods.

3.4.1 General

We can directly apply concatenation methods [E], [C], and [O] to polycubes, and find lower bounds on λ_d. Table 3.2 (found in Section 3.5) shows the best lower bounds obtained in dimensions 2–9, all using the largest known values of the respective sequences. For example, for polyominoes, Methods [E] and [O] provide, both using $n = 56$, the lower bounds $\sqrt[56]{A_2(56)} \approx 3.7031$ and $\sqrt[55]{A_2(56)} \approx 3.7923$, respectively, on λ_2. These bounds are inferior to the best known bound $\lambda_2 > 4.00253$ [18], obtained by the much stronger method of ***twisted cylinders***. However, the cylinders method cannot be generalized efficiently to

higher dimensions because it becomes computationally intractable. (This is because the main factor that affects its running time and consumed memory, namely, ***boundaries*** of polycubes, outnumbers polyominoes when the method is applied to polycubes in three or higher dimensions.) As we show in Section 3.5, we can improve further all lower bounds in $d \geq 3$ dimensions by applying a recursive technique.

3.4.2 Trees

A polycube is a ***tree*** if its cell-adjacency graph (its "dual") is acyclic. Tree polycubes and their respective growth constants are significant on their own, and also attracted interest in the literature (see, ***e.g.***, [47, 69]). Naturally, any lower bound on $\lambda_{d;T}$, the growth constant of d-dimensional tree polycubes, is also a lower bound on λ_d. In order to preserve the tree property, special restrictions must be enforced while concatenating them. Figures 3.3(a,b) show two pairs of tree polycubes T_1, T_2, in two and three dimensions, respectively, having cells σ_1, σ_2 as their largest and smallest cells, respectively, concatenated by Method [E]. To guarantee that the concatenation of the trees T_1 and T_2 is itself a tree (***i.e.***, that it introduces exactly one edge in the cell-adjacency graph), cells σ_1 and σ_2 must touch via their maximum, respectively minimum, faces orthogonal to the most dominant axis of the lexicographic order. This leaves a $(d-1)$-dimensional buffer space, in which any cell would be larger than σ_1 and smaller than σ_2, making it impossible for any cell in T_1 or T_2 to occupy this space. On the other hand, as can be seen in the figure, concatenating T_1 and T_2 along any other axis risks that the original trees also touch along the most dominant axis. This is demonstrated by the simple example shown in Figure 3.3(d). Concatenating along axis x_1 (Figure 3.3(e)) yields a tree, while concatenating along axes x_2 or x_3 (Figures 3.3(f,g), respectively) introduces a cycle in the cell-adjacency graph.

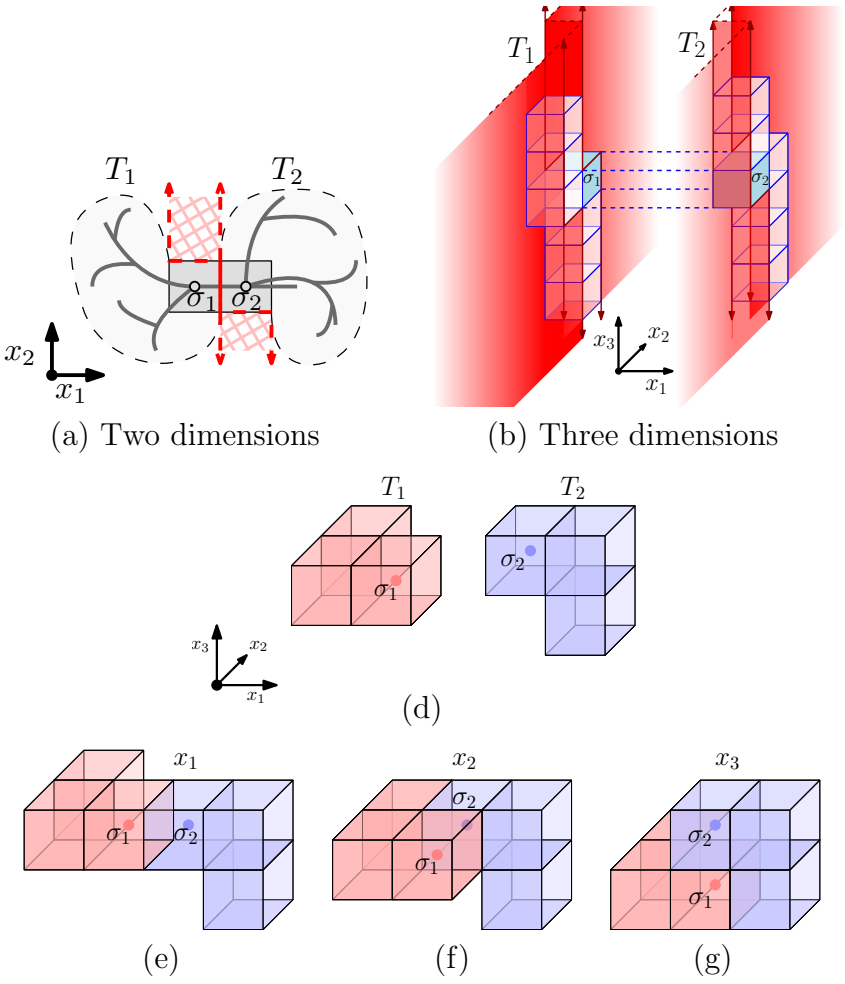


Figure 3.3: Concatenating trees.

Table 3.1: ***Lower*** bounds on the growth constants of tree polycubes of various dimensions.

Dimensions	Known Values	OEIS Sequence	Method [E]
2	44	A066158	3.4045
3	17	A118356	5.5592
4	10	A191094	6.7698
5	10	A191095	8.8035
6	8	A191096	9.4576
7	7	A191097	10.0909
8	7	A191098	11.4891

Let $A_{d;T}(n)$ denote the number of n-cell tree polycubes in d dimensions, and let $\lambda_{d;T}$ denote their respective growth constants. As in similar examples, we obtain the relation $A_{d;T}^2(n) \leq A_{d;T}(2n)$, which evaluates to $\lambda_{d;T} \geq \sqrt[n]{A_{d;T}(n)}$ for all $n \in \mathbb{N}$. For example, for tree polyominoes (in two dimensions), the best (largest) lower bound is obtained with $n = 44$,[3] implying that $\lambda_{2;T} \geq \sqrt[44]{A_{2;T}(44)} \geq 3.4045$. Table 3.1 shows the best lower bounds obtained this way in dimensions 2–8, in all cases using the largest known values of the respective sequences.

3.5 Recursive Bounding

In this section, we present a recursive scheme for improving the bounds obtained by all concatenation methods described in Section 3.3. Let us demonstrate the scheme by a concrete example which aims at setting a lower bound on the growth constant of polycubes, but the general scheme is also suited for obtaining upper bounds.

As observed in previous sections, the sequence enumerating d-dimensional polycubes, $(A_d(n))$, is super-multiplicative and it has a growth constant λ_d, hence, by concatenation Method [O], any term of the form $\sqrt[n-1]{A_d(n)}$ is a lower bound on λ_d. In practice, we can

[3]The currently largest known element of $(A_{2;T}(n))$ [67] is $A_{2;T}(44) = 257{,}571{,}182{,}441{,}471{,}056{,}810{,}356$.

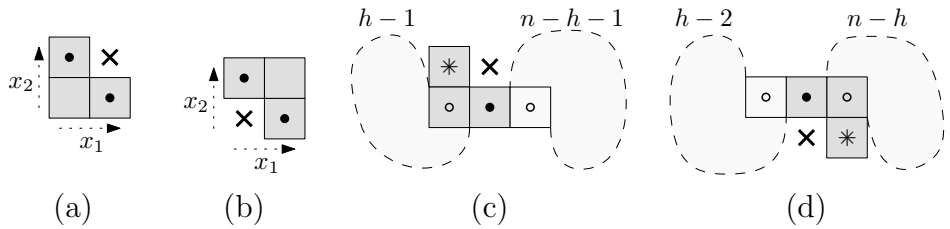


Figure 3.4: Constructions for the proof of Theorem 3.2.

prove relations which are tighter than the super-multiplicativity condition by considering more combinations of polycubes than just configurations obtained by concatenation. Here is an example.

Theorem 3.2. *Let $h = \lfloor (n+1)/2 \rfloor$. Then, for every $n \geq 4$, we have that*

$$A_d(n) \geq A_d(h)A_d(n-h+1) + \frac{d(d-1)^2}{2}(A_d(h-1)A_d(n-h-1) + A_d(h-2)A_d(n-h)). \tag{3.1}$$

Proof. The term on the left side of Relation 3.1 is the number of all d-D polycubes of size n, whereas the terms on the right count a subset of these polycubes.

Let us explain the right side of Relation 3.1. Our goal is to count three types of polycubes, and distinguish between them according to the connectedness of some of their cells.

The first term, $A_d(h)A_d(n-h+1)$, is the number of d-dimensional polycubes obtained by concatenating two polycubes of sizes h and $n-h+1$ with a 1-cell overlap (***i.e.***, Method [O]). Note that in these combinations, the lower h cells, as well the upper $n-h+1$ cells, form two valid polycubes which share the hth cell.

The second factor of the second term, $A_d(h-1)A_d(n-h-1) + A_d(h-2)A_d(n-h)$, counts two types of constructions. In both types, we place 3-cell L-shapes (shown in Figures 3.4(a,b)) in the middle in order to force the hth cell into a position where it is either no longer adjacent to any of the upper $n-h$ cells, or to any of the lower $h-1$ cells, as shown in Figures 3.4(c,d),

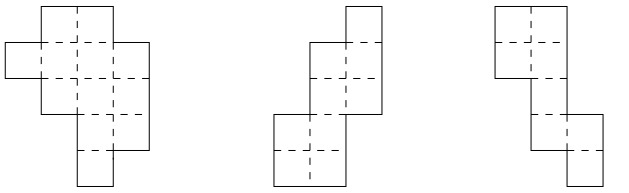
Table 3.2: Lower bounds on λ_d, through each method. (Best previously-published bounds are underlined, our improved bounds appear in bold.)

Dims.	Known Values	OEIS Sequence	Concatenation Methods [E]	[C]	[O]	Other Methods	Recursive Bounding
2	56	A001168	3.7031	3.7492	3.7923	<u>4.00253</u> [18]	3.7944
3	19	A001931	<u>6.0211</u>	6.3795	6.6526	—	**6.6621**
4	16	A151830	<u>8.4627</u>	9.2286	9.7576	—	**9.7714**
5	15	A151831	<u>10.9093</u>	12.1449	12.9398	—	**12.9569**
6	15	A151832	<u>13.5237</u>	15.2396	16.2888	—	**16.3087**
7	14	A151833	<u>15.5985</u>	17.9245	19.2690	—	**19.2927**
8	12	A151834	<u>16.6477</u>	19.7976	21.4975	—	**21.5298**
9	12	A151835	<u>18.8417</u>	22.6277	24.6060	—	**24.6416**

respectively. The largest and smallest cells of the lower and upper polycubes, respectively, are marked by empty circles, and the hth cell of the resulting polycube is marked by an asterisk. The trick is to mix between no-overlap and 1-overlap on the two sides of the L. To achieve this, the L-shape in Figure 3.4(a) is overlapped with the lower polycube of size $h-1$, and concatenated to the upper polycube of size $n-h-1$ (see Figure 3.4(c)). Similarly, the L-shape in Figure 3.4(b) is concatenated to the lower polycube of size $h-2$ and overlapped with the upper polycube of size $n-h$ (see Figure 3.4(d)).

Let us finally explain the first factor of the second term. First, there are $\binom{d}{2}$ options for choosing the orientation of L. (The chosen directions are denoted by x_1 and x_2, where x_1 has precedence over x_2 in the lexicographic order.) Second, the no-overlap concatenation (on one of the sides) can be done in $d-1$ ways: All directions are valid ***except*** direction x_2, the direction which would cover the forbidden cell (marked by an "$\times$" in Figures 3.4(a,b)). This constraint avoids multiple counting; otherwise, we would have in the middle a 2×2 square which can be created in more than one way. Overall, we have a factor of $\binom{d}{2}(d-1) = d(d-1)^2/2$.

It is easy to see that all resulting polycubes are different by construction. □



(a) Convex (b) Staircase

Figure 3.5: Convex and staircase polyominoes.

Relation (3.1) is disadvantageous in the sense that we cannot derive from it "chains" of lower bounds as we did for all concatenation methods. However, it allows us to apply a recursive procedure for bounding from below any value of $A_d(n)$. Since we know values of $A_d(n)$ for $1 \leq n \leq n_0$ (for some $n_0 \in \mathbb{N}$), we can construct a sequence $B(n)$, such that $B(n) \leq A_d(n)$ for every n: For $1 \leq n \leq n_0$, let $B(n) = A_d(n)$; and for $n > n_0$, set $B(n)$ recursively to the value calculated from the right side of Relation 3.1.

One can apply this method for large values of n ***ad infinitum***, or, more practically, until one's available computing resources are exhausted, and choose the best value encountered. We ran this procedure up to $n \approx 12 \cdot 10^6$ for $2 \leq d \leq 9$. In two dimensions, we obtained that $\lambda_2 \geq 3.7944$. This bound is slightly better than the one obtained by Method [O], but it could still not beat the bound $\lambda_2 > 4.0025$ [18] which was obtained by a stronger method. However, we improved the lower bounds on λ_d in ***all*** of $3 \leq d \leq 9$ dimensions. Table 3.2 summarizes the obtained bounds; new bounds appear in boldface.

3.6 Convex Polyominoes

We now provide a much more complex application of concatenation arguments, setting both lower and upper bounds on the growth constant of ***convex*** polyominoes (see Figure 3.5(a)).

Definition 3.1. *A polyomino is* ***convex*** *if any axis-parallel line intersects the polyomino in at most one continuous sequence of cells.*

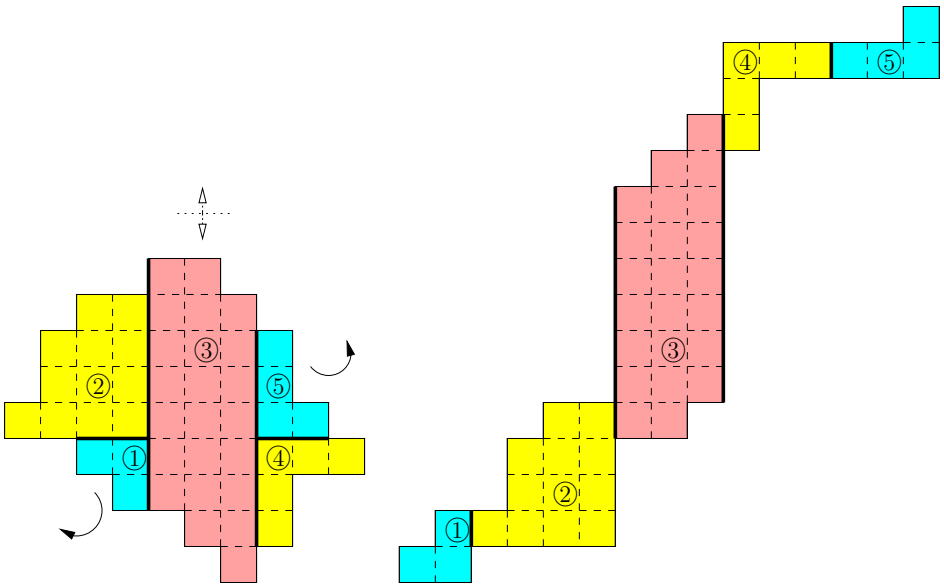
Alternatively, a polyomino is convex if its perimeter (in the usual sense–the length of its boundary curve) is equal to that of its minimum bounding box [29].

Let the symbol $C(n)$ denote the number of convex polyominoes of size n. Although a formula for $C(n)$ seems to be out of reach, Bousquet-Mélou and Fédou [28] developed a rather complex generating function for the enumerating sequence. Klarner and Rivest [80] showed that the growth constant $\lambda_{2;C} := \lim_{n\to\infty} \sqrt[n]{C(n)}$ exists, and provided almost-matching lower and upper bounds on it, both roughly equal 2.3091. Soon after, Bender [23] computed the constant $\lambda_{2;C}$ precisely. We show below how we obtain easily the same result by applying Theorem 3.1.

We begin by defining a subfamily of convex polyominoes (see Figure 3.5(b)).

Definition 3.2. *A polyomino is an* ***ascending*** *(resp.,* ***descending****)* ***staircase*** *(or* ***parallelogram****) if it is convex and its boundary contains the bottom-left and top-right (resp., top-left and bottom-right) corners of its minimal bounding box.*

Denote by $S(n)$ (resp., $S_{\text{all}}(n)$) the number of ascending (resp., all) staircase polyominoes of size n. It is rather easy to verify that $S_{\text{all}}(n) = 2S(n) - \sigma_0(n)$, where $\sigma_0(n)$ is the number of divisors of n (Sequence A000005 in the OEIS [107]). To see this, note that the family of convex polyomoinoes of size n, that are both ascending and descending, is precisely the set of rectangles whose height and widths are divisors of n. Since $2 \leq \sigma_0(n) < 2\sqrt{n}$ for all $n \in \mathbb{N}$, we have that $S_{\text{all}}(n) = \Theta(S(n))$, and, hence, we can investigate $S(n)$ instead of $S_{\text{all}}(n)$. Let us define $\lambda_{2;S} := \lim_{n\to\infty} \sqrt[n]{S(n)}$. Theorem 3.3 below states that this limit exists. Naturally, $S(n) \leq C(n)$ for all $n \in \mathbb{N}$, and the relation is strict for all $n \geq 3$. This implies that $\lambda_{2;S} \leq \lambda_{2;C}$. In fact, the two growth constants are equal, as is also implied by



(a) P (convex) (b) Q (staircase)

Figure 3.6: Converting a convex polyomino into an ascending staircase polyomino.

the next theorem. (This result was already proven by Klarner and Rivest [80, p. 34, Eq. 5] in a different way. We provide here an alternative proof in order to demonstrate the power of concatenation arguments.)

Theorem 3.3. *The quantity* $\lambda_{2;S} := \lim_{n\to\infty} \sqrt[n]{S(n)}$ *exists, and* $\lambda_{2;S} = \lambda_{2;C}$.

Proof. As mentioned above, we have that $S(n) \leq C(n)$ for all $n \in \mathbb{N}$. We show now that, in addition, $C(n) \leq 2(n+1)^6 S(n)$ for all $n \in \mathbb{N}$. Indeed, any n-cell convex polyomino P can be converted into an ascending staircase polyomino Q of the same size, and a code of up to $2(n+1)^6$ possible values, so that P can be reconstructed uniquely from Q and the code, as follows.

Refer to Figure 3.6. First, identify the widest staircase subpolyomino of P that extends the full height of P (marked as part ③, colored in pink), which is either ascending or descending. (The latter case is seen in the figure.) Then, out of the portion remaining on the left, identify the highest ascending part with a straight floor (part ②, colored in yellow), leaving a descending remainder (part ①, colored in cyan). Similarly, out of the remainder

of P on the right of part ③, identify the highest ascending part with a straight ceiling (part ④, colored in yellow), leaving a descending remainder (part ⑤, colored in cyan). Then, rotate by 90° parts ① and ⑤ counter-clockwise and clockwise, respectively, so as to make them ascending. In addition, if part ③ is descending, flip it upside-down for the same purpose. Finally, concatenate parts ①–⑤ horizontally in this order (say, in a horizontal manner) to obtain an ascending staircase polyomino.

The size of each of the parts ①–⑤ is in the range $[0, n]$. Hence, the split of Q into the original parts can be done in at most $(n+1)^4$ ways. (More precisely, the integer number n can be represented as the sum of five non-negative integer numbers in $\binom{n+4}{4}$ ways.) The locations of parts ①, ②, ④, ⑤ relative to part ③ can be coded in a total of at most n^2 ways. Whether or not part ③ was flipped requires one additional bit of information. Altogether, the entire process can be encoded in at most $2(n+1)^6$ ways. (Clearly, the actual number of possible codes is usually much smaller.) Therefore, $\sqrt[n]{S(n)} \leq \sqrt[n]{C(n)} \leq \sqrt[n]{2(n+1)^6 S(n)}$. Applying the squeezing theorem finishes the proof. □

Fortunately, generating-function manipulations and dynamic-programming procedures produce quickly and efficiently elements of $C(n)$ and $S(n)$ for values of n as large as we want. (See sequences A067675 and A006958 in the on-line encyclopedia of integer sequences [107].)

3.6.1 Lower Bound

A standard concatenation argument shows that the sequence $S(n)$ is super-multiplicative.

Theorem 3.4. *$S(m)S(n) < S(m+n)$ for all natural numbers $m, n \geq 2$.* □

A weak relation is easily justified by a concatenation argument identical to that for regular polyominoes. However, the relation is strict since for any pair of numbers $m, n \geq 2$, there

exists a staircase polyomino P of size $m+n$, which cannot be broken lexicographically into two staircase polyominoes of sizes m and n, and whose concatenation yields the polyomino P.

Let us derive a lower bound on $\lambda_{2;S}$, which is clearly also a lower bound on $\lambda_{2;C}$.

Corollary 3.1. $\lambda_{2;S} \geq 2.3091$.

Proof. The following Mathematica program (see Sequence A006958 in the OEIS [107]) implements a continuous fraction (trimmed to five levels) which provides the generating function of $S(n)$ [55], derived from a q-expansion. In order to obtain exact terms of the expansion, one needs to set k, the level of trimming, to $\left\lceil \sqrt{n+1} \right\rceil$, where n is the number of computed terms. However, since we eventually take the nth root of the nth term, we can trim the fraction much earlier without any noticeable effect on the result with the number of decimal digits shown.

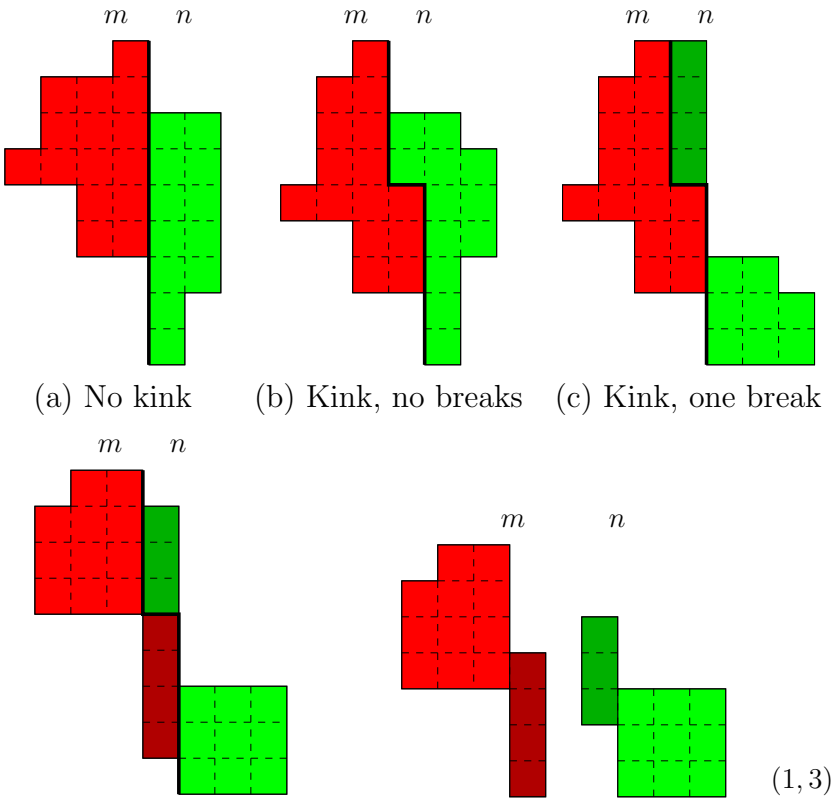
```
n=25000; k=5; Q=1; For[i=k, i>=1, i--, Q=1/(1-x^i/(1-x^i*Q))];
s=CoefficientList[Series[Q,{x,0,n+1}],x][[n+1]]   (* s = S(n) *)
Print[n,", ",ScientificForm[s+0.,10],SetPrecision[s^(1/(n-1)),7]]
```

Using this program, we easily see that $S(25000) \approx 5.289504001 \cdot 10^{9085}$. Using Method [O], we have the relation, $S^2(n) \leq S(2n-1)$ for any $n \in \mathbb{N}$. Hence, by Theorem 3.1(a), we have that $\lambda_{2;S} \geq \sqrt[24999]{S(25000)} \geq 2.30910$. □

3.6.2 Upper Bound

We now show that the sequence $C(n)$ is quasi sub-multiplicative.

Theorem 3.5. $C(m+n) \leq (mn + 2(m+n) + 1)C(m)C(n)$ *for all* $m, n \geq 1$.



(a) No kink (b) Kink, no breaks (c) Kink, one break

Original polyomino Broken polyomino and encoding

(d) Kink, two breaks

Figure 3.7: Breaking a convex polyomino into two polyominoes.

Proof. The claim follows from a concatenation argument. Refer to Figure 3.7. Consider a convex polyomino P of size $m+n$, and break it into two parts, the first one, colored in red, containing the smallest m cells of P according to lexicographic order, and the second one, colored in green, containing the other n cells. Due to the convexity of P, and since the line parting the two parts may have a "kink," both parts can be made of one or two connected components, each of which is convex by itself. (Indeed, since the split is according to lexicographic order, nonconvexity of any of the up to four components will immediately result in the nonconvexity of the original polyomino P.) In addition, if a part is broken into two components, then the component touching the kink from above or below (colored in dark red or dark green) is necessarily a vertical "stick." We use the components on each side of the parting line for constructing two convex polyominoes (of sizes m and n), and encode the combination by a positive integer number which is at most $mn + 2(m+n) + 1$, as we show below. We distinguish between three cases:

1. The parting line does not have a kink (see Figure 3.7(a)). In this case, the original polyomino is broken into two valid (connected) polyominoes. The relative position of the two parts can be encoded by a number in the range $[1, \ldots, m+n-1]$.

2. The parting line has a kink (see Figure 3.7(b)), but neither part (left or right) is broken into two components. In this case, there are cells of the original polyomino immediately below and above the kink, otherwise we were in the first case. Hence, there is only a single option to combine the two parts into the original polyomino, which we can encode by a single code.

3. One or two of the parts are broken into two components (see Figures 3.7(c) and (d,left). In this case, we slide vertically the two components towards each other until they touch horizontally for the first time (see Figure 3.7(d,right)). If needed, the vertical translation on the left (resp., right) side is by at most m (resp., n) cells. Together with the case of no break, a set of at most $m+1$ (resp., $n+1$) codes will encode all

possibilities. Note that we need both codes in order to restore uniquely the original polyomino from the two new polyominoes: First, place the two "sticks," and then the codes will tell us were to position the other component(s). Altogether, at most $(m+1)(n+1)$ different codes can encode all possibilities in this case.

In summary, a total of $(m+n-1)+1+(m+1)(n+1) = mn+2(m+n)+1$ codes suffice for encoding all possibilities, allowing us to reconstruct uniquely the original polyomino from the two polyominoes of sizes m, n and the code. Clearly, the relation is strict for large-enough values of m, n since this mapping is one-to-one but not surjective—not all combinations of polyominoes of sizes m, n and a code can be obtained. □

Corollary 3.2. $\lambda_{2;C} \leq 2.3297$.

Proof. Using a program (written by Ruben G. Spaans [107, Seq. A067675]) which implements a dynamic-programming algorithm, we computed values of $C(n)$ up to $n = 2000$ and found that $C(2000) \approx 2.319117764 \cdot 10^{727}$. The computation took about 85 hours on a home desktop with four Intel(R) Core(TM) i7-6700K CPU @4.00GHz processors (8 logical processors) with 32GB of RAM. By Theorem 3.5,

$$C(2n) \leq (n^2+4n+1)C^2(n) = (1+\frac{4}{n}+\frac{1}{n^2})n^2C^2(n) < 1.0020003 \cdot n^2C^2(n)$$

for all $n \geq 2000$. Hence, by Theorem 3.1(a),

$$\lambda_{2;C} \leq (1.002003(2 \cdot 2000)^2 C(2000))^{1/2000} \leq 2.32963.$$

□

3.6.3 Epilogue

As a result of Theorem 3.3, and Corollaries 3.1 and 3.2, we conclude the following.

Theorem 3.6. $2.3091 \leq \lambda_{2;S} = \lambda_{2;C} \leq 2.3297$. □

The bounds (the lower of which is closer to the true value) can be made as tight as we want since we can produce values of $S(n)$ and $C(n)$ for values of n as high as needed.

Klarner and Rivest [80] took a similar approach, in which they developed sequences of integral equations, from which they obtained generating functions for the enumeration sequence, which implied polynomials in $1/x$ whose largest roots served as a lower and upper bounds on λ_C. Our method is much simpler as it only requires the knowledge of $C(n)$ and $S(n)$ for large-enough values of n, which are, as noted, very easy to compute.

3.7 Conclusion

In this chapter, we explore various types of concatenation arguments, and show their applications to setting bounds on the growth constants of polyominoes and polycubes, as well as some specialized families of them. ***Inter alia***, we consider tree polycubes and convex polyominoes. The available counts of animals of different types support several unproven conjectures about monotonicity and relations between the root and ratio sequences of the counts. We do not use any of these conjectures in order to prove our bounds. However, we explain how they are consistent with the specific values that are used for setting the bounds.

A possible direction for future work is analyzing the quality of our lower bounds obtained by using the methods presented in this chapter. It was recently conjectured [21] that λ_d, the growth constant of d-dimensional polycubes, behaves like $(2d-3)e + O(1/d)$ (as $d \to \infty$), where e is the base of natural logarithms (see the blue line in the graph shown in Figure 3.8).

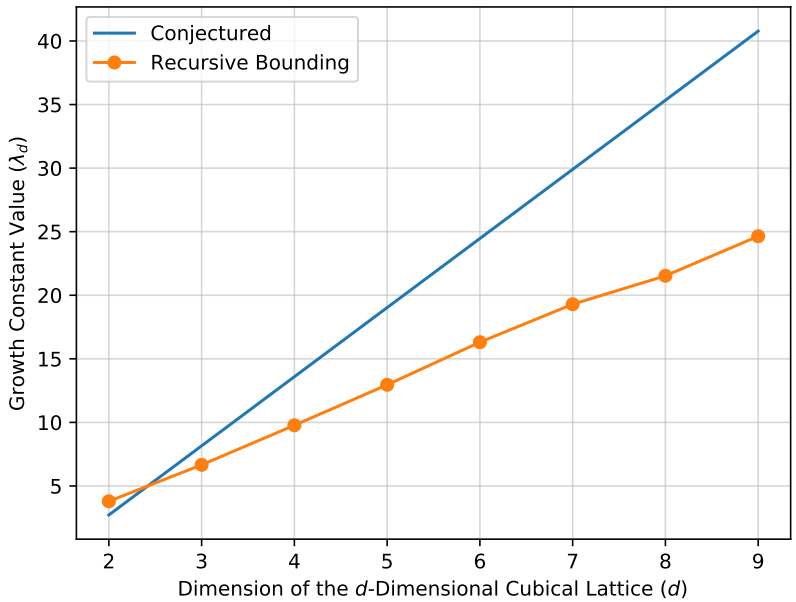


Figure 3.8: Conjectured growth constants of polycubes (blue), and lower bounds produced by our method (orange).

When we plot the lower bounds obtained by the recursive bounding method (the orange line in the same figure), we observe that except in dimensions 8 and 9, they also exhibit a linear dependence on d, which is surprisingly similar to $3.13d-2.63$ (obtained with Rvalue=0.9998 by Python's standard linear least-squares regression tool `scipy.stats.linregress`). We attribute the degradation in dimensions 8 and 9 to the lack of sufficient elements of the sequences enumerating polycubes in high dimensions. Are the approximate slope π and the approximate intercept $-e$ only a coincidence, or are they inherently related to the concatenation method?

Another direction for future research is applying the methods proposed in this chapter to other lattices, such as the planar triangular and hexagonal lattices. A preliminary work in this direction was done in a companion paper [20], where a simple version of Method [C] was used for improving the lower bound on the growth constant of ***polyiamonds*** (animals on the triangular lattice). The main challenge related to this lattice is that it contains two

distinct types of cells, which can attach to each other but not to themselves. We plan to investigate whether any method (especially Method [F] combined with recursive bounding) improves the lower bound further.

Chapter 4

Taming the Knight's Tour: Minimizing Turns and Crossing

4.1 Introduction

The game of chess is a fruitful source of mathematical puzzles. The puzzles often blend an appealing aesthetic with interesting and deep combinatorial properties [116]. An old and well-known problem is the knight's tour problem. A *knight's tour* in a generalized $n \times m$ board is a path through all nm cells such that any two consecutive cells are connected by a "knight move" (Figure 4.1). For a historic treatment of the problem, see [15].

A knight's tour is *closed* if the last cell in the path is one knight move away from the first one. Otherwise, it is *open*. This chapter focuses solely on closed tours, so henceforth we omit the distinction. The knight's tour problem is a special case of the Hamiltonian cycle problem, the problem of finding a simple cycle in a graph that visits all the nodes. Consider the graph with one node for each cell of the board and where nodes are connected if the

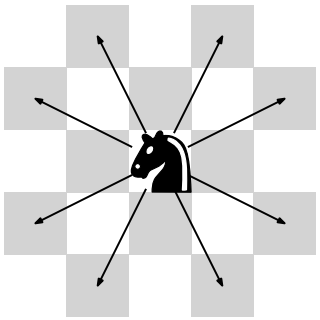


Figure 4.1: A knight moves one unit along one axis and two units along the other.

corresponding cells are a knight move apart. The knight's tour problem corresponds to finding a Hamiltonian cycle in this graph.

We approach the knight's tour problem in a novel way. Existing work focuses on the questions of existence, counting, and construction algorithms. In general, the goal of existing algorithms is to find *any* knight's tour. We propose two new metrics that capture simplicity and structure in a knight's tour, and set the goal of finding tours optimizing these metrics. We define the following optimization problems. We associate each cell in the board with a point (i, j) in the plane, where i is the row of the cell and j is the column.

Definition 4.1 (Turn)**.** *Given a knight's tour, a* ***turn*** *is* ***a triplet of consecutive cells with non-collinear coordinates.***

Remark 4.1 (Minimum turn knight's tour)**.** *Given a rectangular $n \times m$ board such that a knight's tour exists, find the knight's tour with the smallest number of turns.*

Definition 4.2 (Crossing)**.** *Given a knight's tour, a* ***crossing*** *occurs* ***when the two line segments corresponding to moves in the tour intersect****. That is, if $\{c_1, c_2\}$ and $\{c_3, c_4\}$ are two distinct pairs of consecutive cells visited along the tour, a crossing happens if the open line segments (c_1, c_2) and (c_3, c_4) intersect.*

Remark 4.2 (Minimum crossing knight's tour)**.** *Given a rectangular $n \times m$ board such that a knight's tour exists, find the knight's tour with the smallest number of crossings.*

Knight's tours are typically visualized by connecting consecutive cells by a line segment. Turns and crossings make the sequence harder to follow. Minimizing crossings is a central problem in ***graph drawing***, the sub-field of graph theory concerned with the intelligible visualization of graphs (e.g., see the survey in [62]). Problem 4.2 is the natural adaptation for knight's tours. Problem 4.1 asks for the (self-intersecting) polygon with the smallest number of vertices that represents a valid knight's tour.

4.1.1 Our contributions.

We propose a novel algorithm for finding knight's tours with the following features.

- $9.25n + O(1)$ turns and $12n + O(1)$ crossings on a $n \times n$ board.
- A $9.25/6 + o(1)$[1] approximation factor on the minimum number of turns (Problem 4.1).
- A $3 + o(1)$ approximation factor on the minimum number of crossings (Problem 4.2).
- A $O(nm)$ running time on a $n \times m$ board, i.e., linear on the number of cells, which is optimal.
- The algorithm is fully parallelizable, in that it can be executed in $O(1)$ time with $O(nm)$ processors in the CREW PRAM model. More specifically, the cell at a given index in the tour sequence (or, conversely, the index of a given cell) can be determined in constant time, which implies the above.
- It can be generalized to most typical variations of the problem: high-dimensional boards, boards symmetric under 90 degree rotations, tours in boards with odd width and height that skip a corner cell, and tours for $(1, 4)$-leapers, called *giraffes*, which move one cell in one dimension and four in the other.

[1] By $o(1)$, we mean a function $f(n)$ such that for any constant $\varepsilon > 0$, there is an n_0 such that, for all $n \geq n_0$, $f(n) < \varepsilon$.

- The algorithm can be simulated by hand with ease. This is of particular interest in the context of recreational mathematics and mathematics outreach.

The chapter is organized as follows. In Section 4.1.2, we give an overview of the literature on the knight's tour problem and its variants. We describe the algorithm in Section 4.2. We prove the approximation ratios in Section 4.3. Section 4.4 deals with the mentioned extensions. We conclude in Section 4.5.

The tours produced by the algorithm can be generated interactively for different board dimensions at `https://www.ics.uci.edu/~nmamano/knightstour.html`.

4.1.2 Related Work

Despite being over a thousand years old [116], the knight's tour problem is still an active area of research. We review the key questions considered in the literature.

Existence. In rectangular boards, a tour exists as long as one dimension is even and the board size is large enough; no knight's tour exists for dimensions $1 \times n$, $2 \times n$ or $4 \times n$, for any $n \geq 1$ and, additionally, none exist for dimensions 3×6 or 3×8 [104]. In three dimensions or higher, the situation is similar: a tour exists only if at least one dimension is even and large enough [41, 42, 49]. In the case of open knight's tours, a tour exists in two dimensions if both dimensions are at least 5 [38, 36].

Counting. The number of closed knight's tours in an even-sized $n \times n$ board is at least $\Omega(1.35^{n^2})$ and at most 4^{n^2} [83]. The exact number of knight's tours in the standard 8×8 board is $26,534,728,821,064$ [88]. Furthermore algorithms for enumerating multiple [106] and enumerating all [6] knight's tours have also been studied.

Algorithms. Historically, greedy algorithms have been popular. The idea is to construct the tour in order, one step at a time, according to some heuristic selection rule. Warnsdorff's rule and its refinements [98, 6, 108] work well in practice for small boards, but do not scale to larger boards [93]. The basic idea is to choose the next node with fewest continuations. Interestingly, this heuristic can be effective in the more general Hamiltonian cycle problem [98].

To our knowledge, every efficient algorithm for arbitrary board sizes before this chapter is based on a divide-and-conquer approach. The tour is solved for a finite set of small, constant-size boards. Then, the board is covered by these smaller tours like a mosaic. The small tours are connected into a single one by swapping a few knight moves. This can be done in a bottom-up [104, 36, 41, 42, 72] or a top-down recursive [94, 84] fashion. This process is simple and can be done in time linear on the number of cells. Like our algorithm, these algorithms are highly parallelizable [36, 94]. This is because the tours are made of predictive repeating patterns.

Divide-and-conquer is not suitable for finding tours with a small number of turns or crossings. Since each base solution has constant size, a $n \times n$ board is covered by $\Theta(n^2)$ of them, and each one contains turns and crossings. Thus, the divide-and-conquer approach necessarily results in $\Theta(n^2)$ turns and crossings. In contrast, our algorithm has $O(n)$.

Extensions. The above questions have been considered in related settings. Extensions can be classified into three categories, which may overlap:

- **Tours with special properties.** Our work can be seen as searching for tours with special properties. *Magic knight's tours* are also in this category: tours such that the indices of each cell in the tour form a magic square (see [22] for a survey).

The study of symmetry in knight's tours dates back at least to 1917 [24]. Symmetric tours under 90 degree rotations exist in $n \times n$ tours where $n \geq 6$ and n is of the form $4k + 2$ for some k [40]. Parberry extended the divide-and-conquer approach to produce tours symmetric under 90 degree rotations [94]. Jelliss provided results on which rectangular board sizes can have which kinds of symmetry [66].

Both of our proposed problems are new, but minimizing crossings is related to the *uncrossed knight's tour problem*, which asks to find the longest sequence of knight moves without *any* crossings [119]. This strict constraint results in incomplete tours. This problem has been further studied in two [65, 53] and three [82] dimensions.

- **Board variations.** Besides higher dimensions, knight's tours have been considered in other boards, such as torus boards, where the top and bottom edges are connected, and the left and right edges are also connected. Any rectangular torus board has a closed tour [117]. Another option is to consider boards with odd width and height. Since boards with an odd number of cells do not have tours, it is common to search for tours that skip a specific cell, such as a corner cell [94].

- **Move variations.** An (i, j)-*leaper* is a generalized knight that moves i cells in one dimension and j in the other (the knight is a $(1, 2)$-leaper) [91]. Knuth studied the existence of tours for general (i, j)-leapers in rectangular boards [81]. Tours for *giraffes* ($(4, 1)$-leapers) were provided in [40] using a divide-and-conquer approach. Chia and Ong [33] study which board sizes admit generalized (a, b)-leaper tours. Kamčev [72] showed that any board with sufficiently large and even size admits a $(2, 3)$-, $(2, 5)$-, and a $(a, 1)$-leaper tour for any even a, and generalized this to any higher dimensions. Note that a and b are required to be coprime and not both odd, or no tour can exist [72].

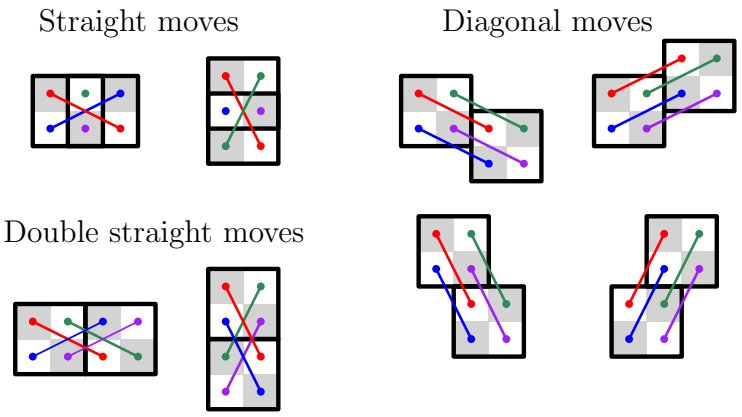


Figure 4.2: Quartet of knights moving in unison without leaving any unvisited squares. Note that, in a straight move, the starting and ending position of the quartet overlap because two of the knights remain in place.

4.2 The Algorithm

Given that one of the dimensions must be even for a tour to exist, we assume, without loss of generality, that the width w of the board is even, while the height h can be odd. We also assume that $w \geq 16$ and $h \geq 12$. The construction still works for some smaller sizes, but may require tweaks to its most general form described here.

Quartet moves. What makes the knight's tour problem challenging is that knight jumps leave "gaps". Our first crucial observation is that a quartet of four knights arranged in a square 2×2 formation can move "like a king": they can move horizontally, vertically, or diagonally without leaving any gaps (Figure 4.2).

By using the "formation moves" depicted in Figure 4.2, four knights can easily cover the board moving vertically and horizontally while remaining in formation. Of course, the goal is to traverse the entire board in a single cycle, not four paths. We address this issue with special structures placed in the bottom-left and top-right corners of the board, which we call *junctions*, and which tie the paths together to create a single cycle. Note that using only straight formation moves leads to tours with a large number of turns and crossings.

Fortunately, two consecutive diagonal moves in the same direction introduces no turns or crossings, so our main idea is to use as many diagonal moves as possible. This led us to the general pattern shown in Figure 4.3.

The full algorithm is given in Algorithm 3. The formation starts at a junction at the bottom-left corner and ends at a junction at the top-right corner. To get from one to the other, it zigzags along an odd number of parallel diagonals, alternating between downward-right and upward-left directions. The junctions in Figure 4.4 have a *height*, which influences the number of diagonals traversed by the formation. At the bottom-left corner, we use a junction with height 5. At the top-right corner, we use a junction with height between 5 and 8. Choosing the height as in Algorithm 3 guarantees that, for any board dimensions, an odd number of diagonals fit between the two junctions. Sequence 1 in Figure 4.5, which we call the *heel*, is used to transition between diagonals along the horizontal edges of the board. The two non-junction corners may require special sequences of quartet moves, as depicted in Figure 4.5. In particular, Sequences $1, 2, 3$, and 0 are used when the last heel ends $0, 2, 4$, and 6 columns away from the vertical edge, respectively. As with the height of the top-right junction, these variations are *predictable* because they cycle as the board dimensions grow[2], so in Algorithm 3 we give expressions for them in terms of w and h.

4.2.1 Correctness

It is clear that the construction visits every cell, and that every node in the underlying graph of knight moves has degree two. However, it remains to be argued that the construction is actually a single closed cycle. For this, we need to consider the choice of junctions.

A junction is a pair of disjoint knight paths whose four endpoints are adjacent as in the quartet formation. Thus, the bottom-left junction connects the knights into two pairs.

[2]See how the variations cycle at `https://www.ics.uci.edu/~nmamano/knightstour.html`.

Algorithm 3: Knight's tour algorithm for even width $w \geq 16$ and height $h \geq 12$.

1. Fill the corners of the board as follows:

   Bottom-left: first junction in Figure 4.4.

   Top-right: junction of height $5 + ((w/2 + h - 1) \bmod 4)$ in Figure 4.4 except the first one.

   Bottom-right: Sequence $(w/2 + 2) \bmod 4$ in Figure 4.5.

   Top-left: Sequence $(3 - h) \bmod 4$ in Figure 4.5 rotated 180 degrees.

2. Connect the four corners using formation moves, by moving along diagonals from the bottom-left corner to the top-right corner as in Figure 4.3. To transition between diagonals:

   Vertical edges: use a double straight up move (Figure 4.2).

   Horizontal edges: use Sequence 1 in Figure 4.5.

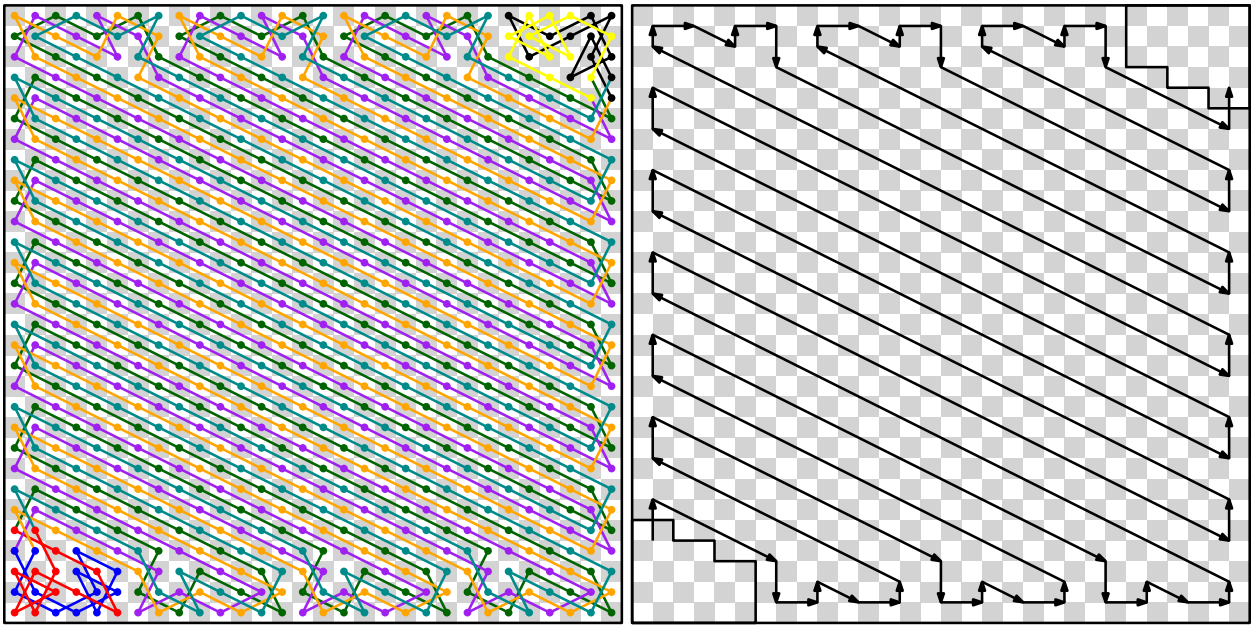


Figure 4.3: Side by side comparison between the knight's tour and the underlying quartet moves in a 30×30 board. The arrows illustrate sequences of consecutive and equal formation moves. Starting from the bottom-left square of the board, the single knight's tour follows the colored sections of the tour in the following order: red, green, yellow, purple, blue, orange, black, cyan, and back to red.

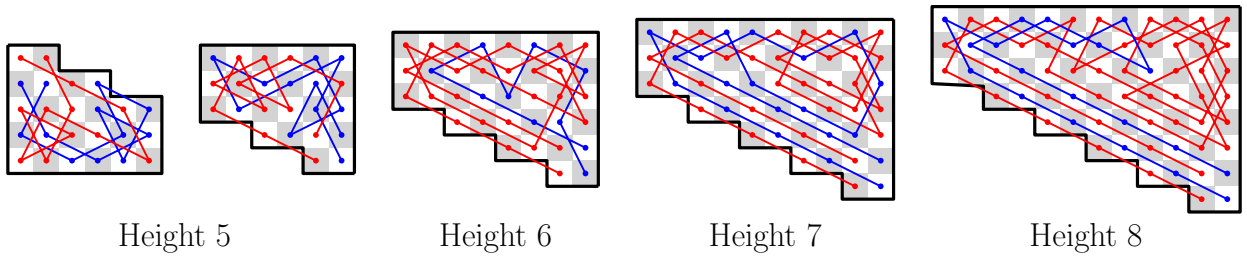


Figure 4.4: Junctions used in our construction.

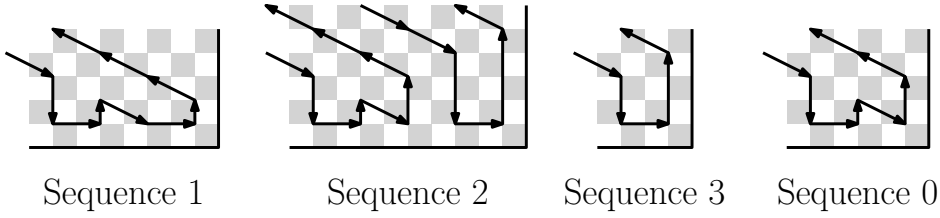


Figure 4.5: The four possible cases for the bottom-right corner.

Denote the four knight positions in the formation by tl, tr, bl, br, where the first letter indicates top/bottom and the second left/right. We consider the three possible *positional matchings* with respect to these positions: horizontal matching $H = (tl, tr), (bl, br)$, vertical matching $V = (tl, bl), (tr, br)$, and cross matching $X = (tl, br), (tr, bl)$. Let $\mathcal{M} = \{H, V, X\}$ denote the set of positional matchings. We are interested in the effect of formation moves on the positional matching. A formation move does not change which knights are matched with which, but a non-diagonal move changes their positions, and thus their labels tl, tr, bl, br also change. For instance, a horizontal matching becomes a cross matching after a straight move to the right.

It is easy to see that a straight move upwards or downwards has the same effect on the positional matching. Similarly for left and right straight moves. Thus, we classify the formation moves in Figure 4.2 (excluding double straight moves, which are a composition of two straight moves) into vertical straight moves $\updownarrow$, horizontal straight moves $\leftrightarrow$, and diagonal moves $\swarrow\!\!\!\nearrow$. Let $\mathcal{S} = \{\updownarrow, \leftrightarrow, \swarrow\!\!\!\nearrow\}$ denote the three types of quartet moves. We see each move type $s \in \mathcal{S}$ as a function $s : \mathcal{M} \to \mathcal{M}$ (see Table 4.1). Note that the diagonal move $\swarrow\!\!\!\nearrow$

	↙↗	↕	↔	↕↔	↔↕	↕↔↕
V	V	X	V	H	X	H
H	H	H	X	X	V	V
X	X	V	H	V	H	X

Table 4.1: Result of applying each type of formation move, as well as three compositions of sequences of moves, to each formation matching.

	↙↗	↕	↔	↕↔	↔↕	↕↔↕
↙↗	↙↗	↕	↔	↕↔	↔↕	↕↔↕
↕	↕	↙↗	↕↔	↔	↕↔↕	↔↕
↔	↔	↔↕	↙↗	↕↔↕	↕	↕↔
↕↔	↕↔	↕↔↕	↕	↔↕	↙↗	↔
↔↕	↔↕	↔	↕↔↕	↙↗	↕↔	↕
↕↔↕	↕↔↕	↕↔	↔↕	↕	↔	↙↗

Table 4.2: Cayley table for the group of positional matching permutations.

is just the identity. Given a sequence of moves $S = (s_1, \ldots, s_k)$, where each $s_i \in \mathcal{S}$, let $S(M) = s_1 \circ \cdots \circ s_k(M)$.

The move types $\updownarrow, \leftrightarrow, \swarrow\!\!\nearrow$ seen as functions are, in fact, permutations (Table 4.1). It follows that *any* sequence of formation moves permutes the positional matchings, according to the composed permutation of each move in the sequence. There are six possible permutations of the three positional matchings, three of which correspond to the "atomic" formation moves $\swarrow\!\!\nearrow, \updownarrow$, and $\leftrightarrow$. The other three permutations can be obtained by composing atomic moves, for instance, with the compositions $\updownarrow\leftrightarrow, \leftrightarrow\updownarrow$, and $\updownarrow\leftrightarrow\updownarrow$ (Table 4.1). Thus, any sequence of moves permutes the positional matchings in the same way as one of the sequences in the set $\{\swarrow\!\!\nearrow, \updownarrow, \leftrightarrow, \updownarrow\leftrightarrow, \leftrightarrow\updownarrow, \updownarrow\leftrightarrow\updownarrow\}$. This is equivalent to saying that this set, under the composition operation, is isomorphic to the symmetric group of degree three. Table 4.2 shows the Cayley table of this group.

Let $T_{w,h}$ be the sequence of formation moves that goes from the bottom-left junction to the top-right one in Algorithm 3 in a $w \times h$ board.

Lemma 4.1. *For any even $w \geq 16$ and any $h \geq 12$, $T_{w,h}(H) = H$.*

Proof. We show that the entire sequence of moves $T_{w,h}$ is either neutral or equivalent to single vertical move, depending on the board dimensions. According to Table 4.1, this suffices to prove the lemma.

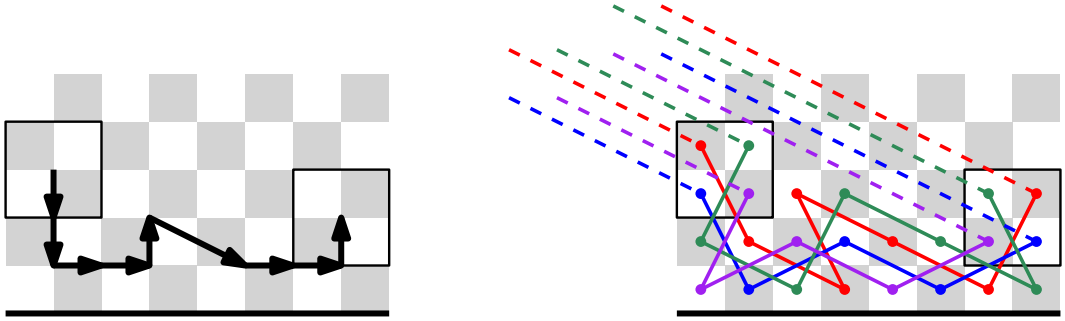


Figure 4.6: Visualization of how the heel permutes the position of the knights. Note that the sequence of moves flips the columns of the knights (the knight in position *tl* moves to *tr* and so on). However, this does not affect their positional matching. For instance, if the knights were paired in a horizontal matching, after flipping the columns, they are still in a horizontal matching. The same holds for vertical and cross matchings.

The sequence $T_{w,h}$ consists mostly of diagonal moves, which are neutral. The transition between diagonals along the vertical edges consist of two vertical moves, which are also neutral ($\updownarrow\updownarrow = \nearrow$). The heel is also neutral, as it consists of the sequence $\updownarrow\updownarrow\leftrightarrow\leftrightarrow\updownarrow\leftrightarrow\leftrightarrow\updownarrow$ (omitting diagonal moves) which is again equivalent to $\nearrow$. This is easy to see by noting that any two consecutive vertical or horizontal moves cancel out. It is depicted in detail in Figure 4.6. Thus, $T_{w,h}$ reduces to composing the sequences in the bottom-right and top-left corners. As mentioned, Sequence 1 (the heel) is neutral. It is easy to see that the other sequences (counting each part of Sequence 2 separately) is equivalent to $\updownarrow$. Thus, we get that $T_{w,h}$ is simply the composition of zero to four vertical moves, depending on the width and height of the board. This further simplifies to zero or one vertical moves. □

Theorem 4.1 (Correctness)**.** *Algorithm 3 outputs a valid knight's tour in any board with even width* $w \geq 16$ *and with height* $h \geq 12$*.*

Proof. Clearly, the construction is a set of disjoint cycles in the underlying knight-move graph. We prove that it is actually one cycle. Given a set of disjoint cycles in a graph, *contracting* a node in one of the cycles is the process of removing it and connecting its two neighbors in the cycle. Clearly, contracting a node in a cycle of length ≥ 3 does not change

the number of cycles. Thus, consider the remaining graph if we contract all the nodes except the four endpoints of the top-right junction.

Note that we use a horizontal matching in the bottom-left junction and a vertical matching in the top-right junction. Contracting the non-endpoint nodes inside the top-right junction leaves the two edges corresponding to the vertical matching. By Lemma 4.1, contracting the nodes outside the top-right junction leaves the edges corresponding to a horizontal matching. Thus, the resulting graph is a single cycle of four nodes. □

The choice of matchings at the junctions is important; using a horizontal matching in the top-right junction would not result in a knight's tour.

4.3 Lower Bounds and Approximation Ratios

In this section, we analyze the approximation ratio that our algorithm achieves for Problem 4.1 and Problem 4.2. For simplicity, we restrict the analysis to square boards. First, we briefly discuss the classification of these problems in complexity classes.

4.3.1 Computational Complexity

Consider the following decision versions of the problems: is there a knight's tour on an $n \times n$ board with at most k turns (resp. crossings)? We do not know if these problems are in P. Furthermore, it may depend on how the input is encoded. Technically, the input consists of two numbers, n and k, which can be encoded in $O(\log n + \log k)$ bits. However, it is more natural to do the analysis as a function of the size of the board (or, equivalently, of the underlying graph on which we are solving the Hamiltonian Cycle problem), that is, $\Theta(n^2)$. It is plausible that the optimal number of turns (resp. crossings) is a simple, arithmetic

function of n. This would be the case if the optimal tour follows a predictable pattern like our construction (note that we can count the number of turns or crossings of our algorithm without constructing it). If that is the case, then the problems are in P, regardless of how the input is encoded.

If the input is represented using $\Theta(n^2)$ space, the problems are clearly in NP, as a tour with k turns/crossings acts as a certificate of polynomial length. However, unless $\mathsf{P} = \mathsf{NP}$, the problems are not NP-hard. To see this, consider the language

$$\{1^n 01^k \mid \text{ there is a tour with at most } k \text{ turns in an } n \times n \text{ board}\},$$

and analogously for crossings. These languages are sparse, meaning that, for any given word length, there is a polynomial number of words of that length in the language. Mahaney's theorem states that if a sparse language is NP-complete, then $\mathsf{P} = \mathsf{NP}$ [86]. This suggests that the problems are in P, though technically they could also be NP-intermediate.

If the input is represented using $O(\log n + \log k)$ bits, then the problems are in NEXP because the "unary" versions above are in NP. Note that, in this setting, simply listing a tour would require time exponential on the input size.

4.3.2 Number of Turns

Upper bound. All the turns in our construction happen near the edges. The four corners account for a constant number of turns. The left and right edges have eight turns for each four rows. As it can be seen in Figure 4.6, the heel has 22 turns, so the top and bottom edges have 22 turns each for each eight columns. Therefore, the number of turns in our construction is bounded by $2\frac{8}{4}n + 2\frac{22}{8}n + O(1) = 9.5n + O(1)$.

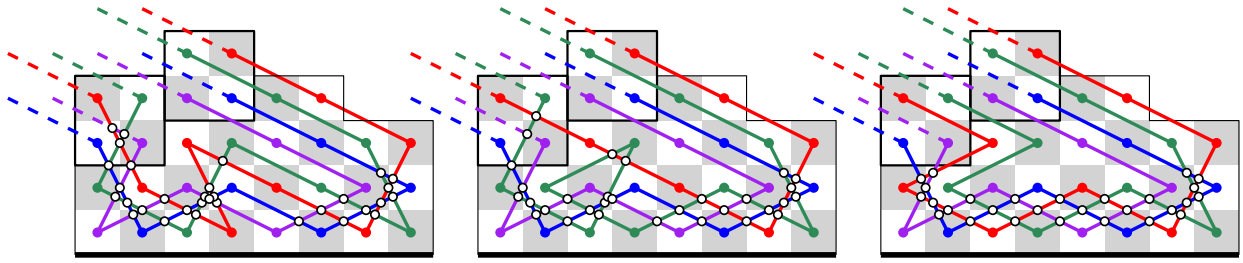


Figure 4.7: **Left:** the heel resulting from formation moves. It has 22 turns and 32 crossings. The crossings are marked with white disks. **Center:** the optimal configuration for minimizing turns. It has 21 turns and 31 crossings. **Right:** the optimal configuration for minimizing crossings. It has 22 turns and 28 crossings.

We can reduce the number of turns slightly by replacing each heel with an alternative combination of moves that covers the same cells and starts and ends with the four knights in the same place. In other words, the four knights "break formation" temporarily and reassemble before leaving the heel in the next diagonal.

We can do an exhaustive search for four knights that cover the squares shown in Figure 4.7, starting and ending in the two groups of four cells on the top-left. Of all the possible solutions, Figure 4.7 shows a solution minimizing the number of turns and one minimizing the number of crossings, including turns and crossings caused by the path continuations outside the heel. There is no single solution minimizing both turns and crossings. The four knights are not required to end in any specific configuration, but the optimal solutions happened to end in the same configuration as the original heel.

Lower bound. We now give a lower bound on the number of turns in the optimal tour. First, note that every cell next to an edge *must* contain a turn. This accounts for $4n - 4$ turns. Here we focus on the main result, a lower bound of $(6 - \varepsilon)n$ for any $\varepsilon > 0$. We start with some intermediate results.

We associate each cell in the board with a point (i, j) in the plane, where i is the row of the cell and j is the column. An edge cell only has four moves available. We call the directions

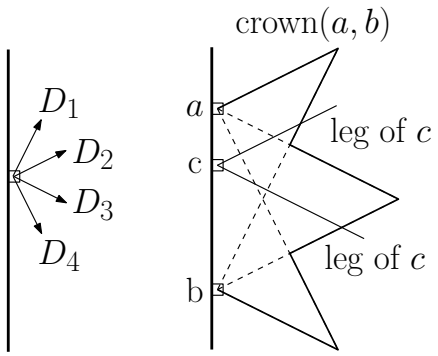


Figure 4.8: Illustration of the terminology for the lower bound. Note that c is a clean cell (with respect to the crown of a and b) because both of its legs escape it.

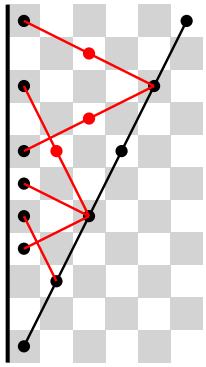


Figure 4.9: The black leg collides would collide with all the red legs.

of these moves D_1, D_2, D_3, and D_4, in clockwise order. For an edge cell c, let $r_i(c)$, with $1 \leq i \leq 4$, denote the ray starting at c and in direction D_i. That is, the ray that passes through the cells reachable from c by moving along D_i.

Let a and b be two cells along the left edge of the board, with a above b. The discussion is symmetric for the other three edges. Given two intersecting rays r and r', one starting from a and one from b, let $S(r, r')$ denote the set of cells in the region of the board *bounded* by r and r': the set of cells below or on r and above or on r'. We define the *crown* of a and b as the following set of cells (see Figure 4.8):

$$\text{crown}(a, b) = S(r_2(a), r_1(b)) \cup S(r_3(a), r_2(b)) \cup S(r_4(a), r_3(b)).$$

We can associate, with each edge cell c, the two maximal sequences of moves without turns in the tour that have c as an endpoint. We call them the *legs* of c. We say that legs begin at c and end at their other endpoint. We say two legs of different cells *collide* if they end at the same cell. Let $C_{a,b}$ denote the set of edge cells along the right edge between a and b (a and b included). The following is easy to see.

Remark 4.1. *Any collision between the legs of edge cells in $C_{a,b}$ happens inside crown(a, b).*

We say that a leg of a cell in $C_{a,b}$ *escapes* the crown of a and b if it ends outside the crown. We say an edge cell in $C_{a,b}$ is *clean*, with respect to $C_{a,b}$, if both of its legs escape. We use the following observation.

Remark 4.2. *Let $m = |C_{a,b}|$ and k be the number of clean cells in $C_{a,b}$. The number of turns inside crown(a, b) is at least $m + (m - k)/2$.*

Proof. Each edge cell is one turn. Further, each of the $m - k$ non-clean cells have a leg that ends in a turn inside the crown. This turn may be because it collided with the leg of another edge cell in the crown. Thus, there is at least one turn for each two non-clean edge cells. □

To obtain the lower bound, we show that there is only a constant number of clean cells inside a crown.

Lemma 4.2. *Let a, b be two cells along the left edge of the board, with a above b. There are at most 122 clean cells inside crown(a, b).*

Proof. First we show that there are at most 60 clean cells such that one of their legs goes in direction D_1. For the sake of contradiction, assume that there are at least 61. Then, there are two, c and d, such that c is $60r$ rows above d, for some $r \in \mathbb{N}, r \geq 1$. The contradiction follows from the fact that the other leg of c, which goes along D_2, D_3, or D_4, would collide with the leg of b along D_1. This is because, for any $l \geq 1$, the leg of b along D_1 collides with (see Figure 4.9):

- any leg along D_2 starting from a cell $3l$ rows above b,
- any leg along D_3 starting from a cell $5l$ rows above b, and
- any leg along D_4 starting from a cell $4l$ rows above b.

Since $60r$ is a multiple of $3, 4$, and 5, no matter what direction the other leg of c goes, it collides with the leg of d. As observed, this collision happens inside the crown. Thus, c and d are not clean.

By a symmetric argument, there are at most 60 clean cells such that one of their legs goes in direction D_4.

Finally, note that there can only be two clean cells with legs in directions D_2 *and* D_3. This is because, by a similar argument, there cannot be two such cells at an even distance of each other; the leg along D_3 of the top one would collide against the leg along D_2 of the bottom one. □

Corollary 4.1. *Suppose that the crown of a and b has $m \geq 122$ edge cells. Then, there are at least $(m - 122)/2$ turns inside the crown at non-edge cells.*

Now, consider the iterative process depicted in Figure 4.10, defined over the unit square. The square is divided in four sectors along its main diagonals. Whereas earlier we used the term 'crown' to denote a set of cells, here we use it to denote the polygon with the *shape* of a crown. On the first step, a maximum-size crown is placed on each sector. At step $i > 1$, we place 2^{i-1} more crowns in each sector. They are maximum-size crowns, subject to being disjoint from previous crowns, in each gap between previous crowns and between the crowns closest to the corners and the main diagonals.

Lemma 4.3. *For any $1 > \varepsilon > 0$, there exists an $i \in \mathbb{N}$ such that at least $(1 - \varepsilon)$ of the boundary of the unit square is inside a crown after i iterations of the process.*

Proof. At each iteration, a constant fraction larger than 0.36 of the length on each side that is not in a crown is added to a new crown (Figure 4.11). This gives rise to a series A_i for the fraction of the side inside crowns after i iterations: $A_1 = 1/3$, $A_{i+1} > A_i + 0.36(1 - A_i)$ for $i > 1$; this series converges to 1. □

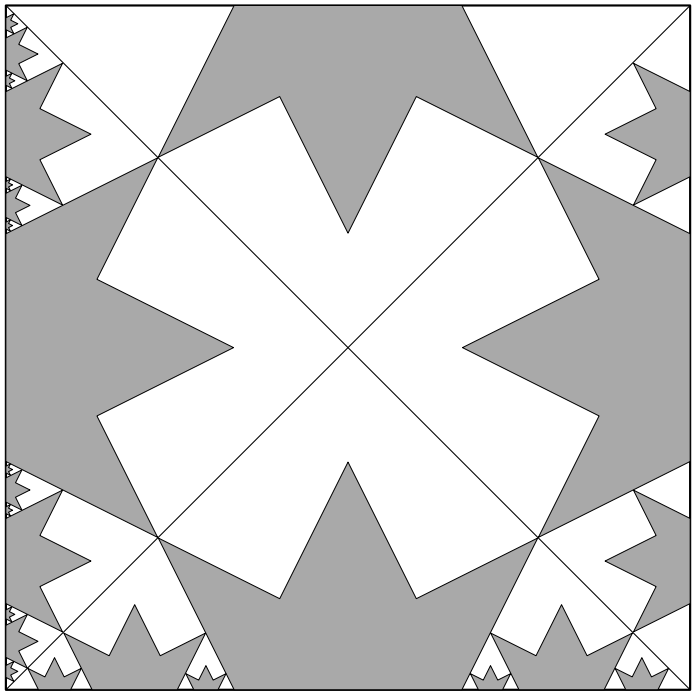


Figure 4.10: Each sector of the square shows the process after a different number of iterations: $1, 2, 3$, and 4 iterations on the top, right, bottom, and left sectors, respectively.

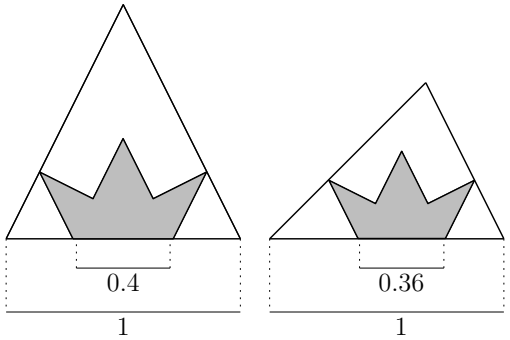


Figure 4.11: Lower bounds on two ratios. **Left:** the ratio between the gap between consecutive crowns and the base of the maximum-size crown that fits in the gap is > 0.4. **Right:** the ratio between the gap between a crown and a main diagonal and the base of the maximum-size crown that fits in the gap is > 0.36.

Theorem 4.2 (Lower bound)**.** *For any constant $\varepsilon > 0$, there is a sufficiently large n such that any knight's tour on a $n \times n$ board requires $(6 - \varepsilon)n$ turns.*

Proof. We show a seemingly weaker form of the claim: that there is a sufficiently large n such that any knight's tour on a $n \times n$ board requires $(6 - 2\varepsilon)n - C_\varepsilon$ turns, where C_ε is a constant that depends on ε but not on n. This weaker form is in fact equivalent because, for sufficiently large n, $C_\varepsilon < \varepsilon n$, and hence $(6 - 2\varepsilon)n - C_\varepsilon > (6 - 3\varepsilon)n$. Thus, the claim is equivalent up to a multiplicative factor in ε, but note that it is a claim about arbitrarily small ε, so it is not affected by a multiplicative factor loss.

Let i be the smallest number of iterations of the iterative process in Figure 4.10 such that at least $(1 - \varepsilon)$ of the boundary of the unit square is inside crown shapes. The number i exists by Lemma 4.3. Fix S to be the corresponding set of crown shapes, and $r = |S|$. Note that $r = 4(2^i - 1)$ is a constant that depends only on ε. Now, consider a square $n \times n$ board with the crown shapes in S overlaid in top of them. Let the board size n be such that the smallest crown in S contains more than 122 edge cells. Then, by Corollary 4.1, adding up the turns at non-edge cells over all the crowns in S, we get at least $4n(1 - \varepsilon)/2 - 61r$ turns. Adding the $4n - 4$ turns at edge cells, we get that the total number of turns is at least $(6 - 2\varepsilon)n - 61r - 4$. To complete the proof, consider $C_\varepsilon = 61r + 4$. □

Corollary 4.2. *Algorithm 3 achieves a $19/12 + o(1)$ approximation on the minimum number of turns.*

Proof. In a $n \times n$ board, let $ALG(n)$ denote the number of turns in the tour produced by Algorithm 3 and $OPT(n)$ denote the minimum number of turns. Let $\varepsilon > 0$ be an arbitrarily small constant. We show that there is an n_0 such that for all $n \geq n_0$, $ALG(n)/OPT(n) < 19/12 + \varepsilon$.

As mentioned, for any even $n \geq 16$, $ALG(n) < 9.5n + c$ for some small constant c. In addition, by Theorem 4.2, for sufficiently large n, $OPT(n) > (6 - \varepsilon)n$. Thus,

$$\frac{ALG(n)}{OPT(n)} < \frac{9.5n + c}{(6 - \varepsilon)n}$$

Furthermore, for large enough n, $c/((6 - \varepsilon)n) < \varepsilon/2$, so

$$\frac{ALG(n)}{OPT(n)} < \frac{9.5}{6 - \varepsilon} + \frac{\varepsilon}{2} = \frac{19}{12 - 2\varepsilon} + \frac{\varepsilon}{2} < \frac{19 + 6\varepsilon}{12} + \frac{\varepsilon}{2} = \frac{19}{12} + \varepsilon.$$

□

4.3.3 Number of Crossings

Similarly to the case of turns, all the crossings in our construction happen near the edges. The four corners account for a constant number of crossings. The left and right edges have 10 crossings for each four rows. The top and bottom edges have 32 crossings for each eight columns (Figure 4.6). Therefore, the number of turns in our construction is bounded by $2\frac{10}{4}n + 2\frac{32}{8}n + O(1) = 13n + O(1)$.

Lower bound. We prove the following lower bound on the number of crossings.

Lemma 4.4. *Any knight's tour on an $n \times n$ board has at least $4n - O(1)$ crossings.*

Proof. Let T be an arbitrary knight's tour on an $n \times n$ board. We show that T has $n - O(1)$ crossings involving knight moves incident to the cells along the left edge of the board. An analogous argument holds for the three other edges of the board, which combined yield the desired bound.

We partition the edge cells along the left-most column into sets of three consecutive cells, which we call *triplets* (if n is not multiple of three, we ignore any remaining cells, as they only account for a constant number of crossings). Two triplets are *adjacent* if they contain adjacent cells. Each triplet has six associated knight moves in the tour T, two for each of its cells. We call the choice of moves the *configuration* of the triplet. Since there are $\binom{4}{2} = 6$ choices of moves for each cell, there are $6^3 = 216$ possible configurations of each triplet.

Consider a weighted directed graph G with a node for each of the 216 possible triplet configuration and an edge from every node to every node, including a loop from each node to itself. The graph has weights on both vertices and edges. Given a node v, let $C(v)$ denote its associated configuration. The weight of v is the number of crossings between moves in $C(v)$. The weight of each edge $v \to u$ is the number of crossings between moves in $C(v)$ and moves in $C(u)$ when $C(v)$ is adjacent and above $C(u)$.

Each path in G represents a choice of move configurations for a sequence of consecutive triplets. Note that if two knight moves in T with endpoints in edge cells cross, the edges cells containing the endpoints are either in the same triplet or in adjacent triplets. Thus, the sum of the weights of the vertices and edges in the path equals the total number of crossings among all of these moves. Since G is finite, any sufficiently long path eventually repeats a vertex. Given a cycle c, let $w(c)$ be the sum of the weights of nodes and edges in c, divided by the length of c. Let c^* be the cycle in G minimizing w. Then, $w(c^*)$ is a lower bound on the number of crossings per triplet along to edge.

By examining G, we can see that $w(c^*) = 3$. Figure 4.12 shows an optimum cycle, which in fact uses only one triplet configuration. The cycle minimizing w can be found using Karp's algorithm for finding the minimum mean weight cycle in directed graphs [75], which runs in $O(|V| \cdot |E|)$ time in a graph with vertex set V and edge set E. However, this requires modifying the graph G, as Karp's algorithm is not suitable for graphs that also have node weights. We transform G into a directed graph G' with weights on only the edges and which

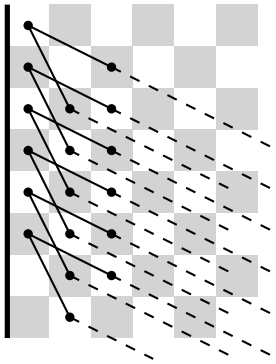


Figure 4.12: A configuration pattern that produces the minimum number of crossings along the edge of the board. The moves in the triplet configurations are shown in black. The dashed continuations illustrate that the moves in the configuration pattern can be extended to any number of columns without extra crossings.

preserves the optimal solution, as follows. We double each node v in G into two nodes v_{in}, v_{out} in G'. We also add an edge $v_{in} \to v_{out}$ in G' with weight equal to the weight of v in G. For each edge $v \to u$ in G, we add an edge $v_{out} \to u_{in}$ in G'. □

Since we only counted crossings between moves incident to the first column, a question arises of whether the lower bound can be improved by considering configurations spanning more columns (e.g., the two or three leftmost columns). The answer is negative for any constant number of columns. Figure 4.12 shows that the edges can be extended to paths that cover any fixed number of rows away from the edge without increasing the number of crossings.

Corollary 4.3. *Algorithm 3 achieves a* $13/4 + o(1)$ *approximation on the minimum number of crossings.*

4.4 Extensions

The idea of using formation moves to cover the board and junctions to close the tour is quite robust to variations of the problem. We show how this can be done in some of the most popular generalizations of the problem.

A variant that we do not consider is torus boards (where opposite edges are connected). The problem of finding tours with a small number of turns seems easier on a torus board, because one is not forced to make a turn when reaching an edge. Nonetheless, in a square $n \times n$ torus board, $\Omega(n)$ turns are still required, because making n consecutive moves in the same direction brings the knight back to the starting position, so at least one turn is required for each n visited cells. The tours for torus boards in [117] match this lower bound up to constant factors, at least for some board dimensions (see the last section in [117]). Crossings are not straightforward to define in torus boards.

4.4.1 High-dimensional boards

We extend our technique to three and higher dimensions. In d dimensions, a knight moves by 1 and 2 cells along any two dimensions, and leaves the remaining coordinates unchanged. A typical technique to extend a knight's tour algorithm to three dimensions is the "layer-by-layer" approach [116, 41, 42]: a 2D tour is reused on each level of the 3D board, adding the minimal required modifications to connect them into a single tour. We also follow this approach. (Watkins and Yulan [101] consider a generalizations of knight moves where the third coordinate also has a positive offset, but this is not as common.)

For illustration purposes, we start with the 3D case, and later extend it to the general case. We require one dimenson to be even and ≥ 16 and another dimension to be ≥ 12, which we assume w.l.o.g. to be the first two. The rest can be any size. Note that at least one dimension must be even, or no tour exists [49].

The construction works as follows. The 2D construction is reused at each level. However, there are only two actual junctions, one on the first layer, of height 5, and one on the last layer, which may have any of the four heights in Figure 4.4. Every other junction is replaced by a sequence of formation moves. At every layer except the last, the formation

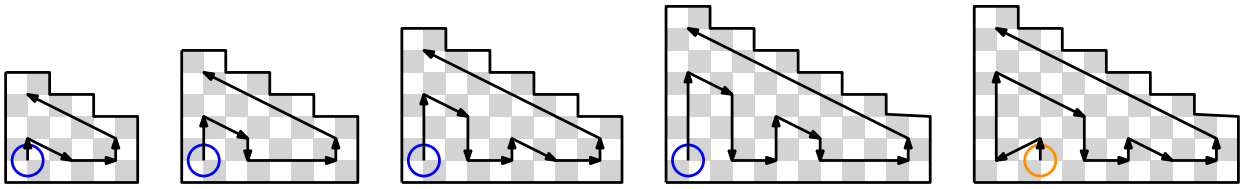


Figure 4.13: Corners where the knights stay in formation and end at specific positions.

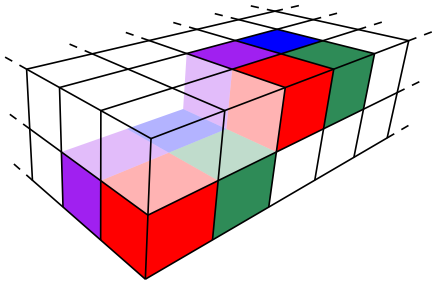


Figure 4.14: Formation move across layers. Each color shows the starting and ending position of one of the knights.

ends adjacent to the corner using one of the sequences of moves in Figure 4.13 (note that we show sequences for 4 different heights, thus guaranteeing that one shape fits for any board dimensions). The layers are connected with a formation move one layer up and two cells to the side, as in Figure 4.14. At every layer except the first, the formation starts with the rightmost sequence of moves in Figure 4.13.

Note that, since the sequence of moves between junctions is more involved than in two dimensions, Lemma 4.1 may not hold. There is, however, an easy fix: if the entire sequence is not a single cycle, replace the first junction with one that has a vertical matching (second junction in Figure 4.4, rotate 180 degrees). This then makes a cycle.

If the number of dimensions is higher than three, simply observe that the same move used between levels can also be used to jump to the next dimension; instead of changing by 1 the third coordinate, change the fourth. After the first such move, the formation will be at the "top" of the second 3D board, which can be traversed downwards. This can be repeated any number of times, and generalizes to any number of dimensions.

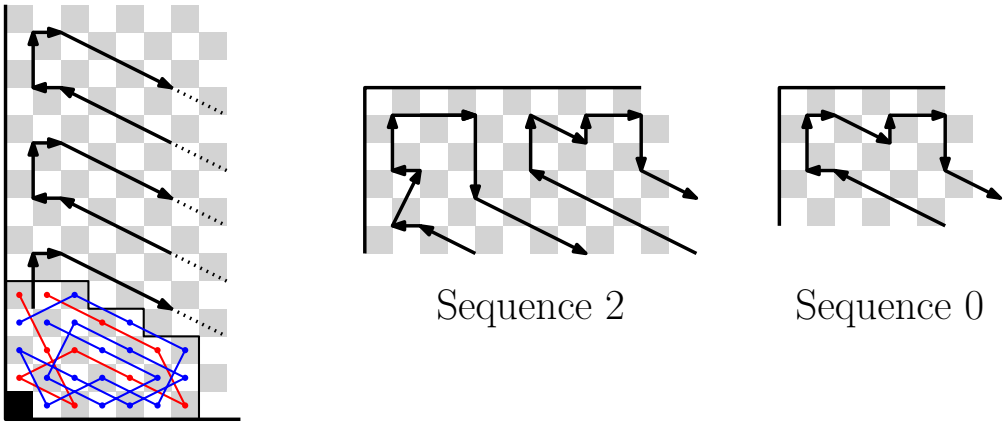


Figure 4.15: Adaptations required to add a row to the left of the normal construction, with a missing cell in the junction.

Note that in a n^d board, $\Omega(n^{d-1})$ turns are needed, because there are n^d cells and a turn must be made after at most $n/2$ moves. Note that our construction has $O(n^{d-1})$ turns, as it consists of n^{d-2} iterations of the 2D tour. Thus, it achieves a constant approximation ratio on the minimum number of turns. We do not know of any lower bound on the number of crossings in higher dimensions.

4.4.2 Odd boards

We show how to construct a tour for a 2D board with odd dimensions which visits every cell except a corner cell. This is used in the next section to construct a tour that is symmetric under 90° rotations.

Let the board dimensions be $w \times h$, where $w > 16$ and $h > 12$ are both odd. First, we use Algorithm 3 to construct a $(w-1) \times h$ tour which is missing the leftmost column. Then, we extend our tour to cover this column, except the bottom cell, with the variations of our construction depicted in 4.15. In particular, for the top-left corner, recall that we use sequence $(3-h) \bmod 4$ in Figure 4.5. Here, the height h is odd, so we only need adaptations for Sequences 2 and 0.

4.4.3 90 Degree Symmetry

In this section, we show how to construct a symmetric tour under 90 degree rotations. We say a tour is symmetric under a given geometric operation if the tour looks the same when the operation is applied to the board.

As a side note, our construction is already nearly symmetric under 180° rotations. For board dimensions such as 30×30 where opposite corners have the same shape, the only asymmetry is in the internal wiring of the junctions. However, the construction cannot easily be made fully symmetric. It follows from the argument in the proof of Lemma 4.1 that if the two non-junction corners are equal, the entire sequence of formation moves from one junction to the other is neutral. Thus, using the same junction in both corners, as required to have symmetry, would result in two disjoint cycles.

Symmetric tours under 90° rotations exist only for square boards where the size $n = 4k + 2$ is a multiple of two but not a multiple of four [40]. In [94], Parberry gives a construction for knight's tours missing a corner cell and then shows how to combine four such tours into a single tour symmetric under 90° rotations. We follow the same approach to obtain a symmetric tour with a number of turns and crossings linear on n, and thus constant approximation ratios.

In our construction from Section 4.4.2, cell $(0, 0)$ is missing, and edge $e = \{(0, 1), (2, 0)\}$ is present. This suffices to construct a symmetric tour. Divide the $2n \times 2n$ board into four equal quadrants, each of which is now a square board with odd dimensions. Use the construction for odd bords to fill each quadrant, rotated so that the missing cell is in the center. Finally, connect all four tours as in Figure 4.16.

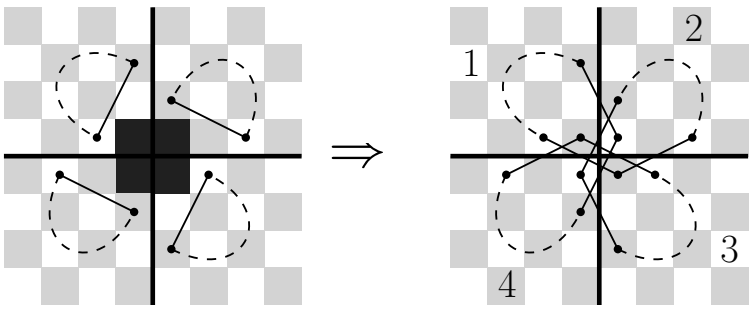


Figure 4.16: This transformation appears in [94]. **Left:** four tours missing a corner square and containing a certain edge. The dashed lines represent the rest of the tour in each quadrant, which cover every square except the dark square. **Right:** single tour that is symmetric under 90° rotations. The numbers on the right side indicate the order in which each part of the tour is visited, showing that the tour is indeed a single cycle.

4.4.4 Giraffe's tour

A giraffe is a leaper characterized by the move $(1, 4)$ instead of $(1, 2)$. Giraffe's tours are known to exist on square boards of even size $n \geq 104$ [72] and on square boards of size $2n$ when n is odd and bigger than 4 [40]. Our result extends this to some rectangular sizes.

We adapt our techniques for finding giraffe's tours with $O(w+h)$ turns and crossings, where w and h are the width and height of the board. We use a formation of 4×4 giraffes. Figure 4.17 shows the formation moves, Figure 4.18 shows the analogous of a heel to be used to transition between diagonals, and Figure 4.19 shows the two junctions. Figure 4.20 shows how these elements are combined to cover the board.

We restrict our construction to rectangular boards where $w = 32k + 20$, for some $k \geq 1$, and $h = 8l + 14$, for some $l \geq 1$ (extending the results to more boards would require additional heel variations). We start at the bottom-left junction as in the knight's tour. We transition between diagonals along the bottom edge with a giraffe heel, and along the top edge with a flipped giraffe heel. We transition between diagonals along the left and right edges with four consecutive upward moves. The junction has width 20 and each heel has width 32, so there are k heels along the bottom and top edges. The junction has height 11 and the tip

of the heel has height 3, so there are l sequences of four upward moves along each side (see Figure 4.20).

It is easy to see that the construction visits every cell. As in the case of knight's tours, for the result to be a valid giraffe's tour it should be a cycle instead of a set of disjoint cycles. Note that the matchings in the two junctions form a cycle. Thus, if the formation reaches the top-right junction in the same matching as they left the bottom-left junction, the entire construction is a single cycle (by an argument analogous to Theorem 4.1). Next, we argue that this is the case.

Let H, F, and U denote the sequences of formation moves in the heel, in the flipped heel, and the sequence of four upward moves, respectively. Let $T_{w,h}$ denote the entire sequence of moves from one junction to the other, where $w = 32k + 20$, for some $k \geq 1$, and $h = 8l + 14$, for some $l \geq 1$. Note that $T_{w,h}$ is a concatenation, in some order, of H k times, F k times, and U $2l$ times (we can safely ignore diagonal moves, which do not change the coordinates of the giraffes within the formation). Let M be the matching of the bottom-left junction. We want to argue that, after all the moves in $T_{w,h}$, the giraffes are still in matching M, that is, $T_{w,h}(M) = M$ using the notation from Section 4.2.

We show that not only the giraffes arrive to the other junction in the same matching but, in fact, they arrive in the same coordinates in the formation as they started. First, note that U has the effect of flipping column 1 with 2 and column 3 with 4 in the formation. Perhaps surprisingly, H and F have the same effect. This is tedious but can be checked for each giraffe (Figure 4.18 shows one in red). Therefore, $T_{w,h}$ is equivalent to U $2(l + k)$ times in a row. Note that after eight consecutive upward moves, or U twice, each giraffe ends where it started. Thus, this is true of the entire tour.

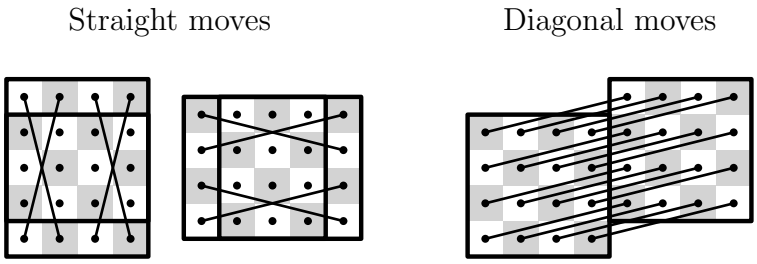


Figure 4.17: Formation of 16 giraffes moving together without leaving any unvisited squares.

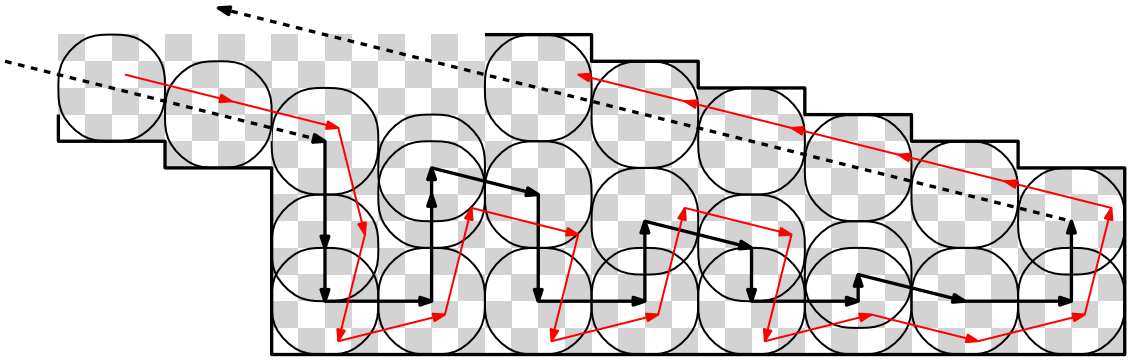


Figure 4.18: A giraffe heel. The formation moves are shown with black arrows (grouping up to four sequential straight moves together) The intermediate positions of the formation are marked by rounded squares, showing that every cell is covered. Note that the tip of the heel fits tightly under the next heel. The red line shows the path of one specific giraffe.

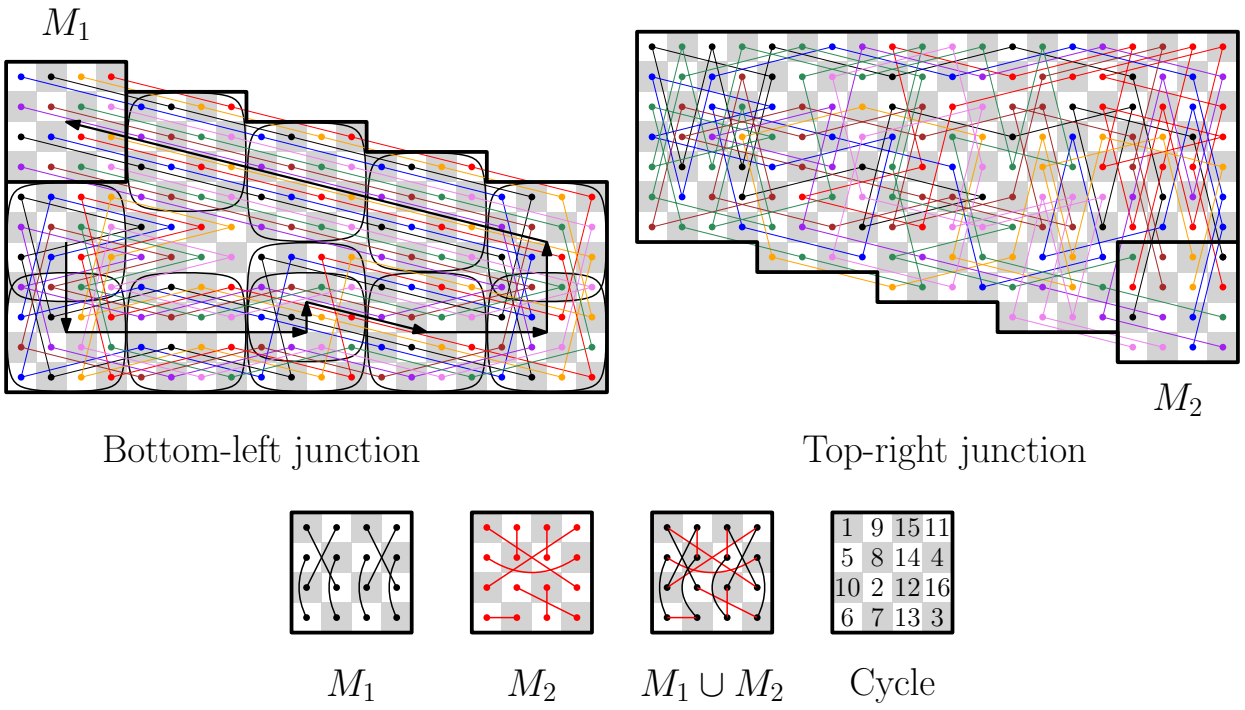


Figure 4.19: Two giraffe junctions, their corresponding matchings, and the union of their matchings. The bottom-left junction consists mostly of formation moves, whereas the top-right one was computed via brute-force search. The cycle through the edges of the union is shown with the index of each node.

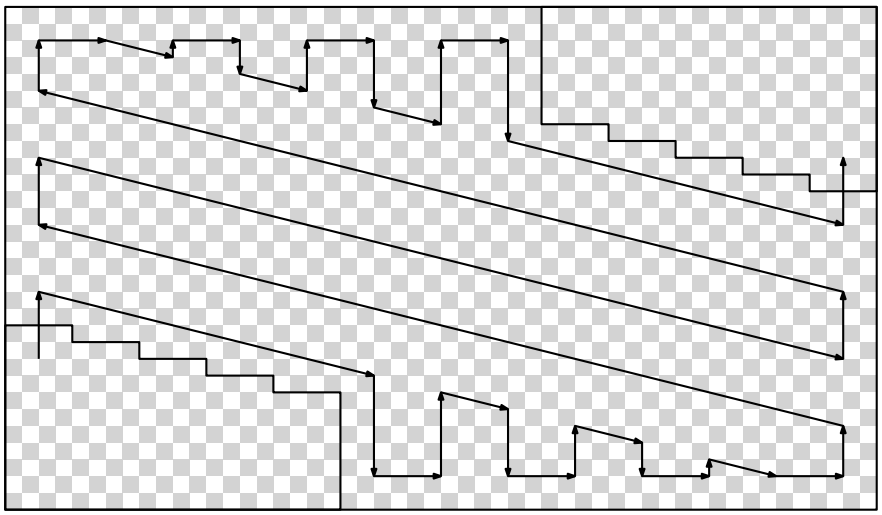


Figure 4.20: The formation moves of a giraffe's tour on a 52×30 board.

4.5 Conclusions

We have introduced two new metrics of "simplicity" for knight's tours: the number of turns and the number of crossings. We provided an algorithm which is within a constant factor of the minimum for both metrics. In doing so, we found that, in a $n \times n$ board, the minimum number of turns and crossings is $O(n)$. Prior techniques such as divide-and-conquer necessarily result in $\Theta(n^2)$ turns and crossings, so at the outset of this work it was unclear whether $o(n^2)$ could be achieved at all.

The ideas of the algorithm, while simple, seem to be new in the literature, which is interesting considering the history of the problem. Perhaps it was our *a priori* optimization goal that led us in a new direction. The algorithm exhibits a number of positive traits. It is simple, efficient to compute, parallelizable, and amenable to generalizations (see Section 4.4). We conclude with some open questions:

- Our tours have $9.5n + O(1)$ turns and $13n + O(1)$ crossings, and we showed respective lower bounds of $(6 - \varepsilon)n$ and $4n - O(1)$. The main open question is closing or reducing these gaps, as there may still be room for improvement in both directions. We conjecture that the minimum number of turns is at least $8n$.

- Are there other properties of knight's tours, besides turns and crossings, that might be interesting to optimize?

- Our method relies heavily on the topology of the knight-move graph. Thus, it is not applicable to general Hamiltonian cycle problems. Are there other graph classes with a similar enough structure that the ideas of formations and junctions can be useful?

Chapter 5

Geometric Polyhedral Point-Set Pattern Matching

5.1 Introduction

A notable recent computational geometry application is for tracking supply chains for natural diamonds, for which the industry and customers are strongly motivated to prefer ethically-sourced provenance (***e.g.***, to avoid so-called "blood diamonds"). For example, the ***Tracr*** system employs a blockchain for tracing the supply chain for a diamond from its being mined as a rough diamond to a customer purchasing a polished diamond [111]. (See Figure 5.1.)

Essential steps in the Tracr blockchain supply-chain process require methods to match point sets against geometric shapes, ***e.g.***, to guarantee that a diamond has not been replaced with one of questionable provenance [111]. Currently, the Tracr system uses standard machine-learning techniques to perform the shape matching steps; however, we believe better accuracy can be achieved by using computational geometry approaches. In particular, motivated by the Tracr application, we are interested in this chapter in efficient methods for matching

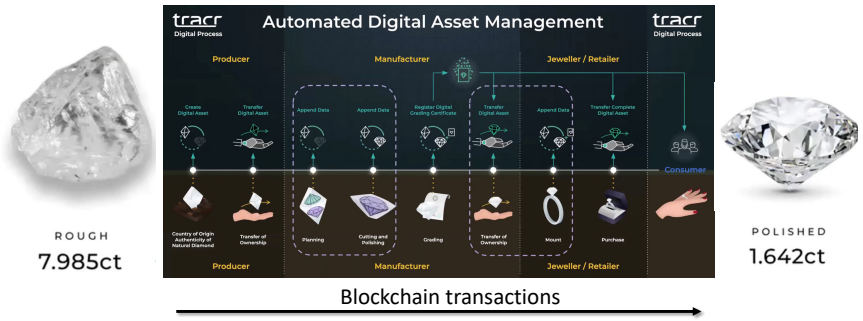


Figure 5.1: Blockchain transactions in a diamond supply chain, providing provenance, traceability, and authenticity of an ethically-sourced diamond.

point sets against geometric shapes, such as polyhedra. Formalizing this problem, we study the problem of finding the best translation and/or rotation of the boundary of a convex polytope, P (***e.g.***, defining a polished diamond shape), to match a set of n points in a d-dimensional ($d \geq 3$) space, where the point set is a "good" sample of the boundary of a polytope that is purported to be P. Since there may be small inaccuracies in the sampling process, our aim is to compute a ***minimum width*** polyhedral annulus determined by P that contains the sampled points. In the interest of optimizing running time, rather than seeking an exact solution, we seek an approximate solution that deviates from the real solution by a predefined quantity $\varepsilon > 0$.

Related Work. We are not familiar with any previous work on the problems we study in this chapter. Nevertheless, there is considerable prior work on the general area of matching a geometric shape to a set of points, especially in the plane. For example, Barequet, Bose, Dickerson, and Goodrich [16] give solutions to several constrained polygon annulus placement problems for offset and scaled polygons including an algorithm for finding the translation for the minimum offset of an m-vertex polygon that contains a set of n points in $O(n \log^2 n + m)$ time. Barequet, Dickerson, and Scharf [17] study the problem of covering a maximum number of n points with an m-vertex polygon (not just its boundary) under translations, rotations, and/or scaling, giving, ***e.g.***, an algorithm running in time $O(n^3 m^4 \log(nm))$ for the general

problem. There has also been work on finding a minimum-width annulus for rectangles and squares, ***e.g.***, see [58, 12, 14, 90].

Chan [32] presents a $(1+\varepsilon)$-approximation method that finds a minimum-width spherical annulus of n points in d dimensions in $O(n\log(1/\varepsilon)+\varepsilon^{O(1)})$ time, and Agarwal, Har-Peled, and Varadarajan [2] improve this to $O(n+1/\varepsilon^{O(d^2)})$ time via coresets [3, 96, 120, 4]. Arya, da Fonseca, and Mount [9] show how to find an ε-approximation of the width of n points in $O(n\log(1/\varepsilon)+1/\varepsilon^{(d-1)/2+\alpha})$ time, for a constant $\alpha>0$. Bae [13] shows how to find a min-width d-dimensional hypercubic shell in $O(n^{\lfloor d/2\rfloor}\log^{d-1}n)$ expected time.

Our Results. Given a set of n points in $\mathbf{R}^d$, we provide an $O(\varepsilon^{-d}n)$-time $(1+\varepsilon)$-approximate polytope-matching algorithm under translations, for $d\geq 3$, and $O(n\log\varepsilon^{-1}+\varepsilon^{-2})$ time for $d=2$, and we provide an $O(f^{d-1}\varepsilon^{1-2d}n)$-time algorithm when also allowing for rotations, where the complexity of the polytope is constant.

5.2 Preliminaries

Definition 5.1. *Following previous convention [7, 10, 11, 50, 8], we say that a point set S is a δ-**uniform sample** of a surface $\Sigma\subset\mathbb{R}^d$ if for every point $p\in\Sigma$, there exists a point $q\in S$ such that $d(p,q)\leq\delta$.*

Definition 5.2. *Let $C\subset\mathbb{R}^d$ be a polyhedron containing the origin. Given C, and $x\in\mathbb{R}^d$, define $x+C=\{x+y : y\in C\}$ (the translation of C by x), and for $r\in\mathbb{R}$, define $rC=\{ry : y\in C\}$. A **placement** of C is a pair (x,r), where $x\in\mathbb{R}^d$ and $r\in\mathbb{R}^{\geq 0}$, representing the translated and scaled copy $x+rC$. We refer to x and r as the **center** and **radius** of the placement, respectively. Two placements are **concentric** if they share the same center.*

Definition 5.3. *Let C be any closed convex body in $\mathbb{R}^d$ containing the origin in its interior. The convex distance function induced by C is the function $d_C : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}^{\geq 0}$, where $d_C(p,q) = \min\{r : r \geq 0 \text{ and } q \in p + rC\}$. Thus, the convex distance between p and q is determined by the minimum radius placement of C centered at p that contains q (see Figure 5.2). When C is centrally symmetric, this defines a metric, but for general C, the function d_C may not be symmetric. We call the original shape C the **unit ball** U_C under the distance function d_C. Note that $d_C(a,c) = d_C(a,b) + d_C(b,c)$ when a, b and c are colinear and appear in that order.*

Definition 5.4. *Define an **annulus** for C to be the set-theoretic difference of two concentric placements $(p + RC) \setminus (p + rC)$, for $0 \leq r \leq R$. The **width** of the annulus is $R - r$.*

Given a δ-uniform sample of points, S, there are three placements of C we are interested in:

Definition 5.5. ***Minimum enclosing ball (MinBall)**: A placement of C of the smallest radius that contains all of the points in S.*

Definition 5.6. ***Maximum enclosed ball (MaxBall)**: A placement of C of the largest radius, centered within the convex hull of S, that contains no points in S.*

Definition 5.7. ***Minimum width annulus (MWA)**: Given a set $S \subset \mathbb{R}^d$ and a convex body C, the minimum width annulus of S is the annulus for C of the smallest width that contains S.*

Note that, following the definition of the MaxBall in Definition 5.6, we require that the center of the MWA must also lie within the convex hull of S. For each of the above placements, we also refer to parameterized versions, for example MinBall(p), MaxBall(p), or MWA(p). These respectively refer to the minimum enclosing ball, maximum enclosed ball, or minimum width annulus centered at the point p.

Further, we use |MinBall(p)| and |MaxBall(p)| to denote the radius of MinBall(p) and MaxBall(p), respectively, and we use |MWA(p)| to denote the width of MWA(p).

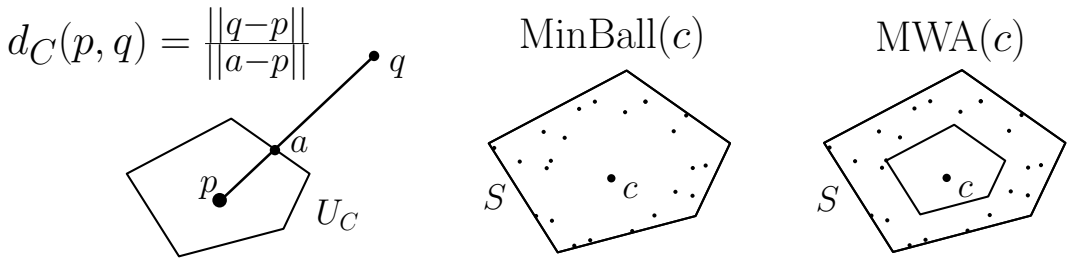


Figure 5.2: **Left:** a visual representation of a polyhedral distance function and the distance between two points. **Center:** The MinBall under d_C containing all points in S, centered at c. **Right:** The MWA of S with all points within MinBall(c)\MaxBall(c).

Definition 5.8. ***Fatness:*** *The fatness, F, of $S \subset \mathbb{R}^d$ under d_C as the ratio of the* MinBall *over the* MaxBall*, such that $F :=$ |MinBall|/|MaxBall|. Note that S can either be a closed body in $\mathbb{R}^d$ or sample of a surface.*

Definition 5.9. ***Concentric Fatness:*** *Extend the notion of fatness such that the concentric fatness, $F_c :=$ |MinBall(c)|/|MaxBall(c)| such that c is the center of the MWA.*

Definition 5.10. ***Slimness:*** *Conversely to concentric fatness, define the slimness, f, as the ratio of the width of the* MWA *centered at c over the radius of* MinBall*(c), such that $f :=$ |MinBall(c)|/|MWA| and $f^{-1} = 1 - F_c^{-1}$.*

Remark 5.1. *In order for a δ-uniform sample to represent the surface, Σ, with enough accuracy for a meaningful* MWA*, the sample must contain at least one point between corresponding facets of the* MWA*. Where corresponding facets refer to facets of the* MinBall *and* MaxBall *representing the same facet of U_C. Therefore, in the remainder of the chapter, we assume we have a δ-uniform sample and that δ is small enough to guarantee this condition for even the smallest facets.*

In practice, it would be easy to determine a small enough δ before sampling Σ, since only sufficiently slim surfaces would benefit from finding the MWA and very fat surfaces would yield increasingly noisy MaxBall. Thus, setting δ to the smallest facet of the MinBall and scaling down by an arbitrary constant larger than the maximum expected fatness, such as 100, will easily yield a small enough δ.

Also, note that, for a center point c, the problem of finding MWA(c) has a unique solution, because a unit ball, U_C, is convex and once the placement's radius grows past $d_C(c,p)$, it must contain $p \in S$. Thus, the inner and outer radii are defined as $\min_{p\in S} d_C(c,p)$ and $\max_{p\in S} d_C(c,p)$, respectively. Further, let us assume that the reference polytope defining our polyhedral distance function has m facets, where m is a fixed constant, since the sample size is expected to be much larger than m. Thus, d_C can be calculated in $O(m)$ time; hence, MWA(c) can be found in $O(mn)$ time, which is $O(n)$ under a fixed polytope.

5.3 Approximating the Minimum Width Annulus

Let us first describe how to find a constant factor approximation of MWA under translations. Note that, by assumption on Definition 5.6, the center c of our approximation lies within the convex hull of S. Let us denote the center, outer radius, inner radius, and width of the optimal MWA as c_{opt}, R_{opt}, r_{opt}, and w_{opt}.

Lemma 5.1. *The center of the* MWA*, c_{opt}, is under w_{opt} distance away from the center of the* MinBall*, c. That is, $d_C(c, c_{opt}) \leq w_{opt}$.*

Proof. Recall our assumption from Remark 5.1. Since some points must always exist in each facet, the MinBall cannot shrink past any facets of MaxBall(c). Suppose for contradiction that $d_C(c, c_{opt}) > w_{opt}$. Let s be the point where a ray projected from c through c_{opt} intersects the boundary of MaxBall(c_{opt}) and let R denote the radius of the MinBall.

$$
\begin{aligned}
R > d_C(c,s) &= d_C(c, c_{opt}) + d_C(c_{opt}, s) && \text{by colinearity} \\
&> w_{opt} + d_C(c_{opt}, s) && \text{by assumption} \\
&> w_{opt} + r_{opt} && \text{by MaxBall}(c_{opt}).
\end{aligned}
$$

Thus, since $w_{opt} + r_{opt} = R_{opt}$, we find $R > R_{opt}$, which is a contradiction since R must be the smallest radius of the MinBall across all possible centers. Therefore, we have that $d_C(c, c_{opt})$ cannot be larger than w_{opt}. □

Lemma 5.1 helps us constrain the region within which c must be contained. Let us now reason about how different center points would serve as approximations.

Lemma 5.2. *Suppose c is an arbitrary center-point in our search region, and the two directed distances between c and c_{opt} are at most t,* ***i.e.****, $t \geq \max\{d_C(c, c_{opt}), d_C(c_{opt}, c)\}$. Then, we have that $|\text{MWA}(c)| \leq w_{opt} + 2t$.*

Proof. Let p be the point where the ray from c through c_{opt} intersects the boundary of MinBall(c_{opt}). In the worst case, MinBall(c) would need to contain p; hence,

$$d_C(c, p) = d_C(c, c_{opt}) + d_C(c_{opt}, p) \leq t + d_C(c_{opt}, p)$$
$$R \leq R_{opt} + t.$$

Conversely, let q be the intersection point where the ray projected from c_{opt} through c intersects the boundary of MaxBall(c_{opt}). In the worst case, MaxBall(c) would need to exclude q; hence, in which case

$$d_C(c, q) = d_C(c_{opt}, q) - d_C(c_{opt}, c) \geq d_C(c_{opt}, q) - t$$
$$r \geq r_{opt} - t.$$

Putting together these worst cases for MinBall(c) and MaxBall(c) implies that

$$|\text{MWA}(c)| \leq w_{opt} + 2t$$

□

Definition 5.11. *For simplicity, let us consider two points a, b to be* ***t-close*** *(under C) whenever $t \geq \max\{d_C(a,b), d_C(b,a)\}$.*

Lemma 5.3. *If c is the center of* MinBall*, then* MWA(c) *is a constant factor approximation,* ***i.e.****, $|\text{MWA}(c)| \leq b|\text{MWA}|$, for some constant $b \geq 1$, under translations.*

Proof. From Lemma 5.1, we have that $d_C(c, c_{opt}) \leq w_{opt}$. If c and c_{opt} are w_{opt}-close, then we can directly apply the second part of Lemma 5.2 to find $r \geq r_{opt} - w_{opt}$ and $R \leq R_{opt}$, such that $|\text{MWA}(c)| \leq R_{opt} - (r_{opt} - w_{opt})$, thus proving that this is a 2-approximation. If d_C is a metric, then $d_C(c_{opt}, c) = d_C(c, c_{opt})$ and this must always be the case. However, if $d_C(c_{opt}, c) > w_{opt}$, then we must use the Euclidean distance to find $d_C(c_{opt}, c)$. Let vector $u := c - c_{opt}$, and let us define unit vectors with respect to d_C and its inverse, $d_{\overline{C}}$, such that

$$\hat{u_C} = \frac{u}{d_C(c_{opt}, c)} \quad , \quad \hat{u_{\overline{C}}} = \frac{\overline{u}}{d_C(c, c_{opt})}$$

$$||\hat{u_C}||d_C(c_{opt}, c) = ||u|| = ||\hat{u_{\overline{C}}}||d_C(c, c_{opt})$$

$$d_C(c_{opt}, c) \leq \frac{||\hat{u_{\overline{C}}}||}{||\hat{u_C}||} w_{opt} \qquad \text{from Lemma 5.1.}$$

Under any convex distance function, $\frac{||\hat{u_{\overline{C}}}||}{||\hat{u_C}||}$ is bounded from above by $A = \max_{v \in \mathbb{R}^d} \frac{||\hat{v_{\overline{C}}}||}{||\hat{v_C}||}$, which corresponds to finding the direction, v, of the largest asymmetry in U_C. Thus, by Lemma 5.2, $|\text{MWA}(c)| \leq (A+1)w_{opt}$. Under our (fixed) polyhedral distance function, A is constant; hence, MWA(c) is a constant-factor approximation. □

5.3.1 Achieving a $(1+\varepsilon)$-approximation.

Let us now describe how to compute a $(1+\varepsilon)$-approximation of MWA.

Lemma 5.4. *Suppose c_{opt} and c are $(\varepsilon w/(2b))$-close, where $w = |\text{MWA}(c_M)|$, c_M is the center of* MinBall*, and b is the constant from Lemma 5.3. Then,* MWA(c) *is a $(1+\varepsilon)$-approximation of* MWA *under translations.*

Proof. To be a $(1+\varepsilon)$-approximation of MWA, the width of our approximated annulus must be at most $(1+\varepsilon)$ times the width of the optimal one. Assuming c and c_{opt} are t-close, and using Lemma 5.2, we require that $w_{opt} + 2t \leq (1+\varepsilon)w_{opt}$, ***i.e.***, $t \leq \varepsilon w_{opt}/2$. Let us then choose $t \leq \varepsilon w/(2b)$, knowing that $w \leq bw_{opt}$ from Lemma 5.3, which is sufficient for achieving a $(1+\varepsilon)$-approximation. □

Knowing how close our approximation's center must be, we can now put together a $(1+\varepsilon)$-approximation algorithm to find a center satisfying this condition.

Theorem 5.1. *One can achieve a $(1+\varepsilon)$-approximation of the* MWA *under translations in $O(\varepsilon^{-d}n)$ time.*

Proof. The MinBall can be computed in $O(n)$ time through a linear program [39]. By Lemma 5.1, we have that $d_C(c, c_{opt}) \leq w_{opt}$, where c is the MinBall center. This implies that c_{opt} must lay within the placement $c + w_{opt}C$ or more generously in P, defined as $c + wC$. Furthermore, from Lemma 5.4, we know that being $(\varepsilon w/(2b))$-close to c_{opt} suffices for a $(1+\varepsilon)$-approximation. Therefore, overlaying a grid G that covers P, such that any point in $p \in P$ is $(\varepsilon w/(2b))$-close to a gridpoint, guarantees the existence of a point $g \in G$ for which MWA(g) is a $(1+\varepsilon)$-approximation.

Since P and $(\varepsilon w/(2b))$-closeness are both defined under d_C, we translate this to a cubic grid for simplicity. Let Q be the smallest cube enclosing P and q be the largest cube enclosed by $(\varepsilon w/(2b))C$. Let us now define a grid, G, to span Q with cubes the size of q. This grid, G, has points Fb/ε apart and $F^d b^d \varepsilon^{-d}$ gridpoints in total, where F corresponds to the fatness of C under the distance function defined by the unit cube (as defined in Definition 5.8).

Let us use q to define the distance function d_q where the unit ball U_q is $(\varepsilon w/(2b))$ times smaller than q. The grid G guarantees that for every point p, there exists a gridpoint $g \in G$ such that $d_q(p, g) \leq \varepsilon w/(2b)$. Since the unit cube is contained within the unit polyhedron, we have that $d_C(a, b) \leq d_q(a, b)\ \forall a, b$; and since d_q defines a metric, p must also be $(\varepsilon w/(2b))$-close under d_C. Calculating the width of $|\text{MWA}(g)| \forall g \in G$ takes $O(F^d b^d \varepsilon^{-d} n)$ time, and therefore finding the gridpoint providing the $(1+\varepsilon)$-approximation also takes $O(F^d b^d \varepsilon^{-d} n)$ time,[1] which, under a fixed d_C, is $O(\varepsilon^{-d} n)$ time. □

5.3.2 Faster grid-search in two dimensions.

The algorithm of Theorem 5.1 recalculates the MWA at every gridpoint. However, small movements along the grid should not affect the MWA much. We use this insight to speed up MWA recalculations for two dimensions.

Let us first define the ***contributing edge*** of a sample point, $p \in S$, as the edge of $C + g$ intersected by the ray emanating from a gridpoint, g, towards p. Under this center-point, p will only directly affect the placement of the contributing edge. In Figure 5.2, if p is a gridpoint then q's contributing edge would be the edge overlaying a.

Observe that given vectors $\overrightarrow{v} \in C$, defined as the vectors directed from the center towards each vertex, the planar subdivision, created by rays for each $\overrightarrow{v}$ originating from g, separates points by their contributing edge. For any two gridpoints, g_1 and g_2, and rays projected from them parallel to $\overrightarrow{v}$, points within these two rays will contribute to different edges under g_1 and g_2. We denote this region as the ***vertex slab*** of vertex v (see Figure 5.3).

Conversely define ***edge slabs*** as the regions outside the vertex slabs. Points within an edge slab will contribute to the same edge under both gridpoints, and so maintaining the

[1]For metrics, MinBall provides a 2-approximation, thus $b = 2$. For non-metrics, we can remove this constant by first using this algorithm with $\varepsilon = 1$ in order to find a 2-approximation in linear-time, and using this approximation for gridding in the main step.

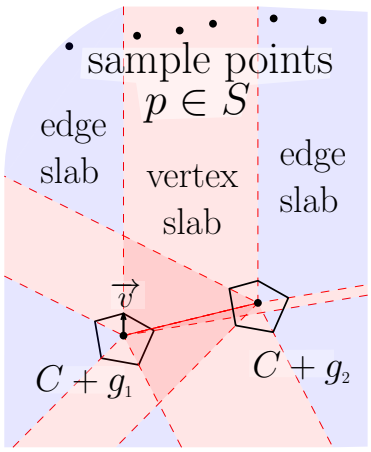


Figure 5.3: Planar subdivision defining vertex slabs (red) and edge slabs (blue) for two candidate center-points, and showing membership of some sample points.

constraints it imposes on the MWA can therefore be achieved with two extreme-most points per edge slab.

If we consider vertex slabs for all $g \in G$, we must be able to quickly calculate the strictest constraints imposed by points in a subset of vertex slabs. An example of the planar subdivision for two points is shown in Figure 5.3.

Given a grid G, we write $g_{i,j} \in G$ to be the gridpoint at index (i, j). Consider the set of all grid lines L_v defined by rays parallel to $\overrightarrow{v}$ starting at each gridpoint. To quickly recalculate changes to edges incident on v as we traverse through gridpoints, we need to quickly identify which slab a sample-point p belongs to, given a planar subdivision defined by L_v.

Lemma 5.5. *For a specific vector $\overrightarrow{v}$ and an $m \times m$ grid, we can identify which slab a sample point, p, belongs to in $O(\log m)$ time with $O(m^2)$-time preprocessing.*

Proof. Consider the orthogonal projection of grid lines in L_v onto a line $\overrightarrow{v_\perp}$ perpendicular to $\overrightarrow{v}$, the order in which these lines appear in $\overrightarrow{v_\perp}$ defines the possible slabs that p could belong to (see Figure 5.4a). We can project a given grid line $l \in L_v$ onto $\overrightarrow{v_\perp}$ in constant time. After sorting these grid lines, we can perform a binary searchfor the projected sample point

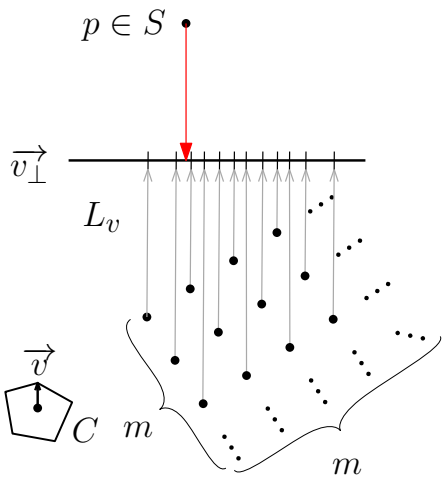
through the m^2 projected gridpoints in $O(\log m)$ time to identify the slab which p would belong to.

Using general sorting algorithms, we could sort the grid lines in $O(m^2 \log m)$ time. However, since these lines belong to a grid, we can exploit the uniformity to sort them in only $O(m^2)$ time. Consider the two basis vectors defining gridpoint positions $\hat{\imath} = g_{(1,0)} - g_{(0,0)}$ and $\hat{\jmath} = g_{(0,1)} - g_{(0,0)}$, and their sizes after orthogonal projection onto $\overrightarrow{v_\perp}$, $|\hat{\imath}_\perp|$, and $|\hat{\jmath}_\perp|$. Without loss of generality, assume that $|\hat{\imath}_\perp| \geq |\hat{\jmath}_\perp|$, in which case grid lines originating from adjacent gridpoints in the same row must be exactly $|\hat{\imath}_\perp|$ away. In addition, any region $|\hat{\imath}_\perp|$-wide, that does not start at a grid line, must contain at most a single point from each row. Furthermore, since points in the same row are always $|\hat{\imath}_\perp|$ away, they must appear in the same order in each region.

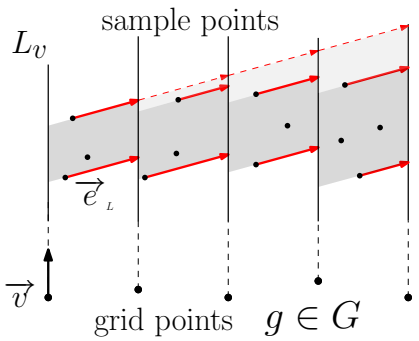
We can therefore initially split $\overrightarrow{v_\perp}$ into regions $|\hat{\imath}_\perp|$ wide. Sorting the grid lines $l \in L_v$ into their region can therefore be calculated in $O(m^2)$ time. Now we can sort the m points in the region containing points from every row in $O(m \log m)$ time. Since each region has the same order, we can place points in other regions by following the order found in our sorted region, thus taking $O(m^2)$ preprocessing time for sorting the points. □

Recall that points to the left of a given line $l \in L_v$ contribute to the edge to the left of v, ***i.e.***, all points belonging to slabs to the left of l. We can therefore isolate the points in these slabs causing the largest potential change in MWA.

Lemma 5.6. *For a vertex $v \in C$ and grid line $l \in L_v$ through gridpoint g, let l_L and l_R refer to the slabs on the subdivision imposed by L_v immediately to the left and right of l, respectively. Assuming l_L maintains the points to the left of l imposing the strictest constraints on* $\mathrm{MWA}(g)$*, and l_R to the right, one can calculate* $\mathrm{MWA}(g)$ *in $O(1)$ time.*



(a) A demonstration of the point location problem with the subdivision, L_v, and a visualization of the gridpoints and sample point projections onto $\overrightarrow{v_\perp}$.



(b) Finding the extreme-most points (red) under $\overrightarrow{e}_L$ in subdivision L_v for each region (solid) and for all regions to its left (dashed).

Figure 5.4: A visual representation of the projections involved while point locating within the vertex slabs and while finding the extreme-most points in each slab.

Proof. Finding $\min_{p \in S} d_C(g, p)$ and $\max_{p \in S} d_C(g, p)$ can now be achieved by optimizing only over the set of points in $\{l_L \cup l_R \ \forall v \in C\}$ and all points in edge slabs. This set would contain two points per vertex and two points per edge, yielding a constant number of points. Thus, MWA(g) can be found in constant time. □

Theorem 5.2. *A* $(1+\varepsilon)$*-approximation of the* MWA *in two dimensions can be found in* $O(n \log \varepsilon^{-1} + \varepsilon^{-2})$ *time under translations.*

Proof. For each vertex, v, we use Lemma 5.5 to identify the slab for every sample point. For each slab, we maintain only the two extreme-most points for each of the edges incident on $\overrightarrow{v}$. Let $\overrightarrow{e}_L \in C$ denote the vector describing the edge incident on $\overrightarrow{v}$ from the left, and vice versa for $\overrightarrow{e}_R \in C$ incident from the right. For each slab, we maintain only points which when projected in the relevant direction, $\overrightarrow{e}$, cause the furthest and closest intersections with the boundary (shown for $\overrightarrow{e}_L$ in Figure 5.4b). With a left-to-right pass, we update a slab's extreme-most points relative to $\overrightarrow{e}_L$ to maintain the extreme-most points for itself and slabs

to its left. With a right-to-left pass, we do the same for $\overrightarrow{e}_R$ and maintain extreme-most points in its slab and slabs to its right.

Thus, for each vertex, we create the slabs in $O(\varepsilon^{-2})$ time, identify a sample points slab in $O(\log \varepsilon^{-1})$ time per sample point, and track the extreme-most points per slab in constant time per sample point. With $O(\varepsilon^{-2})$ time to update the slabs after processing all sample points, and with $O(\varepsilon^{-2})$ time we can update the slabs such that they hold the extreme-most points across all slabs to their left or right (relative to $\overrightarrow{e}_L$ and $\overrightarrow{e}_R$, respectively).

For each edge slab, finding the extreme-most points is much simpler since finding $\min d_C(g, p)$ and $\max d_C(g, p)$ across all points in the edge slab will always be based on the contributing facet.

Thus, after finding the extreme-most points in both vertex slabs and edge slabs, we can calculate $\text{MWA}(g)$ in constant time as described in Lemma 5.6. Taking $O(\varepsilon^{-2})$ time to find $\min_{g \in G} |\text{MWA}(g)|$, which by Theorem 5.1 provides a $(1+\varepsilon)$-approximation of the minimum width annulus, completes the proof of the claimed time bound. □

5.4 Approximating MWA allowing rotations

In this section we consider rotations. As with Lemma 5.4, our goal is to find the maximum tolerable rotation sufficient for a $(1+\varepsilon)$-approximation. Observe that when centered about the global optimum, the solution found under both rotation and translation is at least as good as the solution found solely through rotation (***i.e.***, under a fixed center). We will therefore first prove necessary bounds for a $(1+\varepsilon)$-approximation under rotation only with the understanding that they remain when also allowing for translation.

Consider the polyhedral cone around $\overrightarrow{v}$ and define the ***bottleneck angle*** as the narrowest angle between a point on the surface of the polyhedral cone and $\overrightarrow{v}$. Let θ be the smallest bottleneck angle across all $\overrightarrow{v} \in C$. Let $\text{MWA}_\alpha(c)$ describe the MWA centered at c, where C has been rotated by angle α. Let us also use similar notations for MinBall and MaxBall.

Lemma 5.7. *Rotating by α causes $\text{MinBall}_\alpha(c)$ to grow by at most $\frac{\sin(\pi-\theta-\alpha)}{\sin\theta}$ (and the reciprocal for $\text{MaxBall}_\alpha(c)$).*

Proof. In the worst case, $\text{MinBall}(c)$ must be completely contained within $\text{MinBall}_\alpha(c)$. Let us now consider the triangle formed between c, the vertex v of the original MinBall, v_0, and the rotated vertex v_α (shown in Figure 5.5a). Since our calculations focus towards the same vertex, we will be working with Euclidean distances. The quantity $|v_0 - c|$ defines the radius r_1 of the original polyhedron, and $r_2 = |v_\alpha - c|$ the radius of the rotated one. With $\gamma = \pi - \theta - \alpha$ as the remaining angle in our triangle and using the sine rule, we find that

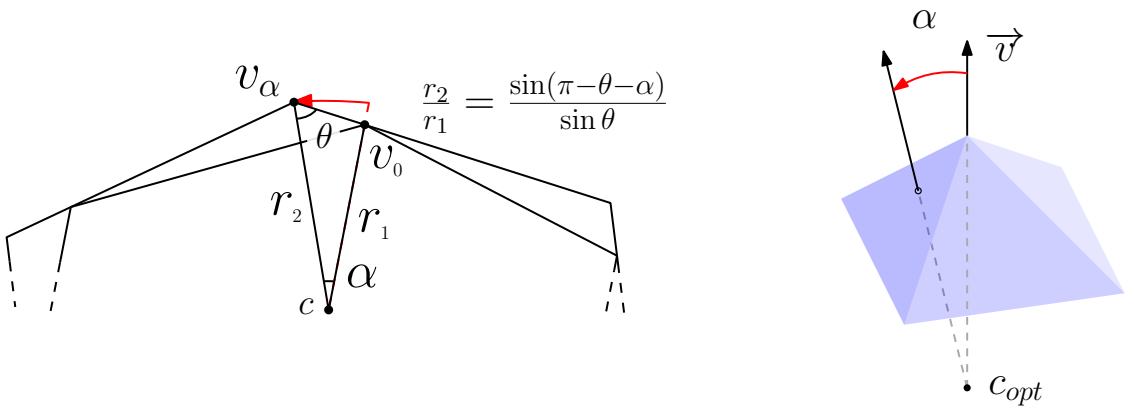
$$\frac{r_2}{r_1} = \frac{\sin\gamma}{\sin\theta} = \frac{\sin(\pi-\theta-\alpha)}{\sin\theta}.$$

Observe that θ is the angle maximizing this scale difference. This applies to rotating by α in any direction about $\overrightarrow{v}$, and since this direction needs not coincide with θ, the scaled polyhedron might not touch the original. For $\text{MaxBall}_\alpha(c)$ to be contained within $\text{MaxBall}(c)$, the same example holds after switching references to the scaled and original. In this case, θ minimizes r_1/r_2. □

Let us now determine the rotation from the optimal orientation that achieves a $(1+\varepsilon)$-approximation.

Lemma 5.8. *Given a center c, we have that $\text{MWA}_\alpha(c)$ is a $(1+\varepsilon)$-approximation when*

$$\alpha \leq \arcsin\left(\frac{\sin\theta}{2f}\left(1+\varepsilon \pm \sqrt{(1+\varepsilon)^2 + 4f(f-1)}\right)\right) - \theta.$$



(a) A demonstration of the scale increase necessary for a polyhedron rotated by α to contain the original.

(b) Visualization of the rotation by α in an arbitrary direction about $\overrightarrow{v}$.

Figure 5.5: Visual representations for the effect of rotating by α, demonstrating the scale increase and demonstrating how a rotation by α is defined for higher dimensions.

Proof. Define f as the ratio of the radius of MinBall(c_{opt}) to w_{opt} (***i.e.***, $fw_{opt} = |\text{MinBall}(c_{opt})|$). Note that f corresponds to the inverse of the concentric slimness of S under d_C over all rotations of C.

Using Lemma 5.7, we know that

$$|\text{MWA}_\alpha(c)| \leq \frac{\sin\gamma}{\sin\theta}|\text{MinBall}(c)| - \frac{\sin\theta}{\sin\gamma}|\text{MaxBall}(c)|$$

$$\frac{\sin\gamma}{\sin\theta} f w_{opt} - \frac{\sin\theta}{\sin\gamma}(f-1)w_{opt} \leq (1+\varepsilon)w_{opt} \tag{5.1}$$

$$\frac{\sin\gamma}{\sin\theta} f - \frac{\sin\theta}{\sin\gamma}(f-1) \leq (1+\varepsilon) \tag{5.2}$$

To be a $(1+\varepsilon)$-approximation, we need $|\text{MWA}_\alpha(c)| \leq (1+\varepsilon)w_{opt}$ imposing the right side of Relation 5.1, its left side follows by definition of f, and Relation 5.2 by cancellation of w_{opt}. Since θ is constant, we can rearrange the above into a quadratic equation and solve for $\sin\gamma$.

$$\sin\gamma = \frac{\sin\theta}{2f}\left(1+\varepsilon \pm \sqrt{(1+\varepsilon)^2 + 4f(f-1)}\right). \tag{5.3}$$

However, arcsin will find $\gamma \leq \pi$, whereas we need the obtuse angle $\pi - \gamma$. Thus, proving this lemma's titular bound, and achieving a $(1+\varepsilon)$-approximation. □

Let us now establish a more generous lower-bound that will prove helpful when developing algorithms.

Lemma 5.9. *The angular deflection required for a $(1+\varepsilon)$-approximation is larger than $\theta\varepsilon/(2f)$.*

Proof. Observe that γ is of the form $\arcsin(k\sin\theta)$ and thus, in order for $\alpha = \gamma - \theta$ to be positive, we must have $\theta < \pi/2$ and $k > 1$. We will prove this is the case.

$$k = \frac{1+\varepsilon}{2f} + \sqrt{\left(\frac{1+\varepsilon}{2f}\right)^2 - \frac{1}{f} + 1} \tag{5.4}$$

$$\sqrt{\frac{1}{4f^2} - \frac{1}{f} + 1} = \left|1 - \frac{1}{2f}\right| \tag{5.5}$$

$$k > \frac{1+\varepsilon}{2f} + \left|1 - \frac{1}{2f}\right| = 1 + \frac{\varepsilon}{2f} \tag{5.6}$$

Equation (5.4) follows from Equation (5.3) after expanding. Relation (5.6) follows after using Equation (5.5) as a lower bound for the square root term in Equation (5.4) since $\varepsilon > 0$ and $f > 1$. This allows us to bound $\arcsin\left(\left(1 + \frac{\varepsilon}{2f}\right)\sin\theta\right)$ by using Taylor's series expansion to find $(1+k)\cdot\theta \leq \arcsin((1+k)\sin\theta)$, thus proving that the bound from Lemma 5.8 is greater than $\frac{\theta\varepsilon}{2f}$. □

Lemma 5.10. *For fixed rotation of C, assume we have an $O(g(n))$-time algorithm for the optimal minimum-width annulus under translation. We can find a $(1+\varepsilon)$-approximation of the* MWA *under rotations and translations in $O(f^{d-1}\varepsilon^{1-d}g(n))$ time.*

Proof. A d-dimensional shape has a $(d-1)$-dimensional axis of rotation. Let us evenly divide the unit circle into k directions. Let us also define a collection of all possible direction combinations as a grid of directions. For each grid direction, rotate C by the defined direction and calculate the MWA in $O(g(n))$ time. The optimal orientation must lie between the $(d-1)$-dimensional cube formed by 2^{d-1} grid directions. Therefore, as long as the diagonal is

smaller than $\frac{\theta\varepsilon}{f}$, there will always exist a grid direction within $\frac{\theta\varepsilon}{2f}$ of the optimal orientation and therefore achieving a $(1+\varepsilon)$-approximation by Lemma 5.9. Thus, we can achieve a $(1+\varepsilon)$-approximation in $O\left(g(n)\cdot\left(\frac{2\pi f\sqrt{d-1}}{\theta\varepsilon}\right)^{d-1}\right)$ time, where d, θ are constant under a fixed distance function d_C and f depends on the inverse of the slimness of the pointset. □

With a fixed center, Lemma 5.10 can be used to approximate MWA under rotations in $O(f^{d-1}\varepsilon^{1-d}n)$ time.

Theorem 5.3. *One can find a* $(1+\varepsilon)$*-approximation of* MWA *under rotations and translations in* $O(f^{d-1}\varepsilon^{1-2d}n)$ *time for* $d \geq 3$*, and* $O(fn\varepsilon^{-1}\log\varepsilon^{-1}+\varepsilon^{-3})$ *time for* $d=2$*.*

Proof. Consider using an approximation algorithm (from Theorems 5.1 or 5.2) instead of an exact algorithm as in Lemma 5.10. Let us define $(1+\xi)$ as the approximation ratio necessary from the subroutines in order to achieve an overall approximation ratio of $(1+\varepsilon)$, such that $(1+\xi)^2 = 1+\varepsilon$. Since $\xi = \sqrt{1+\varepsilon}-1$ and $0<\varepsilon<1$, then ξ must always be larger than $(\sqrt{2}-1)\varepsilon$, and thus, we can always pick a value for ξ which is $O(\varepsilon)$ and achieves the desired approximation. Thus, by following Lemma 5.10, we can find a $(1+(\sqrt{2}-1)\varepsilon)$-approximation using the $(1+(\sqrt{2}-1)\varepsilon)$-approximation algorithm from Theorem 5.1 to find a $(1+\varepsilon)$-approximation in $O(f^{d-1}\varepsilon^{1-d}\cdot\varepsilon^{-d}n)$ time. Alternatively, for two dimensions, we can instead use the algorithm from Theorem 5.2 to find a $(1+\varepsilon)$-approximation in $O(fn\varepsilon^{-1}\log\varepsilon^{-1}+f\varepsilon^{-3})$ time. □

Bibliography

[1] P. Afshani, M. Agrawal, B. Doerr, C. Doerr, K. G. Larsen, and K. Mehlhorn. The query complexity of finding a hidden permutation. In A. Brodnik, A. López-Ortiz, V. Raman, and A. Viola, editors, *Space-Efficient Data Structures, Streams, and Algorithms - Papers in Honor of J. Ian Munro on the Occasion of His 66th Birthday*, volume 8066 of *Lecture Notes in Computer Science*, pages 1–11. Springer, 2013.

[2] P. K. Agarwal, S. Har-Peled, and K. R. Varadarajan. Approximating extent measures of points. *J. ACM*, 51(4):606–635, 2004.

[3] P. K. Agarwal, S. Har-Peled, K. R. Varadarajan, et al. Geometric approximation via coresets. *Combinatorial and Computational Geometry*, 52(1), 2005.

[4] P. K. Agarwal, S. Har-Peled, and H. Yu. Robust shape fitting via peeling and grating coresets. *Discrete & Computational Geometry*, 39(1):38–58, 2008.

[5] G. Altekar, S. Dwarkadas, J. P. Huelsenbeck, and F. Ronquist. Parallel metropolis coupled markov chain monte carlo for bayesian phylogenetic inference. *Bioinform.*, 20(3):407–415, 2004.

[6] K. Alwan and K. Waters. Finding re-entrant knight's tours on n-by-m boards. In *Proceedings of the 30th Annual Southeast Regional Conference*, ACM-SE 30, pages 377–382, New York, NY, USA, 1992. ACM.

[7] N. Amenta, D. Attali, and O. Devillers. Size of Delaunay triangulation for points distributed over lower-dimensional polyhedra: a tight bound. *Neural Information Processing Systems (NeurIPS): Topological Learning*, 2007.

[8] N. Amenta and M. Bern. Surface reconstruction by Voronoi filtering. *Discrete & Computational Geometry*, 22(4):481–504, 1999.

[9] S. Arya, G. D. da Fonseca, and D. M. Mount. Approximate convex intersection detection with applications to width and Minkowski sums. In *26th European Symposium on Algorithms (ESA)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2018.

[10] D. Attali and J.-D. Boissonnat. Complexity of the delaunay triangulation of points on polyhedral surfaces. *Discrete & Computational Geometry*, 30(3):437–452, 2003.

[11] D. Attali and J.-D. Boissonnat. A linear bound on the complexity of the Delaunay triangulation of points on polyhedral surfaces. *Discrete & Computational Geometry*, 31(3):369–384, 2004.

[12] S. W. Bae. Computing a minimum-width square annulus in arbitrary orientation. *Theoretical Computer Science*, 718:2–13, 2018.

[13] S. W. Bae. Computing a minimum-width cubic and hypercubic shell. *Operations Research Letters*, 47(5):398–405, 2019.

[14] S. W. Bae. On the minimum-area rectangular and square annulus problem. *Computational Geometry*, 92:101697, 2021.

[15] W. R. Ball and H. Coxeter. *Mathmatical Recreations & Essays: 12th Edition*. University of Toronto Press, 1974.

[16] G. Barequet, P. Bose, M. T. Dickerson, and M. T. Goodrich. Optimizing a constrained convex polygonal annulus. *J. Discrete Algorithms*, 3(1):1–26, 2005.

[17] G. Barequet, M. T. Dickerson, and Y. Scharf. Covering points with a polygon. *Computational Geometry*, 39(3):143–162, 2008.

[18] G. Barequet, G. Rote, and M. Shalah. $\lambda¿ 4$: An improved lower bound on the growth constant of polyominoes. *Communications of the ACM*, 59(7):88–95, 2016.

[19] G. Barequet and M. Shalah. Improved upper bounds on the growth constants of polyominoes and polycubes. In *LATIN 2020: Theoretical Informatics: 14th Latin American Symposium, São Paulo, Brazil, January 5-8, 2021, Proceedings*, page 532–545, Berlin, Heidelberg, 2021. Springer-Verlag.

[20] G. Barequet, M. Shalah, and Y. Zheng. An improved lower bound on the growth constant of polyiamonds. *Journal of Combinatorial Optimization*, 37(2):424–438, 2019.

[21] R. Barequet, G. Barequet, and G. Rote. Formulae and growth rates of high-dimensional polycubes. *Combinatorica*, 30(3):257–275, 2010.

[22] J. D. Beasley. Magic knight's tours. *The College Mathematics Journal*, 43(1):72–75, 2012.

[23] E. A. Bender. Convex n-ominoes. *Discrete Mathematics*, 8(3):219–226, 1974.

[24] E. Bergholt. Three memoirs on knight's tours. *The Games and Puzzles Journal*, 2(18):327–341, 2001.

[25] A. Bernasconi, C. Damm, and I. E. Shparlinski. Circuit and decision tree complexity of some number theoretic problems. *Inf. Comput.*, 168(2):113–124, 2001.

[26] P. Bestagini, M. Tagliasacchi, and S. Tubaro. Image phylogeny tree reconstruction based on region selection. In *2016 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2016, Shanghai, China, March 20-25, 2016*, pages 2059–2063. IEEE, 2016.

[27] A. Bhattacharjee, K. Z. Sultana, and Z. Shams. Dynamic and parallel approaches to optimal evolutionary tree construction. In *Proceedings of the Canadian Conference on Electrical and Computer Engineering, CCECE 2006, May 7-10, 2006, Ottawa Congress Centre, Ottawa, Canada*, pages 119–122. IEEE, 2006.

[28] M. Bousquet-Mélou and J.-M. Fédou. The generating function of convex polyominoes: the resolution of a q-differential system. *Discrete Mathematics*, 137(1-3):53–75, 1995.

[29] M. Bousquet-Mélou, A. Guttmann, W. Orrick, and A. Rechnitzer. Inversion relations, reciprocity and polyominoes. *Annals of Combinatorics*, 3(2):223–249, 1999.

[30] S. R. Broadbent and J. M. Hammersley. Percolation processes: I. crystals and mazes. *Mathematical proceedings of the Cambridge philosophical society*, 53(3):629–641, 1957.

[31] G. S. Brodal, R. Fagerberg, C. N. S. Pedersen, and A. Östlin. The complexity of constructing evolutionary trees using experiments. In F. Orejas, P. G. Spirakis, and J. van Leeuwen, editors, *Automata, Languages and Programming, 28th International Colloquium, ICALP 2001, Crete, Greece, July 8-12, 2001, Proceedings*, volume 2076 of *Lecture Notes in Computer Science*, pages 140–151. Springer, 2001.

[32] T. M. Chan. Approximating the diameter, width, smallest enclosing cylinder, and minimum-width annulus. In *16th Symposium on Computational Geometry (SoCG)*, pages 300–309, 2000.

[33] G. Chia and S.-H. Ong. Generalized knight's tours on rectangular chessboards. *Discrete Applied Mathematics*, 150(1):80 – 98, 2005.

[34] S. Choi and J. H. Kim. Optimal query complexity bounds for finding graphs. *Artif. Intell.*, 174(9-10):551–569, 2010.

[35] B. Chor and T. Tuller. Maximum likelihood of evolutionary trees: hardness and approximation. In *Proceedings Thirteenth International Conference on Intelligent Systems for Molecular Biology 2005, Detroit, MI, USA, 25-29 June 2005*, pages 97–106, 2005.

[36] A. Conrad, T. Hindrichs, H. Morsy, and I. Wegener. Solution of the knight's hamiltonian path problem on chessboards. *Discrete Applied Mathematics*, 50(2):125 – 134, 1994.

[37] J. C. Culberson and P. Rudnicki. A fast algorithm for constructing trees from distance matrices. *Inf. Process. Lett.*, 30(4):215–220, 1989.

[38] P. Cull and J. De Curtins. Knight's tour revisited. *Fibonacci Quarterly*, 16:276–285, 6 1978.

[39] S. Das, A. Nandy, and S. Sarvottamananda. Linear time algorithm for 1-Center in $\mathfrak{R}^d$ under convex polyhedral distance function. In D. Zhu and S. Bereg, editors, *Frontiers in Algorithmics*, volume 9711, pages 41–52. Springer, 2016.

[40] I. J. Dejter. Equivalent conditions for euler's problem on z_4-hamilton cycles. *Ars Combinatoria*, 16–B:285–295, 1983.

[41] J. DeMaio. Which chessboards have a closed knight's tour within the cube? *the electronic journal of combinatorics*, 14(1):32, 2007.

[42] J. DeMaio and M. Bindia. Which chessboards have a closed knight's tour within the rectangular prism? *the electronic journal of combinatorics*, 18(1):14, 2011.

[43] Z. Dias, S. Goldenstein, and A. Rocha. Exploring heuristic and optimum branching algorithms for image phylogeny. *J. Vis. Commun. Image Represent.*, 24(7):1124–1134, 2013.

[44] Z. Dias, S. Goldenstein, and A. Rocha. Large-scale image phylogeny: Tracing image ancestral relationships. *IEEE Multim.*, 20(3):58–70, 2013.

[45] Z. Dias, A. Rocha, and S. Goldenstein. Image phylogeny by minimal spanning trees. *IEEE Trans. Information Forensics and Security*, 7(2):774–788, 2012.

[46] S. Dobzinski and J. Vondrák. From query complexity to computational complexity. In H. J. Karloff and T. Pitassi, editors, *Proceedings of the 44th Symposium on Theory of Computing Conference, STOC 2012, New York, NY, USA, May 19 - 22, 2012*, pages 1107–1116. ACM, 2012.

[47] J. Duarte and H. Ruskin. The branching of real lattice trees as dilute polymers. *Journal de Physique*, 42(12):1585–1590, 1981.

[48] E. Emamjomeh-Zadeh and D. Kempe. Adaptive hierarchical clustering using ordinal queries. In A. Czumaj, editor, *Proceedings of the Twenty-Ninth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2018, New Orleans, LA, USA, January 7-10, 2018*, pages 415–429. SIAM, 2018.

[49] J. Erde, B. Golénia, and S. Golénia. The closed knight tour problem in higher dimensions. *the electronic journal of combinatorics*, 19(4):9, 2012.

[50] J. Erickson. Nice point sets can have nasty Delaunay triangulations. In *17th Symposium on Computational Geometry (SoCG)*, pages 96–105, 2001.

[51] J. S. Farris. Methods for Computing Wagner Trees. *Systematic Biology*, 19(1):83–92, 03 1970.

[52] J. Felsenstein. Evolutionary trees from dna sequences: a maximum likelihood approach. *Journal of molecular evolution*, 17(6):368–376, 1981.

[53] A. Fischer. New records in nonintersecting knight paths. *The Games and Puzzles Journal*, 2006.

[54] W. M. Fitch. Toward defining the course of evolution: Minimum change for a specific tree topology. *Systematic Zoology*, 20(4):406–416, 1971.

[55] P. Flajolet. *Pólya festoons*. PhD thesis, INRIA, 1991.

[56] D. Gaunt. The critical dimension for lattice animals. *Journal of Physics A: Mathematical and General*, 13(4):L97, 1980.

[57] D. Gaunt, M. Sykes, and H. Ruskin. Percolation processes in d-dimensions. *Journal of Physics A: Mathematical and General*, 9(11):1899, 1976.

[58] O. N. Gluchshenko, H. W. Hamacher, and A. Tamir. An optimal $O(n \log n)$ algorithm for finding an enclosing planar rectilinear annulus of minimum width. *Operations Research Letters*, 37(3):168–170, 2009.

[59] L. A. Goldberg, P. W. Goldberg, C. A. Phillips, and G. B. Sorkin. Constructing computer virus phylogenies. *J. Algorithms*, 26(1):188–208, 1998.

[60] M. T. Goodrich and R. Tamassia. *Algorithm Design and Applications*. Wiley, New York, NY, 2011.

[61] J. J. Hein. An optimal algorithm to reconstruct trees from additive distance data. *Bulletin of mathematical biology*, 51(5):597–603, 1989.

[62] I. Herman, G. Melançon, and M. S. Marshall. Graph visualization and navigation in information visualization: A survey. *IEEE Transactions on visualization and computer graphics*, 6(1):24–43, 2000.

[63] J. P. Huelsenbeck. Performance of phylogenetic methods in simulation. *Systematic biology*, 44(1):17–48, 1995.

[64] M. Jagadish and A. Sen. Learning a bounded-degree tree using separator queries. In S. Jain, R. Munos, F. Stephan, and T. Zeugmann, editors, *Algorithmic Learning Theory - 24th International Conference, ALT 2013, Singapore, October 6-9, 2013. Proceedings*, volume 8139 of *Lecture Notes in Computer Science*, pages 188–202. Springer, 2013.

[65] G. P. Jelliss. Non-intersecting paths by leapers. *The Games and Puzzles Journal*, 2(17):305 – 310, 1999.

[66] G. P. Jelliss. Symmetry in knight's tours. *The Games and Puzzles Journal*, 2(16):282 – 287, 1999.

[67] I. Jensen. Enumerations of lattice animals and trees. *Journal of statistical physics*, 102(3):865–881, 2001.

[68] I. Jensen. Counting polyominoes: A parallel implementation for cluster computing. In *International Conference on Computational Science*, pages 203–212. Springer, 2003.

[69] I. Jensen and A. J. Guttmann. Statistics of lattice animals (polyominoes) and polygons. *Journal of Physics A: Mathematical and General*, 33(29):L257, 2000.

[70] J.-H. Ji, S.-H. Park, G. Woo, and H.-G. Cho. Generating pylogenetic tree of homogeneous source code in a plagiarism detection system. *International Journal of Control, Automation, and Systems*, 6(6):809–817, 2008.

[71] N. C. Jones, P. A. Pevzner, and P. Pevzner. *An introduction to bioinformatics algorithms*. MIT press, 2004.

[72] N. Kamčev. Generalised knight's tours. *the electronic journal of combinatorics*, 21(1):32, 2011.

[73] S. Kannan, E. L. Lawler, and T. J. Warnow. Determining the evolutionary tree using experiments. *J. Algorithms*, 21(1):26–50, 1996.

[74] S. Kannan, C. Mathieu, and H. Zhou. Graph reconstruction and verification. *ACM Trans. Algorithms*, 14(4):40:1–40:30, 2018.

[75] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete mathematics*, 23(3):309–311, 1978.

[76] D. G. Kendall. Shape manifolds, procrustean metrics, and complex projective spaces. *Bulletin of the London mathematical society*, 16(2):81–121, 1984.

[77] V. King, L. Zhang, and Y. Zhou. On the complexity of distance-based evolutionary tree reconstruction. In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms, January 12-14, 2003, Baltimore, Maryland, USA*, pages 444–453. ACM/SIAM, 2003.

[78] D. A. Klarner. Cell growth problems. *Canadian Journal of Mathematics*, 19:851–863, 1967.

[79] D. A. Klarner and R. L. Rivest. A procedure for improving the upper bound for the number of n-ominoes. *Canadian Journal of Mathematics*, 25(3):585–602, 1973.

[80] D. A. Klarner and R. L. Rivest. Asymptotic bounds for the number of convex n-ominoes. *Discret. Math.*, 8(1):31–40, 1974.

[81] D. E. Knuth. Leaper graphs. *The Mathematical Gazette*, 78(483):274–297, 1994.

[82] A. Kumar. Non-crossing Knight's Tour in 3-Dimension. *ArXiv e-prints*, Mar. 2008.

[83] O. Kyek, I. Parberry, and I. Wegener. Bounds on the number of knight's tours. *Discrete Applied Mathematics*, 74(2):171 – 181, 1997.

[84] S.-S. Lin and C.-L. Wei. Optimal algorithms for constructing knight's tours on arbitrary $n \times m$ chessboards. *Discrete Applied Mathematics*, 146(3):219 – 232, 2005.

[85] N. Madras. A pattern theorem for lattice clusters. *Annals of Combinatorics*, 3(2):357–384, 1999.

[86] S. R. Mahaney. Sparse complete sets for np: Solution of a conjecture of berman and hartmanis. *Journal of Computer and System Sciences*, 25(2):130 – 143, 1982.

[87] G. D. Marmerola, M. A. Oikawa, Z. Dias, S. Goldenstein, and A. Rocha. On the reconstruction of text phylogeny trees: evaluation and analysis of textual relationships. *PloS one*, 11(12):e0167822, 2016.

[88] B. D. McKay. Knight's tours of an 8× 8 chessboard. Technical report, Australian National University, Department of Computer Science, 2 1997.

[89] M. Mitzenmacher and E. Upfal. *Probability and Computing: Randomized Algorithms and Probabilistic Analysis*. Cambridge University Press, 2005.

[90] J. Mukherjee, P. R. Sinha Mahapatra, A. Karmakar, and S. Das. Minimum-width rectangular annulus. *Theoretical Computer Science*, 508:74–80, 2013.

[91] C. Nash-Williams. Abelian groups, graphs and generalized knights. In *Mathematical Proceedings of the Cambridge Philosophical Society*, volume 55, pages 232–238. Cambridge University Press, 1959.

[92] M. Pagel. Inferring the historical patterns of biological evolution. *Nature*, 401(6756):877–884, 1999.

[93] I. Parberry. Scalability of a neural network for the knight's tour problem. *Neurocomputing*, 12(1):19 – 33, 1996.

[94] I. Parberry. An efficient algorithm for the knight's tour problem. *Discrete Applied Mathematics*, 73(3):251–260, 1997.

[95] A. Pfeffer, C. Call, J. Chamberlain, L. Kellogg, J. Ouellette, T. Patten, G. Zacharias, A. Lakhotia, S. Golconda, J. Bay, R. Hall, and D. Scofield. Malware analysis and attribution using genetic information. In *7th International Conference on Malicious and Unwanted Software, MALWARE 2012, Fajardo, PR, USA, October 16-18, 2012*, pages 39–45. IEEE Computer Society, 2012.

[96] J. M. Phillips. Coresets and sketches. In *Handbook of Discrete and Computational Geometry*, pages 1269–1288. Chapman and Hall/CRC, 2017.

[97] W. Piel, L. Chan, M. Dominus, J. Ruan, R. Vos, and V. Tannen. Treebase v. 2: A database of phylogenetic knowledge. e-biosphere, 2009.

[98] I. Pohl. A method for finding hamilton paths and knight's tours. *Commun. ACM*, 10(7):446–449, July 1967.

[99] G. Pólya and G. Szegő. *Aufgaben und Lehrsätze aus der Analysis*, volume 1. J. Springer, 1925.

[100] H. Prüfer. Neuer beweis eines satzes über permutationen. *Arch. Math. Phys*, 27(1918):742–744, 1918.

[101] Y. Qing and J. J. Watkins. Knight's tours for cubes and boxes. *Congressus Numerantium*, 01 2006.

[102] L. Reyzin and N. Srivastava. On the longest path algorithm for reconstructing trees from distance matrices. *Inf. Process. Lett.*, 101(3):98–100, 2007.

[103] F. J. Rohlf. J. felsenstein, inferring phylogenies, sinauer assoc., 2004, pp. xx + 664. *J. Classif.*, 22(1):139–142, 2005.

[104] A. J. Schwenk. Which rectangular chessboards have a knight's tour? *Mathematics Magazine*, 64(5):325–332, 1991.

[105] B. Shen, C. W. Forstall, A. de Rezende Rocha, and W. J. Scheirer. Practical text phylogeny for real-world settings. *IEEE Access*, 6:41002–41012, 2018.

[106] J. A. Shufelt and H. J. Berliner. Generating knight's tours without backtracking from errors. Technical report, Carnegie-Mellon University, School of Computer Science, 1993.

[107] N. J. Sloane et al. The on-line encyclopedia of integer sequences. *Published electronically at https://oeis. org*, 2018.

[108] D. Squirrel and P. Cull. A warnsdorff-rule algorithm for knight's tours on square chessboards. 1996.

[109] G. Tardos. Query complexity, or why is it difficult to seperate NP ^{a} cap co np^{a} from p^{a} by random oracles a? *Combinatorica*, 9(4):385–392, 1989.

[110] H. N. V. Temperley. Combinatorial problems suggested by the statistical mechanics of domains and of rubber-like molecules. *Physical Review*, 103(1):1, 1956.

[111] U. Thakker, R. Patel, S. Tanwar, N. Kumar, and H. Song. Blockchain for diamond industry: Opportunities and challenges. *IEEE Internet of Things Journal*, pages 1–1, 2020.

[112] W. Tutte. *Graph Theory*. Cambridge Mathematical Library. Cambridge University Press, 2001.

[113] L. G. Valiant. Universality considerations in VLSI circuits. *IEEE Trans. Computers*, 30(2):135–140, 1981.

[114] Z. Wang and J. Honorio. Reconstructing a bounded-degree directed tree using path queries. In *57th Annual Allerton Conference on Communication, Control, and Computing, Allerton 2019, Monticello, IL, USA, September 24-27, 2019*, pages 506–513. IEEE, 2019.

[115] M. S. Waterman, T. F. Smith, M. Singh, and W. A. Beyer. Additive evolutionary trees. *Journal of theoretical Biology*, 64(2):199–213, 1977.

[116] J. J. Watkins. *Across the Board: The Mathematics of Chessboard Problems*. Princeton Puzzlers. Princeton University Press; Reissue edition, 2012.

[117] J. J. Watkins and R. L. Hoenigman. Knight's tours on a torus. *Mathematics Magazine*, 70(3):175–184, 1997.

[118] A. C. Yao. Decision tree complexity and betti numbers. In F. T. Leighton and M. T. Goodrich, editors, *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing, 23-25 May 1994, Montréal, Québec, Canada*, pages 615–624. ACM, 1994.

[119] L. D. Yarbrough. Uncrossed knight's tours. *Journal of Recreational Mathematics*, 1(3):140–142, 1969.

[120] H. Yu, P. K. Agarwal, R. Poreddy, and K. R. Varadarajan. Practical methods for shape fitting and kinetic data structures using coresets. *Algorithmica*, 52(3):378–402, 2008.