2013

Contributions to computational phylogenetics and algorithmic self-assembly

Brad Shutters
Iowa State University

Contributions to computational phylogenetics and algorithmic self-assembly

by

Brad Shutters

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Computer Science

Program of Study Committee:

David Fernández-Baca, Major Professor

Pavan Aduri

Clifford Bergman

Jack H. Lutz

Giora Slutzki

Iowa State University

Ames, Iowa

2013

DEDICATION

I dedicate this dissertation to my mother, Betty Ann Shutters, whose loving support and encouragement made it possible for me to complete this work.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

Though only my name appears on the cover of this dissertation, a great many people have contributed to its production. I owe my gratitude to all those people who have made this dissertation possible.

My deepest gratitude is to my dissertation advisor, David Fernández-Baca. I have been fortunate to have an advisor who gave me the freedom to explore the topics of interest to me, provided guidance when my steps faltered, and was always patient with my progress. David has been a great example of a first class scientist.

I thank Jack Lutz for getting me started in research and providing me with guidance in the initial stages of my doctoral studies. I am grateful to Steve Kautz, Sylvain Guillemot, and Sudheer Vakati for collaborating with me on the research projects that lead to many of the results presented in this dissertation, and Jim Lathrop for encouraging my research in self-assembly. I also thank Pavan Aduri, Clifford Bergman, and Giora Slutzki for giving their time to serve on my committee. Additionally, there are several other members of the research community who have inspired me with stimulating conversations, and provided valuable feedback on my work.

My biggest thanks are to my mother, Betty Shutters, for her constant encouragement and loving support, both mentally and financially. None of this would have been possible without her help, and I deeply appreciate her belief in me. I am also thankful to my father, Bill Shutters, for the many valuable math lessons over the years.

A few friends have helped me stay sane through these difficult years. In particular, Brad Bossard and Steve Tangeman have given me much needed inspiration and encouragement at various times throughout my studies. I greatly value their friendship.

Finally, I thank the Computer Science Department at the University of Wisconsin-La Crosse for giving me the opportunity to succeed at the next level. This has given me much needed motivation in my final months as a graduate student.

ABSTRACT

This dissertation addresses some of the algorithmic and combinatorial problems at the interface between biology and computation. In particular, it focuses on problems in both computational phylogenetics, an area of study in which computation is used to better understand evolutionary relationships, and algorithmic self-assembly, an area of study in which biological processes are used to perform computation.

The first set of results investigate inferring phylogenetic trees from multi-state character data. We give a novel characterization of when a set of three-state characters has a perfect phylogeny and make progress on a long-standing conjecture regarding the compatibility of multi-state characters.

The next set of results investigate inferring phylogenetic supertrees from collections of smaller input trees when the input trees do not fully agree on the relative positions of the taxa. Two approaches to dealing with such conflicting input trees are considered. The first is to contract a set of edges in the input trees so that the resulting trees have an agreement supertree. The second is to remove a set of taxa from the input trees so that the resulting trees have an agreement supertree. We give fixed-parameter tractable algorithms for both approaches.

We then turn to the algorithmic self-assembly of fractal structures from DNA tiles and investigate approximating the Sierpinski triangle and the Sierpinski carpet with strict self-assembly. We prove tight bounds on approximating the Sierpinski triangle and exhibit a class of fractals that are generalizations of the Sierpinski carpet that can approximately self-assemble.

We conclude by discussing some ideas for further research.

CHAPTER 1. General Introduction

This dissertation addresses some of the algorithmic and combinatorial problems at the interface between biology and computation. In particular, it focuses on problems in both computational phylogenetics, an area of study in which computation is used to better understand evolutionary relationships, and algorithmic self-assembly, an area of study in which biological processes are used to perform computation. General introductions to the problems addressed in each area are given separately below.

1.1 Computational Phylogenetics

Darwin used a metaphor he termed the *tree of life* to describe the evolutionary relatedness among species. Figure 1.1 shows a sketch from his seminal work *The Origin of Species*. For a set of species (or other *taxa*) a *phylogenetic tree* is constructed from the similarities and differences in the *characters* of the taxa, i.e., the observed physical or genetic characteristics. Evolutionary biologists use phylogenetic trees to depict evolution because they convey the

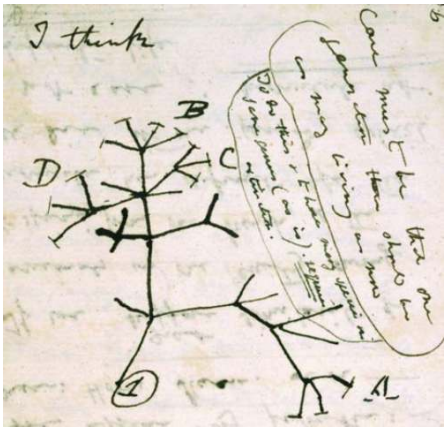


Figure 1.1: Darwin's first diagram of an evolutionary tree (1837) [5].

concept that speciation occurs through the splitting of lineages. This dissertation focuses on two fundamental problems in constructing phylogenetic trees by computational methods: the perfect phylogeny problem (Chapter 2) and the tree compatibility problem (Chapter 3).

1.1.1 The Perfect Phylogeny Problem

Formally, a character χ is a partition of a taxon set L. The parts of χ are called *states*. If χ has k parts, then χ is a k-state character. Given a phylogenetic tree T whose leaves are labeled by L, and a state s of χ, we denote by $T_s(\chi)$ the minimal subtree of T connecting all leaves labeled by taxa having state s for χ. We say that T displays χ if the subtrees $T_i(\chi)$ and $T_j(\chi)$ are vertex disjoint for all states i and j of χ where $i \neq j$. The *perfect phylogeny problem* is to determine if there is a phylogenetic tree for a given set of taxa that displays a given collection of characters on those taxa; in which case those characters are said to be *compatible*.

Since 1971, it has been known that a set of 2-state characters is compatible if and only if it is pairwise compatible, i.e., every pair of characters is compatible. However, it wasn't until 2009 that it was shown that a set of 3-state characters is compatible if and only if every triple of characters is compatible when Lam, Gusfield, and Sridhar [50] characterized the sets of compatible 3-state characters (that are also pairwise compatible) by the existence of one of a set of four forbidden subgraphs in the intersection graph of the characters. This result implies an $O(m^3n)$ time algorithm for finding all minimal subsets of a set of m 3-state characters over n taxa that are incompatible. In Chapter 2, we give an $O(m^2n + p)$ time algorithm for finding all p minimal subsets of incompatible characters, and we completely characterize the sets of compatible 3-state characters (that are also pairwise compatible) by the existence of a single forbidden subgraph in the intersection graph of the characters.

We then turn to the *character compatibility conjecture* which is a long standing conjecture in computational phylogenetics on the necessary and sufficient conditions for the compatibility of a set of k-state characters: *There exists a function $f(k)$ such that, for any set C of k-state characters, C is compatible if and only if every subset of $f(k)$ characters of C is compatible.* If the conjecture is true, it has several algorithmic consequences. For example, we could find a subset of characters to remove from C so that the remaining characters are compatible by

reducing to $f(k)$-hitting set which is fixed-parameter tractable [22]. From the discussion above we have that $f(2) = 2$ and $f(3) = 3$. In 1983, Meacham demonstrated a lower bound of $f(k) \geq k$ for all k [57], and it was long conjectured that $f(k) = k$ for all k. However, in Chapter 2, we show a quadratic lower bound on $f(k)$, i.e., $f(k) \in \Omega(k^2)$.

Our lower bound on $f(k)$ is proven using quartets. Given a phylogenetic tree, a *quartet* is a restriction of that tree to four of its leaves. A collection of quartets is compatible if they are the subset of the quartets of some tree. The previous lower bound on the maximum cardinality of a minimal set of incompatible quartets was $\Omega(n)$ where n is the number of taxa. In Chapter 2, we give an $\Omega(n^2)$ lower bound. We also show that such a set of quartets can have size at most 3 when n = 5, and at most $O(n^3)$ for arbitrary n.

1.1.2 The Tree Compatibility Problem

Due to the biological and computational constraints on constructing large scale phylogenetic trees from multi-state character data, techniques to combine collections of smaller *input trees* on overlapping sets of taxa into a *supertree* for all of the taxa are desired. Determining if such a tree exists for a given collection of input trees is called the *tree compatibility problem*. It is often impossible to construct such a supertree from a collection of smaller trees on overlapping subsets of the taxa because of conflicts with the relative positions of some of the taxa occurring in multiple input trees. Such conflicts arise due to errors in the inference process, or due to biological processes. In Chapter 3, we develop two fixed-parameter algorithms for dealing with this problem when each of the input trees is required to be homeomorphic to the constructed supertree. The first is an algorithm for finding a minimal subset of taxa to remove from the input trees so that a supertree can be constructed (the *taxon removal problem*). The second is an algorithm for finding a minimal subset of the edges of the input trees to contract so that a supertree can be constructed (the *edge contraction problem*).

1.2 Algorithmic Self-Assembly

Self-assembly occurs when simple objects autonomously combine to form complex structures as a consequence of specific, local interactions among the objects themselves without external

direction. It is a fundamental process in nature that generates structural organization on all scales; from water molecules self-assembling into snowflakes to the formation of galaxies. Since the pioneering work of Seeman [69], the self-assembly of DNA molecules has developed into a field with rich interactions between the theory of computing (the information processing properties of DNA) and geometry (the structural properties of DNA), and with many applications to nanotechnology [70].

The primary model used for studying the self-assembly of DNA tiles is the Tile Assembly Model [84]. In this model, simple two-dimensional square *tiles* are designed so that they self-assemble into a desired pattern or shape. The tiles represent DNA double crossover molecules and the single-strand nucleotide sequences making up the four arms of the molecule are known as *sticky ends*. To design a system in which a given structure self-assembles, the challenge is to program the sticky ends so that the tiles stick together forming the target structure. We give an overview of the Tile Assembly Model sufficient for understanding the results presented in this dissertation in Chapter 4.

A major objective of nanotechnology is to engineer the structures found in nature, which are often fractals. In fact, Carbone and Seeman, the father of DNA nanotechnology, stated that "generating fractal structures by self-assembly is a major challenge for nanotechnology" [16]. Figure 1.2 shows atomic force microscopy (AFM) images of several Sierpinski triangles that have self-assembled from DNA tiles [66]. However, what has self-assembled in this case is not the structure of the Sierpinski triangle, but rather a surface on which the pattern of the

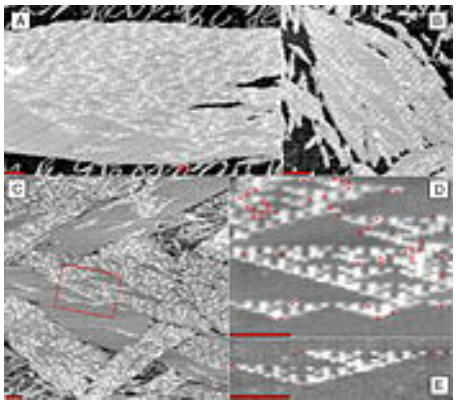


Figure 1.2: AFM image of Sierpinski triangles self-assembled from DNA tiles [66].

Sierpinski triangle can be observed. This is called *weak self-assembly*. In this dissertation we are interested in self-assembling the fractal structure itself, and nothing else. This is known as *strict self-assembly*. It is an open question whether any fractal structure self-assembles in the strict sense, and negative results exist for the Sierpinski triangle [51].

This has motivated the development of techniques to approximate fractal structures with strict self-assembly. Previous techniques for the self-assembly of fractal structures [51, 62] introduced communication fibers that shift the successive stages of the fractal. Although this technique results in a structure with the same fractal dimension as the intended fractal, the fractal pattern cannot be observed in specially labeled tiles. In fact, the resulting structure does not even contain the intended fractal structure, it only visually resembles the intended structure.

Ideally, an approximation of a fractal F would be an *in-place* approximation, i.e., a set $X \supset F$ with the same fractal dimension as F that strictly self-assembles in such a way that those tiles corresponding to F are specially labeled. In other words, communication fibers are allowed to be added to the structure so long as they do not distort the geometry of the target structure, and the additional space used by the communication fibers is no greater than the space used by the target structure itself. If a target structure can be approximated with self-assembly in this manner, then we say that it *approximately self-assembles*. Figure 1.3 shows an approximation of the Sierpinski carpet in which the yellow area represents negative space.

In this dissertation, we focus on the approximate self-assembly of two important fractals in the Tile Assembly Model: the Sierpinski triangle (Chapter 5) and the Sierpinski carpet

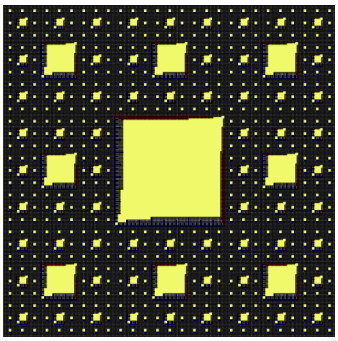


Figure 1.3: Approximation of the Sierpinski carpet in the Tile Assembly Model.

(Chapter 6). For the Sierpinski triangle, we exhibit a tile assembly system in which the target fractal approximately self-assembles, and give tight upper and lower bounds on how closely the Sierpinski triangle can be approximated with self-assembly. We also develop a new method, conditional determinism, to determine if the output of a self-assembly is unique. For the Sierpinski carpet, we show that an infinite class of fractals generalizing the Sierpinski carpet can approximately self-assemble. This self-assembly employs the use of new mechanisms for measuring distances in a growing assembly, namely *rulers* and *readers*, which have many geometric advantages over conventional binary counters, and should be useful in several other self-assembly systems. Additionally, we develop the concept of *embedded fractals* and prove some interesting properties about their dimensions that should be a useful tool for analyzing the dimension of other fractal structure approximations. We conclude by giving strong evidence for a conjecture on the limitations of approximating fractal structures in the Tile Assembly Model.

1.3 Organization of this Dissertation

Chapters 2 and 3 focus on computational phylogenetics. In Chapter 2 we explore the multi-state perfect phylogeny problem. The results given in Chapter 2 are taken from three papers [72, 73, 74]. Chapter 3 is a reproduction of [28] and explores approaches to the tree compatibility problem when the input trees have conflicts in the relative positions of some of the taxa. Chapters 4, 5 and 6 focus on algorithmic self-assembly. In Chapter 4 we give a formal definition of the Tile Assembly Model and a brief introduction to discrete self-similar fractals. In Chapter 5 we study approximating the Sierpinski triangle in the Tile Assembly Model. The results given are taken from [53]. In Chapter 6 we study approximating generalized Sierpinski carpets in the Tile Assembly model. The results given are taken from [48]. Concluding remarks appear in Chapter 7.

1.3.1 Author's Contributions

Chapter 2: All results given in this chapter are the author's own contributions to three manuscripts [72, 73, 74]. The author also wrote all portions of those manuscripts which are reproduced here. *Chapter 3*: The author contributed to the design of all algorithms and struc-

tural results given in the manuscript reproduced for this chapter [28] with the exception of Theorem 22 which was the work of Sylvain Guillemot. The author also made major contributions to the writing of the manuscript. *Chapter 5*: The results in this chapter are from [53]. All results given in this chapter are the author's own contributions with the exception of Theorem 28 which was suggested, along with the idea used in its proof, by Jack Lutz. *Chapter 6*: The results in this chapter are from [48] which was based on a preliminary work of the author in which the self-assembling rulers and readers were developed and used to create a construction in which the Sierpinski carpet approximately self-assembles. The author worked jointly with Steve Kautz to modify that construction to generalized Sierpinski carpets. The construction was primarily the work of the author, but the construction's proof of correctness is primarily the work of Steve Kautz.

CHAPTER 2. Obstructions to Perfect Phylogenies

The work in this chapter is based on the author's contribution to the three manuscripts [72, 73, 74].

Abstract

We study a long standing conjecture on the necessary and sufficient conditions for the compatibility of multi-state characters: There exists a function $f(k)$ such that, for any set C of k-state characters, C is compatible if and only if every subset of $f(k)$ characters of C is compatible. We show that for every $k \geq 2$, there exists an incompatible set C of $\Omega(k^2)$ k-state characters such that every proper subset of C is compatible. This improves the previous lower bound of $f(k) \geq k$ given by Meacham (1983), and $f(4) \geq 5$ given by Habib and To (2011).

For the case when $k = 3$, Lam, Gusfield and Sridhar (2011) recently showed that $f(3) = 3$. We give an independent proof of this result and completely characterize the sets of pairwise compatible 3-state characters by a single forbidden intersection pattern. We then use this result to give an $O(m^2n + p)$ time algorithm to output all p minimal subsets of incompatible characters for a set of m 3-state characters over a set of n taxa.

Our lower bound on $f(k)$ is proven via a result on quartet compatibility that may be of independent interest: For every $n \geq 4$, there exists an incompatible set Q of $\Omega(n^2)$ quartets over n labels such that every proper subset of Q is compatible. This improves the previous lower bound of $n - 2$ given by Steel (1992). We also show that such a set of quartets can have size at most 3 when $n = 5$, and at most $O(n^3)$ for arbitrary n.

2.1 Introduction

The perfect phylogeny problem is a basic question in computational phylogenetics [71]. The input is an n by m matrix $\mathcal{M}$ of integers from the set $K = \{1, \ldots, k\}$. We refer to each row of $\mathcal{M}$ as a *taxon* (plural *taxa*), each column of $\mathcal{M}$ as a *character*, and each value that occurs in a column α of $\mathcal{M}$ as a *state* of character α. We will often write C to represent the set of all characters in $\mathcal{M}$. A *perfect phylogeny* for $\mathcal{M}$ is a tree T with n leaves such that each leaf is labeled by a distinct taxon of $\mathcal{M}$, each internal node is labeled by a vector in K^m, and, for every character α of $\mathcal{M}$, and every state i of α, the nodes labeled with state i for character α form a connected subtree of T. The perfect phylogeny problem is to decide if there exists a perfect phylogeny for $\mathcal{M}$; if so, then we say that the characters of $\mathcal{M}$ are compatible, and incompatible otherwise. See [27, 71] for more on the perfect phylogeny problem. See Figure 2.1 for an example.

If the number of states of each character is unbounded (so k can grow with n), then the perfect phylogeny problem is NP-complete [8, 77]. However, if the number of states of each character is fixed, then the perfect phylogeny problem is solvable in $O(m^2n)$ [23, 43, 2, 44], and $O(mn)$ time when $k = 2$ [35].

It is straightforward to show that every subset of a compatible set of characters is itself compatible. It follows that if no perfect phylogeny exists for $\mathcal{M}$, there must be some minimal subset of the characters of $\mathcal{M}$ that does not have a perfect phylogeny. We call such a set a *minimal obstruction set* for $\mathcal{M}$. Although the above mentioned algorithms can also construct a perfect phylogeny for $\mathcal{M}$ if one exists, they do not output a minimal obstruction set when

$\mathcal{M}$	α β γ
1	3 2 1
2	2 1 3
3	2 2 2
4	3 3 2
5	1 1 3
6	2 2 3

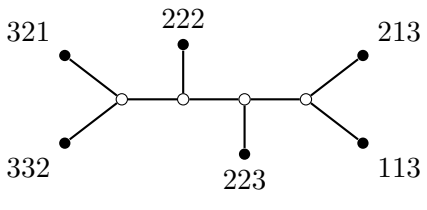


Figure 2.1: Example perfect phylogeny for input matrix $\mathcal{M}$.

there is no perfect phylogeny for $\mathcal{M}$. The focus of this chapter is on these minimal obstruction sets.

It has long been known that when $\mathcal{M}$ consists only of 2-state characters, $\mathcal{M}$ is compatible if and only if every pair of characters in $\mathcal{M}$ is compatible; see [12, 24, 35, 57, 71]. However, when $k > 2$, pairwise character compatibility is no longer sufficient to guarantee the compatibility of $\mathcal{M}$. This was first demonstrated in 1971 by Fitch [29, 30] who exhibited a set of three pairwise compatible 3-state characters that is incompatible. In 1983, Meacham [57] generalized this example by exhibiting, for every $k \geq 3$, an incompatible set of k-state characters of cardinality k such that every proper subset is compatible; see also [50]. This result lead to the following conjecture.

Conjecture 1. *There exists a function $f(k)$ such that, for any set C of k-state characters, C is compatible if and only if every subset of $f(k)$ characters of C is compatible.*

If Conjecture 1 is true, it would follow that we can determine if a set of k-state characters is compatible by testing the compatibility of each subset of $f(k)$ characters, and, in case of incompatibility, output a subset of at most $f(k)$ characters that is incompatible. This would allow us to reduce the character removal problem (i.e., finding a subset of characters to remove from C so that the remaining characters are compatible) to $f(k)$-hitting set which is fixed-parameter tractable [59]. From the discussion above, it follows that $f(2) = 2$ and $f(k) \geq k$ for all $k > 2$.

In this chapter, we start with the 3-state perfect phylogeny problem and build upon the work of Lam, Gusfield, and Sridhar [50] who showed that the maximum cardinality of a minimal obstruction set for a set of 3-state characters has cardinality at most three; establishing that $f(3) = 3$ and implying an $O(m^3n)$ time algorithm to output all minimal obstruction sets for a set of m 3-state characters on n taxa. They also gave a complete characterization of the minimal obstruction sets of pairwise compatible sets of 3-state characters by the existence of one of four forbidden subgraphs in the partition intersection graph of the characters. Here, we give an independent proof that the maximum cardinality of such an obstruction set is three, and completely characterize the minimal obstruction sets of pairwise compatible 3-state

characters by the existence of a single forbidden subgraph in the partition intersection graph of the characters. We then give a $O(m^2n+p)$ time algorithm to output all p minimal obstruction sets. Although there can be $O(m^3)$ minimal obstruction sets for a set of 3-state characters in the worst case, in practice we expect the number of minimal obstruction sets to be small; in which case the algorithm given here is a considerable improvement.

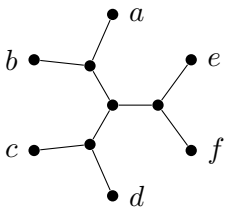
While the above results could lead one to conjecture that $f(k) = k$ for all k, Habib and To [39] recently disproved this possibility by exhibiting a set of five 4-state characters that is incompatible, but every proper subset is compatible, showing that $f(4) \geq 5$. They conjectured that $f(k) \geq k+1$ for every $k \geq 4$. We prove their conjecture by giving a quadratic lower bound on $f(k)$. Formally, we show that for every $k \geq 2$, there exists a set of $\lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil + 1$ incompatible k-state characters such that every proper subset is compatible. Therefore, $f(k) \geq \lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil + 1$ for every $k \geq 2$. Our proof relies on a new result on quartet compatibility we believe is of independent interest. We show that for every $n \geq 4$, there exists an incompatible set of $\lfloor \frac{n-2}{2} \rfloor \cdot \lceil \frac{n-2}{2} \rceil + 1$ quartets over a set of n labels such that every proper subset is compatible. This is an improvement over the previous lower bound on the maximum cardinality of such an incompatible set of quartets of $n-2$ given in [77]. Additionally, we show that such a set of quartets can have cardinality at most 3 when $n = 5$, and at most $O(n^3)$ for arbitrary n.

2.2 Preliminaries and Previous Work

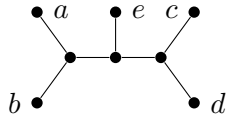
Given a graph G, we represent the vertices and edges of G by $V(G)$ and $E(G)$ respectively. We use the abbreviated notation uv for an edge $\{u, v\} \in E(G)$. For any $e \in E(G)$, $G - e$ represents the graph obtained from G by deleting edge e. For an integer i, we use $[i]$ to represent the set $\{1, 2, \cdots, i\}$.

2.2.1 Unrooted Phylogenetic Trees and Quartets

An *unrooted phylogenetic tree* (or just *tree*) is a tree T whose leaves are in one to one correspondence with a label set $\mathcal{L}(T)$, and has no vertex of degree two. See Fig. 2.2(a) for an example. For a collection $\mathcal{T}$ of trees, the *label set* of $\mathcal{T}$, denoted $\mathcal{L}(\mathcal{T})$, is the union of the label sets of the trees in $\mathcal{T}$. A tree is *binary* if every internal (non-leaf) vertex has degree three. A



(a) A tree T witnessing that the quartets $q_1 = ab|ce$, $q_2 = cd|bf$, and $q_3 = ad|ef$ are compatible; T is also a witness that the characters $\chi_{q_1} = ab|ce|d|f$, $\chi_{q_2} = cd|bf|a|e$, and $\chi_{q_3} = ad|ef|b|c$ are compatible.



(b) T restricted to $\{a, b, c, d, e\}$.

Figure 2.2: A phylogenetic tree and a restricted subtree.

quartet is a binary tree with exactly four leaves. A quartet with label set $\{a, b, c, d\}$ is denoted $ab|cd$ if the path between the leaves labeled a and b does not intersect with the path between the leaves labeled c and d.

For a tree T, and a label set $L \subseteq \mathcal{L}(T)$, the *restriction* of T to L, denoted by $T|L$, is the tree obtained from the minimal subtree of T connecting all the leaves with labels in L by suppressing vertices of degree two. See Fig. 2.2(b) for an example. A tree T *displays* another tree T', if T' can be obtained from $T|\mathcal{L}(T')$ by contracting edges. A tree T displays a collection of trees $\mathcal{T}$ if T displays every tree in $\mathcal{T}$. If such a tree T exists, then we say that $\mathcal{T}$ is *compatible*; otherwise, we say that $\mathcal{T}$ is *incompatible*. See Fig. 2.2(a) for an example. Determining if a collection of unrooted trees is compatible is NP-complete [77].

2.2.1.1 Quartet Rules

We now introduce *quartet (closure) rules* which were originally used in the contexts of psychology [18] and linguistics [19]. The idea is that for a collection Q of quartets, any tree that displays Q may also necessarily display another quartet $q \notin Q$, and if so we write $Q \vdash q$.

Example 1. *Let $Q = \{ab|ce, ae|cd\}$. Then the tree of Fig. 2.2(b) displays Q, and furthermore, it is easy to see that it is the only tree that displays Q. Hence, $Q \vdash ab|de$, $Q \vdash ab|cd$, and $Q \vdash be|cd$.*

We will make use of the following quartet rules.

$$\{ab|cd, ab|ce\} \vdash ab|de \tag{R1}$$

$$\{ab|cd, ac|de\} \vdash ab|ce \tag{R2}$$

For our purposes, we define the *closure* of an arbitrary collection Q of quartets, denoted Q^*, as the minimal set of quartets that contains Q, and has the property that if for some $q_1, q_2 \in Q^*$, $\{q_1, q_2\} \vdash q_3$ using either (R1) or (R2), then $q_3 \in Q^*$. Clearly, any tree that displays Q must also display Q^*. We will use the following lemma which follows by repeated application of (R1) and is formally proven in [20].

Lemma 1. *Let Q be an arbitrary set of quartets with $\{x, y, z_1, \ldots, z_k\} \subseteq \mathcal{L}(Q)$. If*

$$\bigcup_{i=1}^{k-1} \{xy|z_i z_{i+1}\} \subseteq Q^* \ ,$$

then $xy|z_1 z_k \in Q^*$.

We refer the reader to [71] and [33] for more on quartet rules.

2.2.2 Multi-State Characters

There is also a notion of compatibility for sets of partitions of a label set L. A *character* χ on L is a partition of L; the parts of χ are called *states*. If χ has at most k parts, then χ is a k-state character. Given a tree T with $L = \mathcal{L}(T)$ and a state s of χ, we denote by $T_s(\chi)$ the minimal subtree of T connecting all leaves with labels having state s for χ. We say that χ is *convex* on T, or equivalently T *displays* χ, if the subtrees $T_i(\chi)$ and $T_j(\chi)$ are vertex disjoint for all states i and j of χ where $i \neq j$. A collection C of characters is *compatible* if there exists a tree T on which every character in C is convex, i.e., there is a perfect phylogeny for C. If no such tree exists, then we say that C is *incompatible*. See Fig. 2.2(a) for an example. The *perfect phylogeny problem* (or *character compatibility problem*) is to determine whether a given set of characters is compatible.

For an input matrix $\mathcal{M}$, we write $C(\mathcal{M})$ for the set of all characters in $\mathcal{M}$, and just C when the underlying matrix $\mathcal{M}$ is clear. We will also often refer to a set C of characters

without reference to an input matrix $\mathcal{M}$, and in this case the matrix $\mathcal{M}$ is assumed. We write $\mathcal{M}[\alpha_1, \ldots, \alpha_j]$ to denote $\mathcal{M}$ restricted to the columns in $\alpha_1, \ldots, \alpha_j$.

2.2.3 Partition Intersection Graphs

For input matrix $\mathcal{M}$, the *partition intersection graph* (or just *intersection graph*) of $\mathcal{M}$, denoted by $\mathcal{G}(\mathcal{M})$, is the graph having a vertex α_i for each character α of $\mathcal{M}$ and each state i of α, and an edge $\alpha_i\beta_j$ precisely when there is a taxon of $\mathcal{M}$ having state i for character α and state j for character β. Note that $\mathcal{G}(\mathcal{M})$ cannot have an edge between vertices associated with different states of the same character of $\mathcal{M}$. See Figure 2.3 for an example. Here we give a brief overview of some known results relating perfect phylogenies to partition intersection graphs.

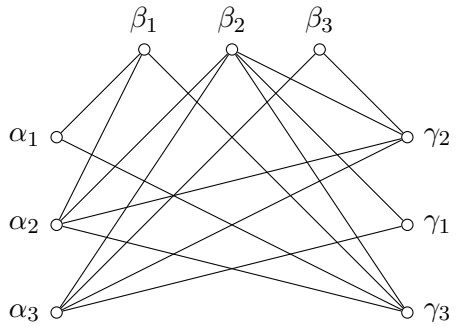


Figure 2.3: Partition intersection graph for the matrix $\mathcal{M}$ of Figure 2.1.

2.2.3.1 Partition Intersection Graphs and Perfect Phylogenies

A graph G is *triangulated* if and only if there are no induced chordless cycles of length four or greater. A *proper triangulation* of $\mathcal{G}(\mathcal{M})$ is a triangulated supergraph of $\mathcal{G}(\mathcal{M})$ such that each edge is between vertices of different characters of $\mathcal{M}$. The following theorem relates perfect phylogenies to proper triangulations [13, 57, 77].

Theorem 1. *There is a perfect phylogeny for $\mathcal{M}$ if and only if $\mathcal{G}(\mathcal{M})$ has a proper triangulation.*

We say that $\mathcal{M}$ is *pairwise compatible* if, for every pair α, β of characters in $\mathcal{M}$, there is a perfect phylogeny for $\mathcal{M}[\alpha, \beta]$. The following theorem gives a simple test for pairwise compatibility using the intersection graph of $\mathcal{M}$ [25].

Theorem 2. *Let α and β be two characters of $\mathcal{M}$. Then $\mathcal{M}[\alpha, \beta]$ has a perfect phylogeny if and only if $\mathcal{G}(\mathcal{M}[\alpha, \beta])$ is acyclic.*

The following theorem shows that Theorem 2 is sufficient for determining the compatibility of $\mathcal{M}$ when $\mathcal{M}$ consists only of 2-state characters [12, 24, 35, 57, 71].

Theorem 3. *An input matrix $\mathcal{M}$ of 2-state characters is compatible if and only if $\mathcal{M}$ is pairwise compatible.*

2.2.3.2 Partition Intersection Graphs and 3-State Perfect Phylogenies

When $\mathcal{M}$ contains characters with more than two states, pairwise compatibility is no longer sufficient to guarantee the compatibility of $\mathcal{M}$. This was first demonstrated in 1971 by Fitch [29, 30] who exhibited a set of three pairwise compatible 3-state characters that is incompatible. However, a recent breakthrough by Lam, Gusfield, and Sridhar [50] gives necessary and sufficient conditions on the compatibility of pairwise compatible 3-state characters by the existence of one of a set of forbidden subgraphs in the intersection graph of $\mathcal{M}$.

Theorem 4. *An input matrix $\mathcal{M}$ of 3-state characters admits a perfect phylogeny if and only if both of the following hold: (i) for every pair α, β of characters in $\mathcal{M}$, $\mathcal{G}(\mathcal{M}[\alpha, \beta])$ is acyclic; and (ii) for every triple $\{\alpha, \beta, \gamma\}$ of characters of $\mathcal{M}$, $\mathcal{G}(\mathcal{M}[\alpha, \beta, \gamma])$ does not contain, up to relabeling of characters and states, any of the subgraphs shown in Fig. 2.4.*

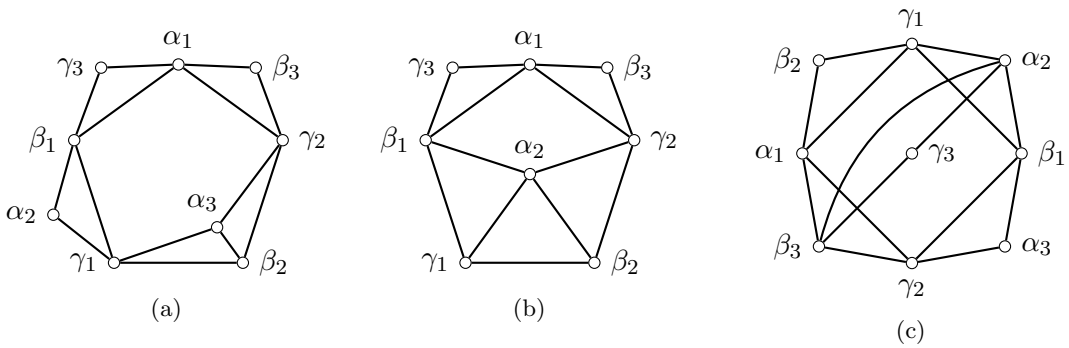


Figure 2.4: Three forbidden subgraphs for 3-state character compatibility from [50].

Note that, in regards to Conjecture 1, Theorem 4 shows that $f(3) = 3$.

Corollary 1. *$\mathcal{M}$ has a perfect phylogeny if and only if, for every triple α, β, γ of characters in $\mathcal{M}$, $\mathcal{M}[\alpha, \beta, \gamma]$ has a perfect phylogeny.*

2.2.4 Solving 3-State Perfect Phylogeny with 2-State Characters

Here we review a result of Dress and Steel [23]. Our exposition closely follows that of [37]. Our goal is to derive a matrix of 2-state characters $\overline{\mathcal{M}}$ from the matrix $\mathcal{M}$ of 3-state characters in such a way that the properties of $\overline{\mathcal{M}}$ enable use to find a perfect phylogeny for $\mathcal{M}$. The matrix $\overline{\mathcal{M}}$ contains three characters $\alpha(1)$, $\alpha(2)$, $\alpha(3)$ for each character α in $\mathcal{M}$. The characters $\alpha(1), \alpha(2), \alpha(3)$ are constructed so that all of the taxa that have state i for α in $\mathcal{M}$ are given state 1 for character $\alpha(i)$ in $\overline{\mathcal{M}}$, and the other taxa are given state 2 for $\alpha(i)$ in $\overline{\mathcal{M}}$.

Since every character in $\overline{\mathcal{M}}$ has two states, two characters $\alpha(i)$ and $\beta(j)$ of $\overline{M}$ are incompatible if and only if the two columns corresponding to $\alpha(i)$ and $\beta(j)$ contain all four of the pairs $(1,1)$, $(1,2)$, $(2,1)$, and $(2,2)$, otherwise they are compatible. This is known as the *four gametes test* [71]. The following theorem is from [23].

Theorem 5. *There is a perfect phylogeny for $\mathcal{M}$ if and only if there is a subset C of the characters of $\overline{M}$ such that*

(i) the characters in C are pairwise compatible, and

(ii) for each character α in $\mathcal{M}$, C contains at least two of the characters $\alpha(1)$, $\alpha(2)$, $\alpha(3)$.

Theorem 5 was used in [23] to give an $O(m^2 n)$ time algorithm to decide if there is a perfect phylogeny for $\mathcal{M}$. It was also used in [37] to reduce the 3-state perfect phylogeny problem in polynomial time to the well known 2-SAT problem, which is in P.

2.2.5 The Algorithm of Kannan and Warnow

We will use several structural results from an algorithm for the 3-state perfect phylogeny problem given by Kannan and Warnow [43].

The algorithm of [43] takes a divide and conquer approach to determining the compatibility of a set of 3-state characters. An instance is reduced to subproblems by finding a partition S_1, S_2 of the taxon set S of C with both of the following properties:

1. $2 \leq |S_i| \leq n-2, i = 1, 2.$

2. Whenever C is compatible on S, there is a perfect phylogeny P that contains an edge e whose removal breaks P into subtrees P_1 and P_2 with $\mathcal{L}(P_i) = S_i, i = 1, 2.$

A partition of S satisfying both of these properties is a *legal partition*, and the following theorem shows that finding such a partition for a given set of characters is the crux of the algorithm.

Theorem 6. *[43] Given a set C of three state characters, we can in $O(nk)$ time either find a legal partition of S or determine that the set of characters is incompatible.*

2.2.5.1 Finding a Legal Partition

We now discuss the manner in which such a legal partition is found for a set of 3-state characters C. Let T be a tree witnessing that C is compatible. The *canonical labeling* of T is the labeling where, for each internal node v of T, and each character $\alpha \in C$, if there are leaves x and y in different components of $T - \{v\}$ such that $\alpha(x) = \alpha(y)$, then $\alpha(v) = \alpha(x)$; otherwise $\alpha(v) = *$ where $*$ denotes a *dummy* state for C. Note that such a labeling of T always exists and is unique. We will assume that every compatible tree for C is canonically labeled.

The *tree-structure* for a character α in T is formed by repeatedly contracting edges of T connecting nodes that have the same state (other than $*$) for α. Note that this tree does not depend on the sequence of edge-contractions and is thus well defined. Furthermore, there is exactly one node for each state (other than the dummy state) of α, and each node labeled by $*$ has degree at least three. A tree-structure for α that is formed from some compatible tree for C is called a *realizable tree-structure* for α. There are four possible realizable tree-structures for a 3-state character α which are shown in Fig. 2.5.

To find a realizable tree structure for a character α, the algorithm examines the pairwise intersection patterns of α with every other character $\beta \in C$, and applies the following rules to rule out possible tree structures for α.

(a) A path P^i for each $i \in \{1, 2, 3\}$. (b) A star S^*.

Figure 2.5: The four possible realizable tree-structures for a 3-state character α.

Rule 1. *Let α and β be two characters of C. If, under some relabeling of the states of α and β, we have that $\alpha_1 \subseteq \beta_1$, $\alpha_2 \cap \beta_2 \neq \emptyset$, and $\alpha_3 \cap \beta_2 \neq \emptyset$, then P^1 is not a realizable tree-structure for α. If this is the case, we say that α and β match Rule 1 with respect to α_1.*

Rule 2.[1] *Let α and β be two characters of C. If, under some relabeling of the states of α and β, we have that $\alpha_1 \cap \beta_1 \neq \emptyset$, $\alpha_2 \cap \beta_1 \neq \emptyset$, $\alpha_2 \cap \beta_2 \neq \emptyset$, and $\alpha_3 \cap \beta_2 \neq \emptyset$, then P^2 is the only possible realizable tree-structure for α. If this is the case, we say that α and β match Rule 2 with respect to α_2.*

The set Q_α^C of *candidate* tree-structures for α are all of those possible tree-structures for α that are not ruled out after comparing the intersection pattern of α with every other character in C and applying Rules 1 and 2.

The following theorem which follows from [43] shows that a legal partition is found by choosing an arbitrary $\alpha \in C$ for which $Q_\alpha^C \neq \emptyset$. Furthermore, if there is an $\alpha \in C$ for which $Q_\alpha^C = \emptyset$, then C is incompatible.

Theorem 7 ([43])**.** *If $Q_\alpha^C \neq \emptyset$, then we can find a legal partition of S.*

Corollary 2. *A set C of 3-state characters is compatible if and only if $Q_\alpha^C \neq \emptyset$ for every $\alpha \in C$.*

2.3 Results on 3-state Perfect Phylogenies

Our goal in this section is to give a characterization of the situation when $\mathcal{M}$ is pairwise compatible, but does not have a perfect phylogeny, that can be inferred from the partition intersection graphs of the pairs of characters in $\mathcal{M}$. Thus, in this section, we fix $\mathcal{M}$ to be a

[1] *Rule 2 was stated incorrectly in [43]*

pairwise compatible n by m matrix of integers from the set $\{1, 2, 3\}$. Note that Theorem 2 gives a simple characterization of the situation when $\mathcal{M}$ is not pairwise compatible.

We say that a state i for a character α of $\mathcal{M}$ is *dependent* precisely when there exists a character β of $\mathcal{M}$, and two states j and k of β, such that $\alpha(i)$ is incompatible with both $\beta(j)$ and $\beta(k)$. In such a case, we say that the character β is a *witness* that the state i of α is dependent.

Lemma 2. *Let α be a character of $\mathcal{M}$ and let i be a dependent state of α. Then no pairwise compatible subset of characters in $\overline{M}$ satisfying Theorem 5 contains $\alpha(i)$.*

Proof. Let I be a pairwise compatible subset of the characters in $\overline{M}$ that contains $\alpha(i)$. Since state i of α is dependent, there is a character β in $\mathcal{M}$ and two states j, k of β, such that $\alpha(i)$ is incompatible with both $\beta(j)$ and $\beta(k)$. It follows that $\beta(j) \notin I$ and $\beta(k) \notin I$. But then I cannot possibly contain two of $\beta(1)$, $\beta(2)$, and $\beta(3)$. Thus, I cannot satisfy the conditions required in Theorem 5. □

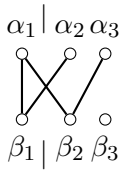
The next lemma gives a characterization of when a state is dependent using partition intersection graphs. We first introduce some notation: if $p : p_1p_2p_3p_4p_5$ is a path of length four in a graph, then we write middle$[p]$ to denote p_3, the *middle* vertex of p.

Lemma 3. *Let $\mathcal{M}$ be pairwise compatible. A state i of a character α of $\mathcal{M}$ is a dependent state if and only if there is a character β of $\mathcal{M}$ and a path p of length four in $\mathcal{G}(M[\alpha, \beta])$ with* middle$[p] = \alpha_i$.

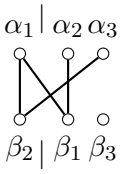
Proof. W.l.o.g. assume that $i = 1$, i.e., $\alpha_i = \alpha_1$.

($\Rightarrow$) Since 1 is a dependent state of α, there exists a character β in $\mathcal{M}$ such that $\alpha(1)$ is incompatible with two of $\beta(1)$, $\beta(2)$, and $\beta(3)$. W.l.o.g., assume $\alpha(1)$ is incompatible with both $\beta(1)$ and $\beta(2)$. Then, $\alpha_1\beta_1$ and $\alpha_1\beta_2$ are edges of $\mathcal{G}(M[\alpha, \beta])$, and, since $\mathcal{M}$ has no cycles, either $\beta_2\alpha_2$ and $\beta_1\alpha_3$ or $\beta_2\alpha_3$ and $\beta_1\alpha_2$ are edges of G. If $\beta_2\alpha_2$ and $\beta_1\alpha_3$ are edges of $\mathcal{G}(M[\alpha, \beta])$, then $\alpha_2\beta_2\alpha_1\beta_1\alpha_3$ is the required path of length four. If $\beta_2\alpha_3$ and $\beta_1\alpha_2$ are edges of $\mathcal{G}(M[\alpha, \beta])$, then $\alpha_3\beta_2\alpha_1\beta_1\alpha_2$ is the required path of length four.

($\Leftarrow$) Let β be a character of $\mathcal{M}$ such that there is a path p of length four in $\mathcal{G}(M[\alpha, \beta])$ with middle$[p] = \alpha_1$. Since $\mathcal{G}(M[\alpha, \beta])$ cannot contain edges between to states of the same



(a) $\alpha(1)$ and $\beta(1)$ are incompatible.



(b) $\alpha(1)$ and $\beta(2)$ are incompatible.

Figure 2.6: Illustrating the proof of Lemma 3.

character, we can assume w.l.o.g. that p is the path $\alpha_2\beta_1\alpha_1\beta_2\alpha_3$. Then, it is easy to verify that $\alpha(1)$ is incompatible with both $\beta(1)$ and $\beta(2)$. This is illustrated in Figure 2.6. □

Lemma 4. *If there is a character of $\mathcal{M}$ that has two dependent states, then no perfect phylogeny exists for $\mathcal{M}$.*

Proof. Let i and j be two dependent states of a character α of $\mathcal{M}$. Then, by Lemma 2, no pairwise compatible subset I of the characters of $\overline{M}$ that satisfy the condition required in Theorem 5 can contain $\alpha(i)$ or $\alpha(j)$. But then I can only contain one of $\alpha(1)$, $\alpha(2)$, or $\alpha(3)$. Hence, no pairwise compatible subset I of the characters of $\overline{M}$ can satisfy the condition required in Theorem 5. Hence, by Theorem 5, there is no perfect phylogeny for $\mathcal{M}$. □

We now show that the converse of Lemma 4 holds. We first give an observation relating the Rule 2 intersection pattern from the algorithm of Kannan and Warnow with dependent states.

Observation 1. *Let C be a set of 3-state characters, let $\alpha \in C$, and let α_i be a state of α. If there is a $\beta \in C$ where the intersection pattern of α and β matches Rule 2 with respect to α_i, then α_i is a dependent state of α.*

Theorem 8. *If a set C of 3-state characters is incompatible, then there exists a character $\alpha \in C$, and two distinct states α_i and α_j of α, such that both of the following hold:*

1. *There is a $\beta \in C$ where the intersection pattern of α and β matches Rule 2 with respect to α_i.*

2. *There is a $\gamma \in C$ where the intersection pattern of α and γ matches Rule 2 with respect to α_j.*

Proof. If C is pairwise incompatible, then by Theorem 2, there is a pair $\alpha, \beta \in C$ whose intersection graph contains a cycle. Since the intersection graph is bipartite, this cycle must have length at least four and contain at least two states of each character. Let α_i and α_j be the two states of α on this cycle. Then, the intersection pattern of α and β matches Rule 2 with respect to both α_i and α_j, and so the theorem holds. So we may assume that C is incompatible but pairwise compatible.

It follows from Corollary 2 that there exists an $\alpha \in C$ such that $Q_\alpha^C = \emptyset$. Then there must exist a character $\beta \in C$ such that the intersection pattern of α and β matches Rule 2 with respect to some state α_i of α; otherwise $S^* \in Q_\alpha^C$. Hence, $Q_\alpha^C \subseteq \{P^i\}$. Then, since $Q_\alpha^C = \emptyset$, there must be a character $\gamma \in C$ such that the intersection pattern of α and γ places a constraint on Q_α^C that prevents Q_α^C from containing P^i. There are two possibilities.

Case 1: There is a state α_j of α where $j \neq i$ and the intersection pattern of α and γ matches Rule 2 with respect to α_j. In this case the theorem holds.

Case 2: The intersection pattern of α and γ matches Rule 1 with respect to α_i. W.l.o.g., we fix $i = 1$, and relabel the states of α, β, and γ so that $\alpha_1 \cap \beta_1 \neq \emptyset$, $\alpha_1 \cap \beta_2 \neq \emptyset$, $\alpha_2 \cap \beta_1 \neq \emptyset$, $\alpha_3 \cap \beta_2 \neq \emptyset$, $\alpha_1 \subseteq \gamma_1$, $\alpha_2 \cap \gamma_2 \neq \emptyset$, and $\alpha_3 \cap \gamma_2 \neq \emptyset$. Such a labeling exists since, by assumption, α and β matches Rule 2 with respect to α_1, and α and γ matches Rule 1 with respect to α_1.

If $\alpha_2 \cap \gamma_1 \neq \emptyset$, then the intersection pattern of α and γ matches Rule 2 with respect to α_2, in which case the theorem holds. If $\alpha_3 \cap \gamma_1 \neq \emptyset$, then the intersection pattern of α and γ matches Rule 2 with respect to α_3, in which case the theorem holds. So we may assume hat $\alpha_1 = \gamma_1$. Now, since $\alpha_1 \cap \beta_1 \neq \emptyset$, $\alpha_1 \cap \beta_2 \neq \emptyset$, and $\alpha_1 = \gamma_1$, we have that both $\beta_1 \cap \gamma_1 \neq \emptyset$ and $\beta_2 \cap \gamma_2 \neq \emptyset$.

Clearly γ_3 must have a nonempty intersection with at least one state of α, and since $\alpha_1 = \gamma_1$, we have that $\alpha_1 \cap \gamma_3 = \emptyset$. So γ_3 has a nonempty intersection with either α_2 or α_3. Due to the symmetry of the intersection graph of α and β, we may assume, w.l.o.g., that $\alpha_3 \cap \gamma_3 \neq \emptyset$.

By assumption, $\alpha_2 \cap \gamma_1 = \emptyset$, and if $\alpha_2 \cap \gamma_3 \neq \emptyset$, then the intersection graph of α and β contains a cycle, contradicting our assumption that C is pairwise compatible. So we may assume that $\alpha_2 \subset \gamma_2$. Then, since $\beta_1 \cap \alpha_2 \neq \emptyset$, we have that $\beta_1 \cap \gamma_2 \neq \emptyset$.

Let $s \in \alpha_3 \cap \beta_2$. Since, by assumption, $\alpha_3 \cap \gamma_1 = \emptyset$, we have that either $s \in \gamma_2$ or $s \in \gamma_3$.

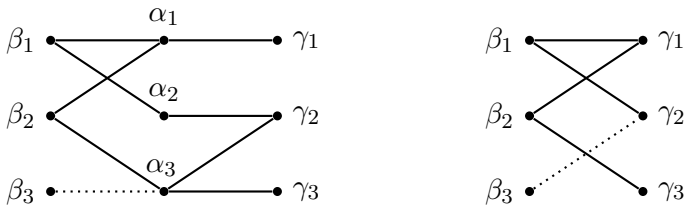


Figure 2.7: Illustrating the proof of Theorem 8.

However, if $s \in \gamma_2$, then $\beta_2 \cap \gamma_2 \neq \emptyset$ and intersection graph of β and γ contains a cycle, contradicting our assumption that C is pairwise compatible. Hence $s \in \gamma_3$ and $\beta_2 \cap \gamma_3 \neq \emptyset$.

We have now established all of the edges of the intersection graph of α, β, and γ represented by the solid edges in Fig. 2.7. Now, let $t \in \alpha_3 \cap \gamma_2$. Now t must be in some state of β. If $t \in \beta_1$, then $t \in \beta_1 \cap \alpha_3$ and the intersection graph of β and α contains a cycle, contradicting our assumption that C is pairwise compatible. If $t \in \beta_2$, then $t \in \beta_2 \cap \gamma_2$, and the intersection graph of β and γ contains a cycle, again contradicting our assumption that C is pairwise compatible. Hence $t \in \beta_3$. Then, we have that $t \in \beta_3 \cap \alpha_3$ and $t \in \beta_3 \cap \gamma_2$, witnessing the dotted edges in Fig. 2.7. So we have that the intersection pattern of β and α matches Rule 2 with β_2 as witness, and the intersection pattern of β and γ matches Rule 2 with β_1 as witness. Hence the theorem holds. □

Note that in the statement of Theorem 8, the characters β and γ are not necessarily distinct. However, in cases where they are not distinct, C contains an incompatible pair.

Observation 1 together with Theorem 8 gives the following corollary which is the converse of Lemma 4.

Corollary 3. *If no perfect phylogeny exists for $\mathcal{M}$, then there is a character of $\mathcal{M}$ that has two dependent states.*

We remark here that the forbidden subgraphs for 3-state character compatibility given in [50] can also be used to establish Corollary 3. Figure 2.8 reproduces each of the forbidden subgraphs of [50] with the paths of length four witnessing the dependent states highlighted in blue and red.

Lemma 4 together with Corollary 3 gives the following theorem.

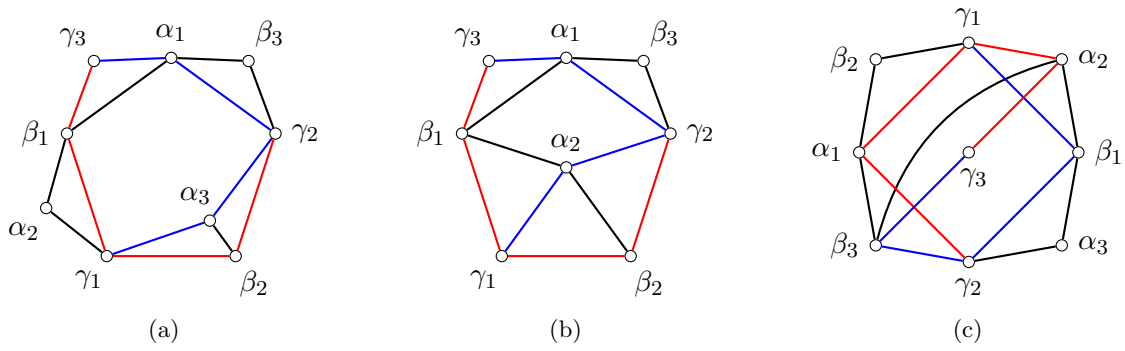


Figure 2.8: An alternate proof of Corollary 3 using the forbidden subgraphs of [50].

Theorem 9. *If the characters of $\mathcal{M}$ have at most three states, then $\mathcal{M}$ is compatible if and only if each character of $\mathcal{M}$ has at most one dependent state.*

In [50], it was shown using an elaborate argument on triangulations of the intersection graph of C, that C is compatible if and only if every subset of at most three characters of C is compatible. Since each dependent state of a character α requires only one other character as witness, and two dependent states of the same character are necessary and sufficient for compatibility, Theorem 9 provides an independent proof of this result with the following corollary.

Corollary 4. *A set C of 3-state characters is compatible if and only if every subset of at most three characters of C is compatible.*

Another interesting and immediate consequence of Theorem 9 is that every set C of 3-state characters has a canonical subset that *does* have a perfect phylogeny, namely the subset $\{\alpha \in C : \alpha \text{ has at most one dependent state}\}$.

2.3.1 A Single Forbidden Subgraph for 3-state Perfect Phylogenies

In [50], it was also shown that we can determine the compatibility of a pairwise compatible set C of 3-state characters by testing the intersection patterns of C for the existence of one of a set of four forbidden patterns. As a corollary to Theorem 8, we have that a single forbidden pattern suffices to determine the compatibility of C.

Corollary 5. *A pairwise compatible set C of 3-state characters is compatible if and only if the partition intersection graph of C does not contain, up to relabeling of characters and states, the subgraph of Figure 2.9.*

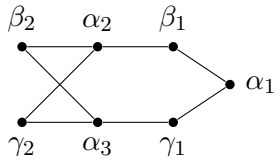


Figure 2.9: A single forbidden subgraph for 3-state character compatibility.

2.3.2 Finding All Minimal Obstruction Sets

We now describe an algorithm, denoted by $\mathcal{A}$, which outputs all of the minimal obstruction sets for $\mathcal{M}$. Step 1 of $\mathcal{A}$ computes for each character α of $\mathcal{M}$ the following.

- A set $\mathcal{B}(\alpha)$ of all characters β of $\mathcal{M}$ such that $\mathcal{G}(\mathcal{M}[\alpha, \beta])$ contains a cycle.
- For each state i of α, a set $\mathcal{D}(\alpha, i)$ of all characters β of $\mathcal{M}$ such that $\beta \notin \mathcal{B}(c)$ and there is a path p of length four in $\mathcal{G}(\mathcal{M}[\alpha, \beta])$ with $\alpha_i = \text{middle}[p]$.

Step 2 of $\mathcal{A}$ visits each character α of $\mathcal{M}$ and outputs the following.

- For each character β in $\mathcal{B}(\alpha)$, the set $\{\alpha, \beta\}$.
- For each pair of states $\{i, j\}$ of α with both $\mathcal{D}(\alpha, i)$ and $\mathcal{D}(\alpha, j)$ non-empty, each element of the set $\{\{\alpha, \chi_i, \chi_j\} : \chi_i \in \mathcal{D}(c, i), \chi_j \in \mathcal{D}(c, j)\}$.

Theorem 10. *$\mathcal{A}$ outputs all p minimal obstruction sets for $\mathcal{M}$ in $O(m^2n + p)$ time.*

Proof. We first establish the following claim.

Claim 1. For each character α of $\mathcal{M}$, the sets $\mathcal{B}(\alpha)$, $\mathcal{D}(\alpha, 1)$, $\mathcal{D}(\alpha, 2)$, and $\mathcal{D}(\alpha, 3)$ are pairwise disjoint.

Proof of Claim 1. By construction, for each character α of $\mathcal{M}$, $\mathcal{B}(\alpha) \cap (\mathcal{D}(\alpha, 1) \cup \mathcal{D}(\alpha, 2) \cup \mathcal{D}(\alpha, 3)) = \emptyset$. So it suffices to show that for each character α of $\mathcal{M}$, the sets $\mathcal{D}(\alpha, 1)$, $\mathcal{D}(\alpha, 2)$,

and $\mathcal{D}(\alpha, 3)$ are pairwise disjoint. W.l.o.g. let α be a character of $\mathcal{M}$ and let $\beta \in \mathcal{D}(c, 1) \cap \mathcal{D}(c, 2)$. Let $G = \mathcal{G}(\mathcal{M}[\alpha, \beta])$. Since $\beta \in \mathcal{D}(\alpha, 1) \cap \mathcal{D}(\alpha, 2)$, $\beta \notin \mathcal{B}(\alpha)$, and there are paths p_1 and p_2 of length four in G with $\alpha_1 = \text{middle}[p_1]$ and $\alpha_2 = \text{middle}[p_2]$. Since $\beta \notin \mathcal{B}(\alpha)$, G is acyclic. W.l.o.g. suppose that p_1 is the path $\alpha_2\beta_1\alpha_1\beta_2\alpha_3$. Since $\text{middle}[p_2] = \alpha_2$, there must be two edges from α_2 to vertices associated with states of β. We have that $\alpha_2\beta_1$ is an edge of p_1. If $\alpha_2\beta_2$ is an edge of p_2, then we have a cycle in G. So $\alpha_2\beta_3$ and either $\beta_3\alpha_3$ or $\beta_3\alpha_1$ are edges of p_2. In either case, there is a cycle in G. ◇

By Lemma 3, $\mathcal{A}$ finds all dependent states, and hence, by Theorems 4, outputs all of the minimal obstruction sets for $\mathcal{M}$. By Claim 1, every obstruction set output by $\mathcal{A}$ is minimal. We now establish the runtime. Step 1 of $\mathcal{A}$ takes $O(m^2n)$ time to construct the intersection graphs of each pair of characters of $\mathcal{M}$. Since each intersection graph has exactly six vertices and at most nine edges, it follows that it all cycles and paths of length four can be found in $O(1)$ time. Hence step 1 takes $O(m^2n)$ time. Step 2 of $\mathcal{A}$ visits each of the m characters of $\mathcal{M}$ and takes $O(1)$ time per set output. Any minimal obstruction set of cardinality two will be output twice. If follows from Claim 1 that each minimal obstruction set of cardinality three will be output at most three times. Thus, step 2 takes $O(m + p)$ time where p is the number of minimal obstruction sets. Hence, $\mathcal{A}$ takes $O(m^2n + p)$ time to complete both steps 1 and 2. □

2.3.3 Dependent States and Legal Minimal Separators

Several approaches to determining the existence of a perfect phylogeny for $\mathcal{M}$ studied in the literature make use of separating sets in $\mathcal{G}(\mathcal{M})$ [36, 38]. For two vertices a and b of $\mathcal{G}(\mathcal{M})$, an $a - b$ *separator* is a set of vertices whose removal separates a from b. An $a - b$ separator is *minimal* if no subset of it is an $a - b$ separator. A *minimal separator* is a separator that is a minimal $a - b$ separator for some pair a, b of vertices of $\mathcal{G}(\mathcal{M})$. A minimal separator S of $\mathcal{G}(\mathcal{M})$ is *legal* if, for each character c of $\mathcal{M}$, S contains at most one vertex corresponding to a state of c. A pair of vertices of $\mathcal{G}(\mathcal{M})$ represent different states of the same character is *monochromatic*. The following theorem follows from results given in [36, 38, 50].

Theorem 11. *$\mathcal{M}$ admits a perfect phylogeny if and only if both of the following hold: (i) the characters of $\mathcal{M}$ are pairwise compatible; and (ii) every monochromatic pair of vertices in $\mathcal{G}(\mathcal{M})$ is separated by a legal minimal separator.*

We conclude with Theorem 12 that relates dependent states to legal minimal separators. A consequence of Theorem 12 is that algorithm $\mathcal{A}$ can be easily modified to output monochromatic pairs of vertices of $\mathcal{G}(\mathcal{M})$ with no legal minimal separator.

Theorem 12. *Suppose that the characters of $\mathcal{M}$ are pairwise compatible. Two states i and j of a character α of $\mathcal{M}$ are dependent if and only if there is no legal minimal separator for α_i and α_j in $\mathcal{G}(\mathcal{M})$.*

Proof. By Theorems 4 and 11, it suffices to show that the theorem holds for every minimal obstruction set, i.e., for each graph in Fig. 2.4, a monochromatic pair of vertices has no legal minimal separator if and only if they correspond to a pair of dependent states. This is verified by inspection. In the graph of Fig. 2.4a: $\{\gamma_1, \gamma_2\}$ is the only *monochromatic* pair of vertices with no legal minimal separator; 3 is the only dependent state of α; 2 is the only dependent state of β; and 1 and 2 are the only dependent states of γ. In the graph of Fig. 2.4b: (β_1, β_2) and (γ_1, γ_2) are the only monochromatic pairs of vertices with no legal minimal separator; there are no dependent states of α; 1 and 2 are the only dependent states of β; and 1 and 2 are the only dependent states of γ. In the graph of Fig. 2.4c: (α_1, α_2), (β_1, β_3), (γ_1, γ_2), and are the only monochromatic pairs of vertices with no legal minimal separator; 1 and 2 are the only dependent states of α; 1 and 3 are the only dependent states of β; and 1 and 2 are the only dependent states of γ. □

2.4 Results on Multi-State Perfect Phylogenies

We now turn to the general case where we consider where $\mathcal{M}$ is composed of k-state characters for $k \geq 2$. Our main result is to prove the conjecture stated in [39] the function $f(k)$ of Conjecture 1 is bounded below by $f(k) \geq k + 1$. We do so by giving a quadratic lower bound on $f(k)$. Our proof relies on a new result on quartet compatibility which is given first. We then

exploit a natural correspondence between quartet compatibility and character compatibility to prove the lower bound on $f(k)$.

2.4.1 Incompatible Quartets

For every $s, t \geq 2$, we fix a set of labels $\mathcal{L}_{s,t} = \{a_1, a_2, \ldots, a_s, b_1, b_2, \ldots, b_t\}$ and define the set

$$Q_{s,t} = \{a_1 b_1 | a_s b_t\} \cup \bigcup_{i=1}^{s-1} \bigcup_{j=1}^{t-1} \{a_i a_{i+1} | b_j b_{j+1}\}$$

of quartets with $\mathcal{L}(Q_{s,t}) = \mathcal{L}_{s,t}$. We denote the quartet $a_1 b_1 | a_s b_t$ by q_0, and a quartet of the form $a_i a_{i+1} | b_j b_{j+1}$ by $q_{i,j}$.

Observation 2. *For all $s, t \geq 2$, $|Q_{s,t}| = (s-1)(t-1) + 1$.*

Lemma 5. *For all $s, t \geq 2$, $Q_{s,t}$ is incompatible.*

Proof. For each $i \in [s-1]$,

$$\bigcup_{j=1}^{t-1} \{a_i a_{i+1} | b_j b_{j+1}\} \subseteq Q_{s,t} \subseteq Q^*_{s,t}.$$

Then, by Lemma 1, it follows that for each $i \in [s-1]$, $a_i a_{i+1} | b_1 b_t \in Q^*_{s,t}$. So,

$$\bigcup_{i=1}^{s-1} \{b_1 b_t | a_i a_{i+1}\} \subseteq Q^*_{s,t}.$$

Then, again by Lemma 1, it follows that $b_1 b_t | a_1 a_s \in Q^*_{s,t}$. But then $\{a_1 b_1 | a_s b_t, b_1 b_t | a_1 a_s\} \subseteq Q^*_{s,t}$. It follows that any tree that displays $Q_{s,t}$ must display both $a_1 b_1 | a_s b_t$ and $b_1 b_t | a_1 a_s$. However, no such tree exists. Hence, $Q_{s,t}$ is incompatible. □

Lemma 6. *For all $s, t \geq 2$, every proper subset of $Q_{s,t}$ is compatible.*

Proof. Since every subset of a compatible set of quartets is compatible, it suffices to show that for every $q \in Q_{s,t}$, $Q_{s,t} \setminus \{q\}$ is compatible. Let $q \in Q_{s,t}$. Either $q = q_0$ or $q = q_{x,y}$ for some $1 \leq x < s$ and $1 \leq y < t$. In either case, we exhibit a tree witnessing that $Q_{s,t} \setminus \{q\}$ is compatible.

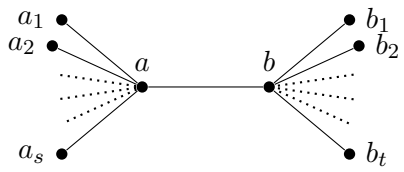


Figure 2.10: Case 1 in the proof of Lemma 6.

Case 1. Suppose $q = q_0$. We build the tree T as follows: There is a node ℓ for each label $\ell \in \mathcal{L}_{s,t}$ and two additional nodes a and b along with the edge ab. There is an edge $a_x a$ for every $a_x \in \mathcal{L}_{s,t}$, and an edge $b_x b$ for every $b_x \in \mathcal{L}_{s,t}$. There are no other nodes or edges in T. See Fig. 2.10 for an illustration of a tree displaying $Q_{s,t} \setminus \{q_0\}$. Now consider any quartet $q \in Q_{s,t} \setminus \{q_0\}$. Then $q = a_i a_{i+1} | b_j b_{j+1}$ for some $1 \leq i < s$ and $1 \leq j < t$. Then, the minimal subgraph of T connecting leaves with labels in $\{a_i, a_{i+1}, b_j, b_{j+1}\}$ is the quartet q. Hence T displays q.

Case 2. Suppose $q = q_{x,y}$ for some $1 \leq x < s$ and $1 \leq y < t$. We build the tree T as follows: There is a node ℓ for each label $\ell \in \mathcal{L}_{s,t}$ and six additional nodes a_ℓ, b_ℓ, ℓ, h, a_h, and b_h. There are edges $a_\ell \ell$, $b_\ell \ell$, ℓh, $h a_h$, and $h b_h$. For every $a_i \in \mathcal{L}_{s,t}$, there is an edge $a_i a_\ell$ if $i \leq x$, and an edge $a_i a_h$ if $i > x$. For every $b_j \in \mathcal{L}_{s,t}$ there is an edge $b_j b_\ell$ if $j \leq x$, and an edge $b_j b_h$ if $j > y$. There are no other nodes or edges in T. See Fig. 2.11 for an illustration of a tree displaying $Q_{s,t} \setminus \{q_{x,y}\}$. Now consider any quartet $q \in Q_{s,t} \setminus \{q_{x,y}\}$. Either $q = q_0$ or $q = q_{i,j}$ where $i \neq x$ or $j \neq y$. If $q = q_0$, then the minimal subgraph of T connecting leaves with labels in $\{a_1, b_1, a_s, b_t\}$ is the subtree of T induced by the nodes in $\{a_1, a_\ell, \ell, b_\ell, b_1, a_s, a_h, h, b_h, b_t\}$. Suppressing all degree two vertices results in a tree that is the same as q_0. So T displays q. So assume that $q = a_i a_{i+1} | b_j b_{j+1}$ where $i \neq x$ or

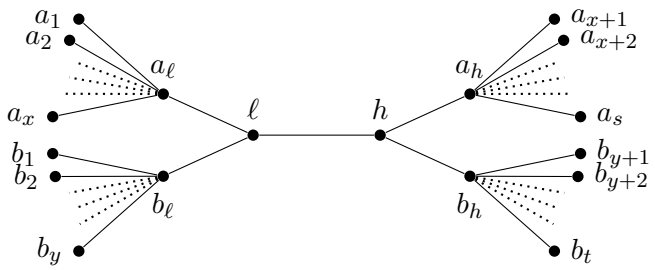


Figure 2.11: Case 2 in the proof of Lemma 6.

$j \neq y$. We define the following subset of the nodes in T:

$$V = \begin{cases} \{a_i, a_{i+1}, a_\ell, \ell, b_\ell, b_j, b_{j+1}\} & \text{if } i < x \text{ and } j < y, \\ \{a_i, a_{i+1}, a_\ell, \ell, b_y, b_\ell, h, b_h, b_{y+1}\} & \text{if } i < x \text{ and } j = y, \\ \{a_i, a_{i+1}, a_\ell, \ell, h, b_h, b_j, b_{j+1}\} & \text{if } i < x \text{ and } j > y, \\ \{a_x, a_\ell, \ell, h, a_h, a_{x+1}, b_\ell, b_j, b_{j+1}\} & \text{if } i = x \text{ and } j < y, \\ \{a_x, a_\ell, \ell, h, a_h, a_{x+1}, b_h, b_j, b_{j+1}\} & \text{if } i = x \text{ and } j > y, \\ \{a_j, a_{j+1}, a_h, h, \ell, b_\ell, b_j, b_{j+1}\} & \text{if } i > x \text{ and } j < y, \\ \{a_j, a_{j+1}, a_h, h, b_y, b_\ell, \ell, b_h, b_{y+1}\} & \text{if } i > x \text{ and } j = y, \\ \{a_j, a_{j+1}, a_h, h, b_h, b_j, b_{j+1}\} & \text{if } i > x \text{ and } j > y. \end{cases}$$

Now, the subgraph of T induced by the nodes in V is the minimal subgraph of T connecting leaves with labels in q. Suppressing all degree two vertices gives q. Hence, T displays q.

□

With $s = \lfloor \frac{n}{2} \rfloor$ and $t = \lceil \frac{n}{2} \rceil$, Observation 2 and Lemmas 5 and 6 imply the following theorem.

Theorem 13. *For every integer $n \geq 4$, there exists a set Q of quartets over n taxa such that all of the following conditions hold.*

1. *Q is incompatible.*
2. *Every proper subset of Q is compatible.*
3. *$|Q| = \lfloor \frac{n-2}{2} \rfloor \cdot \lceil \frac{n-2}{2} \rceil + 1$.*

2.4.1.1 Incompatible Quartets on Five Taxa

When Q is a set of quartets over five taxa, we show that the set of quartets given by Theorem 13 is as large as possible. We hope that the technique used in the proof of the following theorem might be useful in proving tight bounds for $n > 5$.

Theorem 14. *If Q is an incompatible set of quartets over five taxa such that every proper subset of Q is compatible, then $|Q| \leq 3$.*

Proof. Let Q be an incompatible set of quartets with $\mathcal{L}(Q) = \{a, b, c, d, e\}$ and $q_0 = ab|cd \in Q$. We will show that Q contains an incompatible subset of at most three quartets. If Q contains two different quartets on the same four taxa, then Q must contain an incompatible pair of quartets. So, we may assume that each quartet is on a unique subset of four of the five taxa. Hence, every pair of quartets in Q shares three taxa in common. We have the following two cases.

> *Case 1:* Q contains at least one of the quartets $ac|be$, $ac|de$, $ad|be$, $ad|ce$, $ae|bc$, $ae|bd$, $bc|de$, or $bd|ce$. W.l.o.g. we may assume that Q contains $q_1 = ac|de$, as all other cases are symmetric. By (R2), $\{q_0, q_1\} \vdash ab|ce$. Then, by (R1), $\{q_0, q_1, ab|ce\} \vdash ab|de$. Then, again by (R1), $\{q_0, q_1, ab|ce, ab|de\} \vdash bc|de$. Now let $Q' = \{q_0, q_1, ab|ce, ab|de, bc|de\}$. Now, any quartet in Q must be either in Q' or be pairwise incompatible with a quartet in Q'. Since Q' is compatible, but by assumption, Q is incompatible, Q must contain a quartet q_2 that is pairwise incompatible with some quartet in Q'. Hence, $\{q_0, q_1, q_2\}$ is an incompatible subset of Q.
>
> *Case 2:* Q contains none of the quartets $ac|be$, $ac|de$, $ad|be$, $ad|ce$, $ae|bc$, $ae|bd$, $bc|de$, or $bd|ce$. Then every quartet in Q is either of the form $ab|xy$ where $\{x, y\} \neq \{c, d\}$, or $cd|xy$ where $\{x, y\} \neq \{a, b\}$. But then Q is compatible, contradicting our assumption that Q is incompatible.

In either case, the theorem holds. □

2.4.1.2 Incompatible Quartets on Arbitrarily Many Taxa

We say a set Q of compatible quartets is *redundant* if for some $q \in Q$, $Q \setminus \{q\} \vdash q$; otherwise, we say that Q is *irredundant*. The following lemma establishes a connection between sets of irredundant quartets and minimal sets of incompatible quartets.

Lemma 7. *If Q is incompatible, but every proper subset of Q is compatible, then every proper subset of Q is irredundant.*

Proof. Suppose that Q is incompatible and every proper subset of Q is compatible. Furthermore, suppose that some proper subset Q' of Q is redundant. Since every compatible superset of a redundant set of quartets is also redundant, we may assume w.l.o.g., that there is a unique quartet $q \in Q \setminus Q'$ (i.e., $|Q| = |Q'| + 1$). Since Q' is redundant, there exists a $q' \in Q'$ such that $Q' \setminus \{q'\} \vdash q'$. But then $(Q' \setminus \{q'\}) \cup \{q\}$ is incompatible, contradicting that every proper subset of Q is compatible. □

It follows from Lemma 7 that any upper bound on the maximum cardinality of an irredundant set of quartets can be used to place an upper bound on the maximum cardinality of a set of quartets satisfying the first two conditions of Theorem 13. The theorem follows from [20].

Theorem 15. *Let Q be a set of quartets over a set of n taxa. If Q is irredundant, then Q has cardinality at most $(n-3)(n-2)^2/3$.*

Lemma 7 together with Theorem 15 gives the following upper bound on the maximum cardinality of a set Q of quartets over $n > 5$ taxa that satisfies the first two conditions of Theorem 13.

Theorem 16. *Let Q be a set of incompatible quartets over a set of n taxa such that every proper subset of Q is compatible. Then $|Q| \leq (n-3)(n-2)^2/3 + 1$.*

2.4.2 Incompatible Characters

There is a natural correspondence between quartet compatibility and character compatibility that we now describe. Let Q be a set of quartets, $n = |\mathcal{L}(Q)|$, and $r = n - 2$. For each

$q = ab|cd \in Q$, we define the r-state character corresponding to q, denoted χ_q, as the character where a and b have state 0 for χ_q; c and d have state 1 for χ_q; and, for each $\ell \in \mathcal{L}(Q) \setminus \{a, b, c, d\}$, there is a state s of χ_q such that ℓ is the only label with state s for character χ_q (see Example 2). We define the set of r-state characters corresponding to Q by $C_Q = \bigcup_{q \in Q} \{\chi_q\}$.

Example 2. *Consider the quartets and characters given in Fig. 2.2(a): χ_{q_1} is the character corresponding to q_1, χ_{q_2} is the character corresponding to q_2, and χ_{q_3} is the character corresponding to q_3.*

The following lemma relating quartet compatibility to character compatibility is well known [78], and its proof is omitted here.

Lemma 8. *A set Q of quartets is compatible if and only if C_Q is compatible.*

The next theorem allows us to use our result on quartet compatibility to establish a lower bound on $f(k)$.

Theorem 17. *Let Q be a set of incompatible quartets over n labels such that every proper subset of Q is compatible, and let $r = n - 2$. Then, there exists a set C of $|Q|$ r-state characters such that C is incompatible, but every proper subset of C is compatible.*

Proof. We claim that C_Q is such a set of incompatible r-state characters. Since for two quartets $q_1, q_2 \in Q$, $\chi_{q_1} \neq \chi_{q_2}$, it follows that $|C_Q| = |Q|$. Since Q is incompatible, it follows by Lemma 8 that C_Q is incompatible. Let C' be any proper subset of C. Then, there is a proper subset Q' of Q such that $C' = C_{Q'}$. Then, since Q' is compatible, it follows by Lemma 8 that C' is compatible. □

Theorem 13 together with Theorem 17 gives the main theorem of this paper.

Theorem 18. *For every integer $k \geq 2$, there exists a set C of k-state characters such that all of the following hold.*

1. *C is incompatible.*

2. *Every proper subset of C is compatible.*

3. $|C| = \lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil + 1$.

Proof. By Theorem 13 and Observation 2, there exists a set Q of $\lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil + 1$ quartets over $k+2$ labels that that are incompatible, but every proper subset is compatible, namely $Q_{\lfloor \frac{k+2}{2} \rfloor, \lceil \frac{k+2}{2} \rceil}$. The theorem follows from Theorem 17. □

The quadratic lower bound on $f(k)$ follows from Theorem 18.

Corollary 6. $f(k) \geq \lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil + 1$.

2.5 Conclusion

We have shown that for every $k \geq 2$, $f(k) \geq \lfloor \frac{k}{2} \rfloor \cdot \lceil \frac{k}{2} \rceil + 1$, by showing that for every $n \geq 4$, there exists an incompatible set Q of $\lfloor \frac{n-2}{2} \rfloor \cdot \lceil \frac{n-2}{2} \rceil + 1$ quartets over a set of n labels such that every proper subset of Q is compatible. Previous results [12, 24, 35, 50, 57, 71], along with our discussion in Section 2.4.2, show that our lower bound on $f(k)$ is tight for $k = 2$ and $k = 3$. For quartets, our discussion in Section 2.4.1 gives an upper bound on the maximum cardinality of a minimal set of incompatible quartets. However, this argument does not extend to multi-state characters. Indeed, an upper bound on the maximum cardinality of a minimal set of incompatible k-state characters remains a central open question. We give the following conjecture.

Conjecture 2. $f(k) \in \Theta(k^2)$.

A less ambituous goal would be to narrow the gap between the upper bound of $O(n^3)$ and lower bound of $\Omega(n^2)$ on the maximum cardinality of a minimal incompatible set of quartets over n taxa given in Section 2.4.1. Note that, due to Theorem 17, a proof of Conjecture 2 would also show that the number of incompatible quartets given in the statement of Theorem 13 is also as large as possible.

CHAPTER 3. Fixed-Parameter Algorithms for Finding Agreement Supertrees

Modified from a paper submitted to *SIAM Journal on Computing*

David Fernández-Baca, Sylvain Guillemot, Brad Shutters, and Sudheer Vakati

Abstract

We study the agreement supertree approach for combining rooted phylogenetic trees when the input trees do not fully agree on the relative positions of the taxa. Two approaches to dealing with such conflicting input trees are considered. The first is to contract a set of edges in the input trees so that the resulting trees have an agreement supertree. We show that this problem is NP-complete and give an FPT algorithm for the problem parameterized by the number of input trees and the number of edges contracted. The second approach is to remove a set of taxa from the input trees so that the resulting trees have an agreement supertree. An FPT algorithm for this problem when the input trees are all binary was given by Guillemot and Berry (2010). We give an FPT algorithm for the more general case when the input trees have arbitrary degree.

3.1 Introduction

A phylogeny, or evolutionary tree, is a tree representing the evolutionary history of a set of species. The leaves of the tree represent the current species (taxa), and the internal nodes of the tree represent the hypothetical ancestors. A fundamental problem in phylogenetics is to construct a supertree from smaller input trees with overlapping taxa in such a way that the inferred supertree complies as closely as possible with the topological information of the

input trees. This problem is motivated by the biological and computational constraints on constructing large scale phylogenies. The supertree problem was introduced in [31], and a variety of supertree construction methods have been proposed. See [7, 68, 3, 77, 11] for more on supertrees.

In this paper we use the agreement supertree approach for combining rooted phylogenetic trees. The goal of this approach is to search for a supertree such that each of the input trees is a restriction of the supertree to a subset of its taxa. Formally, we have the following decision problem.

Agreement Supertree (AST)

Input: A collection $\mathcal{T}$ of k rooted phylogenetic trees on a set of n taxa.

Question: Does there exist an agreement supertree for $\mathcal{T}$?

The answer to an instance of AST is "yes" if and only if the input trees fully agree on the relative positions of the taxa, in which case the input trees are said to agree. There is a polynomial time algorithm for the AST problem, which returns an agreement supertree if one exists [58].

The input trees may fail to have an agreement supertree because of conflicts with respect to the relative positions of some taxa. Such conflicts arise due to errors in the inference process, or due to biological processes, e.g., lateral gene transfer, gene duplication, and others [55, 52]. Here we consider two approaches for dealing with conflicting input trees. The first addresses the case where conflict is due to unnecessary edges in the input trees. The goal is to find a subset of the edges of the input trees to contract so that the resulting collection of trees agree. Formally, we focus on the following decision problem.

Agreement Supertree Edge Contraction (AST-EC)

Input: A collection $\mathcal{T}$ of k rooted phylogenetic trees on a set of n taxa, and an integer p.

Question: Can we contract at most p internal edges of $\mathcal{T}$ so that the trees in $\mathcal{T}$ agree?

The AST-EC problem does not seem to have been considered before. We prove that problem is NP-complete, and show that it is fixed-parameter tractable for parameters k and p.

The second approach we study addresses the case where disagreement is due to misplaced taxa. The goal is to find a subset of the taxa to remove from the input trees so that the resulting collection of trees agree. Formally, we focus on the following decision problem.

Agreement Supertree Taxon Removal (AST-TR)

Input: A collection $\mathcal{T}$ of k rooted phylogenetic trees on a set of n taxa, and an integer p.

Question: Can we remove at most p taxa so that the input trees agree?

The AST-TR problem is NP-complete [42, 6], but was shown to be fixed-parameter tractable in k and p when restricted to the case when the input trees are all binary [34]. Our contribution is to show that the more general AST-TR problem, where the input trees are allowed to have arbitrary degree, is fixed-parameter tractable in k and p. It was also shown in [6] that if AST-TR is parameterized by only k or p, then the problem is fixed-parameter intractable. We also note that the optimization version of AST-TR, i.e., finding a minimum set of taxa to remove, is the dual of the Maximum Agreement Supertree (Smast) problem [6, 42, 45]. Exact algorithms for Smast on binary trees are known that run in time $O(6^k n^k)$ [34, 40] and, when the maximum degree of the input trees is d, [40] gives an $O((kd)^{kd+3}2^k n^k)$ time algorithm for Smast.

The rest of this paper proceeds as follows. In Section 3.2, we give basic definitions needed for the remainder of the paper. In Section 3.3, we develop a characterization of when a set of input trees agree. We then use this characterization to develop an algorithm for testing agreement that solves the AST problem. We remark here that this algorithm could be easily modified to produce an agreement supertree when the set of input trees agree. If the algorithm

answers in the negative, it returns a subset of the internal nodes of the trees in $\mathcal{T}$ encapsulating the taxa on which the trees in $\mathcal{T}$ disagree. In Section 3.4 we use these internal nodes to develop $O((2k)^p kn^2)$ time algorithms to solve the AST-EC and AST-TR problems. We also prove the NP-completeness of the AST-EC problem by giving a reduction from MULTICUT to AST-EC. Section 3.5 contains proofs of some results from Sections 3.3 and 3.4, that were deferred for readability. We conclude in Section 3.6 with some ideas for future research.

3.2 Definitions

Let $\mathcal{T} = \{T_1, \ldots, T_k\}$ be a collection of rooted phylogenetic trees, and let T be some arbitrary tree in $\mathcal{T}$. We use $V(T)$, $E(T)$, $\mathcal{I}(T)$, $\hat{E}(T)$, and $r(T)$ to denote the vertices, edges, internal vertices, internal edges, and root vertex of T respectively. We use $\mathcal{L}(T)$ and $L(T)$ to denote the leaves of T and the set of labels mapped to the leaves of T respectively. We write $L(\mathcal{T})$ for $\bigcup_{i\in[k]} L(T_i)$ and $\hat{E}(\mathcal{T}) = \bigcup_{i\in[k]} \hat{E}(T_i)$, where $[k]$ stands for $\{1, \ldots, k\}$. We represent that a vertex v is an ancestor of u in T by $u \leq_T v$. For two vertices u and v such that $u <_T v$, we write $\text{child}_T(v, u)$ to denote the child of v along the path from v to u in T. For each $u \in V(T)$, we use $\text{parent}(u)$, $\text{Ch}(u)$, $T(u)$, and $L(u)$ to denote the parent of u, the children of u, the subtree of T rooted at u, and the set of labels mapped to the leaves of $T(u)$, respectively.

For a label set L, the *restriction* of T to L, denoted by $T|L$, is the minimal homeomorphic subtree of T connecting leaves with labels in L. For a set $L \subseteq L(\mathcal{T})$, we write $\mathcal{T}|L$ for the collection $\{T_1|L, \ldots, T_k|L\}$ of trees in $\mathcal{T}$ restricted to L. For a set $F \subseteq \hat{E}(T)$ we use T/F to denote the tree obtained from T by contracting the edges of F. For a set $F \subseteq \hat{E}(\mathcal{T})$, we denote the set $\{T_1/F, ..., T_k/F\}$ by $\mathcal{T}/F$. Given two trees S and T where $L(T) \subseteq L(S)$, T is an *induced subtree* of S if and only if $S|L(T) = T$. Note that all degree two vertices in $S|L(T)$ other than the root are assumed to be suppressed. An *agreement supertree* for $\mathcal{T}$ is a tree S such that $L(S) = L(\mathcal{T})$, and each T_i is an induced subtree of S. We say that the trees in $\mathcal{T}$ *agree* if and only if there is an agreement supertree for $\mathcal{T}$.

3.3 Solving the Agreement Supertree Problem

The main result of this section, presented in Section 3.3.3, is an $O(kn^2)$ time algorithm, called TestAgreement, that determines whether or not a collection $\mathcal{T}$ of phylogenetic trees has an agreement supertree. The algorithm relies on a recursive characterization of agreement, based on an intersection graph that we define in Section 3.3.1. In Section 3.3.2, we use this graph to develop an algorithm GetSuccessors that decomposes an agreement problem into smaller subproblems, or reports that no such decomposition is possible. In the latter case, GetSuccessors returns a small set of internal nodes from the input collection $\mathcal{T}$ that, in a sense, obstruct agreement. GetSuccessors is thus at the core of TestAgreement, and the information it produces is essential for our algorithms for the AST-EC and AST-TR problems.

3.3.1 An auxiliary graph

A *position* π in $\mathcal{T}$ is a tuple $(v_1, v_2, \ldots, v_k)$ where each v_i is either a vertex from the tree T_i or the symbol $\perp$. A *reduced position* is a position where each component is an internal node or $\perp$. The *reduction of a position* π, denoted by $\pi\downarrow$, is derived by substituting every leaf vertex in π by $\perp$. We use $\pi_\top$, respectively $\pi_\perp$, to denote the *initial*, respectively *final*, positions where $v_i = r(T_i)$, respectively $v_i = \perp$, for each $i \in [k]$. We write $L(\pi)$ for $\bigcup_{i\in[k]} L(\pi[i])$. By an agreement supertree for π, we mean an agreement supertree for the collection of trees $\{T_1(\pi[1]), \ldots, T_k(\pi[k])\}$.

We now introduce an auxiliary graph $G(\mathcal{T}, \pi)$, defined in [34], which is useful for identifying an agreement supertree for the position π. We will look for a specific partition of this graph, called a *nice partition*, that allows us to break the problem into smaller subproblems, or to conclude that there is no solution. The vertex set of $G(\mathcal{T}, \pi)$ consists of the children of all the vertices in π, and there is an edge between two vertices u and v if and only if $L(u) \cap L(v) \neq \emptyset$. Note that the graph $G(\mathcal{T}, \pi)$ is only defined when π is a reduced position. In the rest of this paper, $G(\mathcal{T}, \pi)$ is denoted by $G = (V, E)$ and $V_i = \mathrm{Ch}(\pi[i])$ for each $i \in [k]$.

A subset $U \subseteq V$ is *nice* if, for each $i \in [k]$, U contains either zero, one, or all of the elements of V_i. A partition P of V is a *nice partition* of G if every set of P is nice, and, for every

$\{C, C'\} \subseteq P$, C and C' are disconnected in G. The *successor of* π *w.r.t. a nice set* U, denoted by π_U, is defined as:

$$\pi_U[i] = \begin{cases} \bot & \text{if } V_i \cap U = \emptyset \\ p & \text{if } V_i \cap U = \{p\} \\ \pi[i] & \text{if } |V_i \cap U| \geq 2 \end{cases}$$

for each $i \in [k]$. We write $L(U)$ for $\bigcup_{v \in U} L(v)$. We have the following relationship between $L(U)$ and $L(\pi_U)$ which follows from the definition of a successor position.

Lemma 9. *Let $U \subseteq V$ be a nice set, then $L(U) = L(\pi_U)$.*

In the following, we will focus on a specific nice partition of G that is minimal in a certain sense. For partitions P and Q of V, we say P is *finer* than Q, denoted $P \sqsubseteq Q$, if and only if, for every $C \in P$, there exists a $D \in Q$ such that $C \subseteq D$. Let $\mathcal{P}$ represent the set of all nice partitions of G, and let $(\mathcal{P}, \sqsubseteq)$ represent the poset formed by partitions of $\mathcal{P}$ ordered under $\sqsubseteq$.

Lemma 10. $(\mathcal{P}, \sqsubseteq)$ *has a unique minimal element.*

Proof. Assume, towards a contradiction, that P and Q are distinct minimal elements of $(\mathcal{P}, \sqsubseteq)$. Consider the set $P \sqcap Q$ defined as:

$$P \sqcap Q = \{C \cap D : C \in P, D \in Q\} \setminus \{\emptyset\}. \tag{3.1}$$

It is known that $P \sqcap Q$ is also a partition of V, s.t. $P \sqcap Q \sqsubset P$. We show that $P \sqcap Q$ is also in $\mathcal{P}$, which will contradict the minimality of P, Q.

Consider any $X \in P \sqcap Q$, and let us show that X is a nice set. We have $X = C \cap D$ for some $C \in P, D \in Q$. Fix $i \in [k]$, and assume that $|X \cap V_i| \geq 2$. Since C and D are nice sets, $V_i \subseteq C$ and $V_i \subseteq D$, and thus, $V_i \subseteq X$. As this holds for any $i \in [k]$, we conclude that X is a nice set.

Consider two distinct classes $X, X' \in P \sqcap Q$, and let us show that they are disconnected in G. We have $X = C \cap D$ for some $C \in P, D \in Q$, and $X' = C' \cap D'$ for some $C' \in P, D' \in Q$. As $X \neq X'$, we have either $C \neq C'$ or $D \neq D'$. In the first case, C, C' are disconnected, and in the second case D, D' are. We conclude that X, X' are disconnected. Hence, $P \sqcap Q$ is a nice partition of V. $\square$

We call the unique minimal element of $(\mathcal{P}, \sqsubseteq)$ the *minimum nice partition* of G. Suppose that the minimum nice partition P of G is a singleton. Let $K = \{i \in [k] : \pi[i] \neq \perp\}$. We say that a set $I \subseteq V$ is *interesting* for a reduced position π of $\mathcal{T}$ if both $|I \cap V_i| = 2$ for each $i \in K$, and there is a set $F \subseteq E$ such that all of the following conditions hold: (i) $|F| \leq 2|K| - 1$; (ii) for each $v \in I$, there exists an $e \in F$ such that $v \in e$; and (iii) the subgraph of G induced by F has a minimum nice partition that is a singleton. Intuitively, a set of interesting vertices certifies that the minimum nice partition of G has a unique class. As we will see in Section 3.3.3, this condition will guarantee that there is no agreement supertree for π. In this case, we say that π is an *obstructing position* for $\mathcal{T}$.

3.3.2 Finding successor positions and interesting vertices

We now present algorithm GetSuccessors, which takes as input a position π in a collection $\mathcal{T}$ of rooted phylogenetic trees, and finds the set Π of successor positions for each class in the minimum nice partition of G (see Algorithm 1). When the minimum nice partition of G is a singleton, the algorithm returns a set I of interesting vertices for π.

The central idea behind the GetSuccessors algorithm is that, for a given $\ell \in L(\pi)$, all of the vertices in the set $S_\ell = \{v \in V : \ell \in L(v)\}$ are connected, and hence, must be in the same class of the minimum nice partition. The algorithm builds the successor positions by iterating over each label ℓ and building a position for ℓ which is denoted by π_ℓ, by examining each vertex $v \in S_\ell$. If v is already covered by a position π', then π' will need to be merged with π_ℓ. Once this merge is completed, the position π' is no longer needed. For implementation efficiency, instead of deleting positions, we keep a flag $\mathsf{active}(\pi_\ell)$ for each position π_ℓ. If $\mathsf{active}(\pi_\ell)$ is set to true, then π_ℓ is one of the successor positions of the graph G restricted to only those labels which have already been processed by the algorithm. If $\mathsf{active}(\pi_\ell)$ is set to false, then it is no longer used by the algorithm. If v is not already covered by some position, then we simply add v to π_ℓ.

After merging a position π' with π_ℓ, it may be the case that π_ℓ contains multiple vertices from some input tree T. In such a case, the algorithm needs to merge with π_ℓ all of the positions covering any of the vertices from T. Furthermore, since each $\ell \in L(\pi)$ can be in the subtree of

Algorithm 1: GetSuccessors$(\mathcal{T}, \pi)$

Input: A position π in a collection $\mathcal{T}$ trees.

Output: A tuple (Π, I) where Π is the set of successor positions of each class in the minimum nice partition of G, and when Π is a singleton, I is a set of interesting vertices for the unique $\pi \in \Pi$.

```
1  foreach ℓ ∈ L(π) do S_ℓ ← ∅
2  foreach i ∈ [k] do
3  |  position(π[i]) ← ∅
4  |  foreach v ∈ V_i do
5  |  |  position(v) ← ∅
6  |  |  foreach ℓ ∈ L(v) do S_ℓ ← S_ℓ ∪ {v}
7  Π ← ∅ ; I ← ∅
8  foreach ℓ ∈ L(π) do
9  |  π_ℓ ← π_⊥ ; Z ← ∅
10 |  foreach v ∈ S_ℓ do
11 |  |  if position(π[tree(v)]) ≠ ∅ then Z ← Z ∪ position(π[tree(v)])
12 |  |  else if position(v) ≠ ∅ then Z ← Z ∪ position(v)
13 |  |  else π_ℓ[tree(v)] ← v ; position(v) ← {π_ℓ}
14 |  while there is a π_p ∈ Z with active(π_p) = true do
15 |  |  active(π_p) ← false
16 |  |  foreach i ∈ [k] such that π_ℓ[i] ≠ π[i] and π_p[i] ≠ ⊥ do
17 |  |  |  if π_ℓ[i] = ⊥ then  π_ℓ[i] ← π_p[i] ; position(π_p[i]) ← {π_ℓ}
18 |  |  |  else
19 |  |  |  |  I ← I ∪ {π_ℓ[i], π_p[i]} ; π_ℓ[i] ← π[i]
20 |  |  |  |  position(π[i]) ← {π_ℓ}
21 |  |  |  |  foreach w ∈ V_i do Z ← Z ∪ position(w)
22 |  active(π_ℓ) ← true
23 |  Π ← Π ∪ {π_ℓ}
24 return ({π_ℓ ∈ Π : active(π_ℓ) = true}, I)
```

at most one v per V_i, it follows that the first time two vertices from the same input tree end up in the same partition, those two vertices are unique and we add them to the set I of interesting vertices.

If GetSuccessors determines that the minimum nice partition of G is a singleton, then the set Π returned has π as its only element. In this case, the set I of vertices returned by the algorithm is a set of interesting vertices for π.

For each vertex v that is either an element of position π or the child of a vertex in π, the algorithm keeps a reference $\mathsf{position}(v)$ that points to a set containing the active position containing v. This is determined in the following manner:

- if v is an element of the position π, then $\mathsf{position}(v)$ points to an active position that contains all of the children of v; and
- if v is a child of a vertex in π, then $\mathsf{position}(v)$ points to an active position that contains v.

Also, note that the set $\mathsf{position}(v)$ is pointing to is always either the empty set or a singleton. The purpose for using sets for $\mathsf{position}(v)$ is to simplify the code of lines 11, 12 and 21.

In the initialization phase of the algorithm (lines 1-6), for each label $\ell \in L(\mathcal{T})$ we construct a set S_ℓ that contains all of the vertices $v \in V(\mathcal{T})$ such that both of the following conditions hold:

(i) $v \in V_i$ for some $i \in [k]$, i.e., v is a child of some vertex in position π; and

(ii) $\ell \in L(v)$, i.e., ℓ is a label of the subtree rooted at v.

Also, since at the initialization phase, there are no active positions, the $\mathsf{position}$ references are all set to $\emptyset$.

The algorithm then turns to the construction phase (lines 7-23), where the positions π_ℓ are constructed. Note that for each vertex $v \in V(\mathcal{T})$, the algorithm also uses a function $\mathsf{tree}(v)$ that returns the unique index i such that $v \in V(T_i)$. The algorithm maintains two sets. Set Π will hold all of the successor positions created, and set I will hold the interesting vertices. The position π_ℓ is built in two phases. Naturally, each position π_ℓ is going to hold all those vertices of G that contain ℓ in their subtree, and these are precisely the vertices in the set S_ℓ. Thus, in the first phase (the loop in lines 10-13) the algorithm iterates over the elements of S_ℓ to ensure that they are included in the position. While doing so, the algorithm may discover new positions that need to be merged with π_ℓ and it stores these positions in a buffer Z. Now, for each $v \in S_\ell$ it performs the following tests in the specified order:

1. If $\mathsf{position}(\pi[\mathsf{tree}(v)])$ is non-empty, then it points to some position π' that contains all of the elements of V_i. Since $v \in V_i$, π' also contains v. Hence π' needs to be merged into π_ℓ, and so π' is added to Z.

2. If $\mathsf{position}(\pi[\mathsf{tree}(v)])$ is empty, but $\mathsf{position}(v)$ is non-empty, then $\mathsf{position}(v)$ points to some other position π' containing v, and hence π' needs to be merged into π_ℓ. Thus, π' is added to Z.

3. If both $\mathsf{position}(\pi[\mathsf{tree}(v)])$ and $\mathsf{position}(\pi[\mathsf{tree}(v)])$ are empty, then no position currently contains v, so we add v to π_ℓ.

Note that if tests 1 or 2 succeed, we do not immediately add v to position π_ℓ. That is, we add some position π' to Z, and so eventually π' will be merged with π_ℓ. After this happens, π_ℓ will contain v.

After processing the set S_ℓ, it may be the case that there are active components in Z. These components need to be merged with π_ℓ. This is done in the merge phase of the algorithm (lines 14-21). Let π_p be the position being merged with π_ℓ. Since π_p is being merged with π_ℓ, π_p will no longer represent a successor position, so the algorithm first sets $\mathsf{active}(\pi_p)$ to false. We now compare positions π_ℓ and π_p index by index. If $\pi_\ell[i] = \pi[i]$, then π_ℓ already contains all of the children of V_i, so no work needs to be done for index i. If $\pi_p[i] = \perp$, then no new vertices need to be added to π_ℓ for index i. Otherwise, either $\pi_\ell[i] = \perp$ or $\pi_\ell[i] \neq \perp$

Case 1. If $\pi_\ell[i] = \perp$, then either $\pi_p[i]$ is a single element of V_i, or $\pi_p[i] = \pi[i]$. In either case, to merge the two positions, we only need to copy the value of $\pi_p[i]$ to $\pi_\ell[i]$. Then, we need to update the $\mathsf{position}(\pi_p[i])$ to now refer to π_ℓ instead of π_p.

Case 2. If $\pi_\ell[i] \neq \perp$, then it must be the case that both $\pi_\ell[i]$ and $\pi_p[i]$ each contain an element of V_i and that these vertices are different. Thus, $\pi_\ell[i]$ now contains two elements of V_i and hence must contain all elements of V_i, so we set $\mathsf{position}(\pi[i])$ to π_ℓ, add the two vertices in $\pi_\ell[i]$ and $\pi_p[i]$ to the set of interesting vertices, and add any positions containing a child of V_i to Z, as they now also need to be merged with π_ℓ.

Lines 22 and 23 complete the process of constructing π_ℓ. This is done by first setting $\mathsf{active}(\pi_\ell)$ to true since it represents a successor position of G restricted to the labels processed by the algorithm so far. It then adds π_ℓ to the set Π containing all positions constructed so far.

The algorithm finishes in line 24, after all labels have been processed, by returning the set

of positions that remain active. We later formally prove (Theorem 19) that these are indeed the successor positions of G. The algorithm also returns the set I of vertices collected during the execution of the algorithm. If there is more than one active position in Π, then the set I of vertices has no meaningful value to us. However, as we show later (Theorem 20), if there is only one active position in Π, I is indeed a set of interesting vertices for π. The next two theorems summarize the essential properties of algorithm GetSuccessors. For technical reasons, their proofs are deferred to Section 3.5.1

Theorem 19. GetSuccessors *can be implemented to run in* $O(kn)$ *time, and in the tuple* (Π, I) *returned,* Π *is exactly the successor positions of each class of the minimum nice partition* P *of* G.

Theorem 20. *If the set* Π *returned by* GetSuccessors *is a singleton with* $\Pi = \{\pi\}$*, then* I *is a set of interesting vertices for* π.

3.3.3 Testing for an agreement supertree

We now present algorithm TestAgreement, which takes a position π in a collection $\mathcal{T}$ of phylogenetic trees, and decides if there is an agreement supertree for π (see Algorithm 2). If there is no agreement supertree for π, the algorithm returns an obstructing position π' and a set of interesting vertices for π'. To test for the existence of an agreement supertree for $\mathcal{T}$, it suffices to call TestAgreement on the initial position $\pi_\top$. The set of interesting vertices returned by TestAgreement will be used in the remainder of the algorithms discussed in this paper.

The algorithm TestAgreement proceeds in a recursive top-down fashion. If the position π is not reduced, it considers instead the reduced position $\pi\downarrow$, as justified by Lemma 11 below. Then, it calls GetSuccessors to compute the set Π of successor positions corresponding to the minimum nice partition of $G(\mathcal{T}, \pi)$. If Π has a single class, the algorithm concludes that there is no agreement supertree for π. Otherwise, it recursively checks for agreement on the successor positions $\pi' \in \Pi$. The correctness of this step follows from Lemma 13 below.

Lemma 11. *The following statements hold:*

Algorithm 2: TestAgreement$(\mathcal{T}, \pi)$

Input: A position π in a collection $\mathcal{T}$ of trees.

Output: A tuple (B, π', I) where B is a boolean indicating whether there is an agreement supertree for π, and, when B is no, π' is an obstructing position and I is a set of interesting vertices for π'.

```
1 π ← π↓
2 if π = π_⊥ then return (yes, ∅, ∅)
3 (Π, I) ← GetSuccessors(T, π)
4 if |Π| = 1 then return (no, π, I)
5 foreach π' ∈ Π do
6 |   (B, π'', I) ← TestAgreement(T, π')
7 |   if B = no then return (no, π'', I)
8 return (yes, ∅, ∅)
```

1. *There is an agreement supertree for $\mathcal{T}$ if and only if there is an agreement supertree for every position π of $\mathcal{T}$.*

2. *There is an agreement supertree for a position π of $\mathcal{T}$ if and only if there is an agreement supertree for $\pi\downarrow$.*

Proof. Statement 1 holds because ($\Rightarrow$) for any agreement supertree S of $\mathcal{T}$ and any position π of $\mathcal{T}$, $S|L(\pi)$ is an agreement supertree for π; and ($\Leftarrow$) any agreement supertree for $\pi_\top$ is an agreement supertree for $\mathcal{T}$. Statement 2 holds since ($\Rightarrow$) $L(\pi\downarrow) \subseteq L(\pi)$, so for any supertree S for π, $S|L(\pi\downarrow)$ is a supertree for $\pi\downarrow$; and ($\Leftarrow$) for each $i \in [k]$ for which $\pi\downarrow[i] \neq \pi[i]$, we have that $\pi[i]$ is a leaf whose label can be added to a supertree for $\pi\downarrow$ (that is, we simply add each such leaf as child of the root, to get a supertree for π). □

We will need the following characterization of induced subtrees in terms of embeddings. This lemma follows from the definitions and is stated without proof.

Lemma 12. *Let S and T be two phylogenetic trees where $L(T) \subseteq L(S)$. The following statements are equivalent*

1. *T is an induced subtree of S.*

2. *There exists an injective function $\phi : V(T) \to V(S)$ with the following properties.*

   (a) *For every $p \in \mathcal{L}(T)$, $\phi(p)$ is a leaf of S with the same label.*

(b) For every $p \in \mathcal{I}(T)$, if $q \in \mathrm{Ch}(p)$, then $\phi(q) <_S \phi(p)$. Also, for every $\{u, v\} \subseteq \mathrm{Ch}(p)$, $\mathrm{child}_S(\phi(p), \phi(u)) \neq \mathrm{child}_S(\phi(p), \phi(v))$.

We call ϕ an embedding of T into S.

The following lemma is the central result on which our recursive algorithm is based. Note that we only have to consider a reduced position π, according to Lemma 11.

Lemma 13. *Let π be a reduced position such that $\pi \neq \pi_\perp$. The following statements are equivalent.*

1. *There is an agreement supertree for π.*
2. *There exists a nice partition P of G where P has at least two classes, and, for every $X \in P$, π_X has an agreement supertree.*

Proof. ($\Longrightarrow$) Suppose that π has an agreement supertree S with $L(S) = L(\pi)$. By Lemma 12, for every $i \in [k]$ there exists an embedding ϕ_i of $T_i(\pi[i])$ into S. Let $r = r(S)$. As π is a reduced position different from $\pi_\perp$, we have $|L(S)| = |L(\pi)| \geq 2$, and thus $\mathrm{Ch}(r)$ is not empty. Let $\mathrm{Ch}(r) = \{u_1, \ldots, u_m\}$. We build a partition $P = \{C_1, \ldots, C_m\}$ of V as follows. For every $v \in V_i$ and $i \in [k]$, if $\phi_i(v) \leq_S u_{i'}$ for some $i' \in [m]$, then add v to $C_{i'}$. Note that there will always exist one such $u_{i'}$. We now show that P is a nice partition with at least two classes, and that for every $X \in P$, π_X has an agreement supertree.

As S is a phylogenetic tree, we have $m \geq 2$ and thus $|P| \geq 2$. Let us now show that P is a nice partition of G. Fix $i \in [k]$, and let $r_i = \pi[i]$. If $\phi_i(r_i) = r$, then by definition of an embedding the nodes $\phi_i(u)$ ($u \in V_i$) belong to distinct classes of P. On the other hand, if $\phi_i(r_i) \leq_S u_j$, then the nodes $\phi_i(u)$ ($u \in V_i$) all belong to C_j. Thus, every class of P is nice. Consider any $u \in C_i$ and $v \in C_j$ where $i \neq j$, with $u \in V_a$ and $v \in V_b$. Since $L(u_i) \cap L(u_j) = \emptyset$, $\phi_a(u) \leq_S u_i$ and $\phi_b(v) \leq_S u_j$, then $L(u) \cap L(v) = \emptyset$. Thus, vertices of C_i and C_j will be disconnected in G and P is a nice partition of G.

Lastly, for any $i \in [m]$, we show that $S(u_i)$ is an agreement supertree for π_{C_i}. Observe that $L(S(u_i)) = L(C_i)$, which is equal to $L(\pi_{C_i})$ by Lemma 9. For any $j \in [k]$, we define an embedding ϕ'_j from $T_j(\pi_{C_i}[j])$ to $S(u_i)$ as follows. For every $v \in V(T_j(\pi_{C_i}[j]))$, set $\phi'_j(v) =$

$\phi_j(v)$. Since C_i is a nice set and ϕ_j is an embedding, ϕ'_j also satisfies all the properties of an embedding. Thus, $T_j(\pi_{C_i}[j])$ is an induced subtree of $S(u_i)$.

($\Longleftarrow$) Let $P = \{C_1, \ldots, C_m\}$ and let S_i represent the agreement supertree for π_{C_i} for every $i \in [m]$, such that $L(S_i) = L(\pi_{C_i})$. We build an agreement supertree S for π as follows. Add an edge from a vertex r to $r(S_i)$ for every $i \in [m]$. Set r to be the root of S. For any $1 \leq i \neq j \leq m$, the sets C_i and C_j are disconnected in G. Thus, $L(C_i) \cap L(C_j) = \emptyset$, and by Lemma 9 it follows that $L(S_i) \cap L(S_j) = \emptyset$. Furthermore, since P is a partition of V, we have $\bigcup_i L(C_i) = L(V)$. By Lemma 9, we deduce that $\bigcup_i L(S_i) = L(\pi)$. Thus, S is a phylogenetic tree with label set $L(\pi)$. We will prove that S is an agreement supertree for π by showing that for any $i \in [k]$, $T_i(\pi[i])$ is an induced subtree of S. We have the following two cases for vertex $\pi[i]$.

Case 1: $\pi[i] = \pi_{C_j}[i]$ for some $j \in [m]$. As S_j is an agreement supertree for π_{C_j}, it follows that the tree $T_i(\pi[i])$ is an induced subtree of S_j. Since S_j is an induced subtree of S, $T_i(\pi[i])$ is also an induced subtree of S.

Case 2: $\pi[i] \neq \pi_{C_j}[i]$ for every $j \in [m]$. We build an embedding ϕ_i of $T_i(\pi[i])$ into S as follows. By Lemma 12, there exists an embedding $\phi_{p,q}$ from $T_p(\pi_{C_q}[p])$ to S_q for every $p \in [k]$ and $q \in [m]$. For every $v \in V(T_i(\pi[i]))$, set $\phi_i(v) = \phi_{i,j}(v)$ if $v \leq_{T_i} \pi_{C_j}[i]$. Now, set $\phi_i(\pi[i]) = r$. Let us show that ϕ_i is an embedding of $T_i(\pi[i])$ into S. First, observe that ϕ_i satisfies Conditions (a)-(b) of Lemma 12 for any $v <_{T_i} \pi[i]$. Indeed, for such a v we have $v \leq_{T_i} \pi_{C_j}[i]$ for some $j \in [m]$, and Conditions (a)-(b) hold for $\phi_{i,j}$ as it is an embedding of $T_i(\pi_{C_j}[i])$ into S_j. It remains to verify Condition (b) of Lemma 12 for $v = \pi[i]$. For every u child of v, we have $\phi_i(u) = r(S_j)$ for some $j \in [m]$, and thus $\phi_i(u) <_S \phi_i(v)$. Moreover, given two children u, u' of v, as they belong to different classes of P, $\text{child}_S(\phi_i(v), \phi_i(u)) \neq \text{child}_S(\phi_i(v), \phi_i(u'))$. Thus, ϕ_i is an embedding of $T_i(\pi[i])$ into S and hence, $T_i(\pi[i])$ is an induced subtree of S. □

Theorem 21 states the runtime and correctness of the TestAgreement algorithm (Algorithm 2).

Theorem 21. TestAgreement *can be implemented to run in* $O(kn^2)$ *time and correctly decides if there is an agreement supertree for a position* π *in* $\mathcal{T}$*. If there is no agreement supertree for* π*, it returns an obstructing position* π' *and a set* I *of interesting vertices for* π'*.*

Proof. We first justify the running time of the algorithm. Since at each execution of a recursive call, the label sets in each position returned by GetSuccessors are disjoint, it follows that the recursion tree has $O(n)$ leaves. So there are $O(n)$ recursive calls to TestAgreement. Since each execution of the loop in line 5 results in a recursive call, and the body of the loop takes $O(1)$ time outside of the recursive call, it follows that the algorithm spends, over all recursive calls, a total of $O(n)$ time executing lines 5-7. Thus, it suffices to show that each recursive call spends at most $O(kn)$ time outside of lines 5-7. Clearly, lines 1 and 2 can be done in $O(k)$ time. The call to GetSuccessors takes $O(kn)$ time by Theorem 19. Line 4 and 8 can clearly be done in $O(1)$ time.

We now argue for the correctness of the algorithm. We show by induction on the height of the recursion tree, that TestAgreement$(\mathcal{T}, \pi)$ correctly decides if π has an agreement supertree, and, in case there is no agreement supertree for π, the position π'' and set I returned on line 7 are an obstructing position for $\mathcal{T}$ and a set of interesting vertices for π''. Let P be the minimum nice partition of G.

There are two base cases: (i) when $\pi = \pi_\perp$; and (ii) when P has a single class. In case (i) there is an agreement supertree for π and the algorithm returns "yes" on line 2. In case (ii), by Lemma 13, there is no agreement supertree for π. By Theorem 19, the set Π returned in line 3 will be a singleton and π is an obstructing position. By Theorem 20, I is a set of interesting vertices for π. Line 4 returns π along with the set of interesting vertices returned by the call to GetSuccessors.

Now suppose that P has more than one class and Π is the set of successor positions returned by GetSuccessors. Then by induction hypothesis, for each $\pi' \in \Pi$, TestAgreement correctly decides whether there is an agreement supertree for π'. If there is an agreement supertree for each position in Π, then by Lemma 13, there is an agreement supertree for π and the algorithm returns "yes" on line 8. If there is no agreement supertree for some position $\pi' \in \Pi$, then by the inductive hypothesis, TestAgreement$(\mathcal{T}, \pi')$ will answer in the negative, and also return an obstructing position π'' and a set of interesting vertices for π''. By Lemma 11, π'' is an obstructing position for $\mathcal{T}$, so the algorithm returns π'' along with the set I of interesting vertices returned by GetSuccessors. □

3.4 Solving the AST-EC and AST-TR problems

In this section we show that both the AST-EC and AST-TR problems are fixed-parameter tractable for parameters k and p. Our two algorithms build upon the results of Section 3.3, in the sense that we use the interesting vertices found by TestAgreeement to identify small obstructions. These obstructions allow us to solve the problems by a bounded search tree approach, giving rise to FPT algorithms with running time $O((2k)^p kn^2)$. This section is organized as follows. Section 3.4.1 presents an auxiliary algorithm, called Merge, on which we will rely to construct and prove the correctness of the obstructions for both problems. In Section 3.4.2, we prove that AST-EC is NP-complete and we give an FPT algorithm for the problem. In Section 3.4.3, we give an FPT algorithm for AST-TR.

3.4.1 An auxiliary algorithm

We define the *closure* of a set $C \subseteq V$ as the set $\langle C \rangle_G \subseteq V$ such that: (i) if $C \cap V_i = \emptyset$ then $\langle C \rangle_G \cap V_i = \emptyset$; (ii) if $C \cap V_i = \{v\}$, then $\langle C \rangle_G \cap V_i = \{v\}$; and (iii) if $|C \cap V_i| \geq 2$ then $\langle C \rangle_G \cap V_i = V_i$. Our auxiliary algorithm, called Algorithm Merge, takes as input G and a set $I \subseteq V$, and proceeds as follows. We maintain a partition P of I. Initially P contains a class $\{v\}$ for each $v \in I$. At a given step, suppose that $P = \{C_1, \ldots, C_p\}$. An edge of G is a *transverse edge* if it connects two nodes in the closures of two different sets in P. If G contains a transverse edge joining $\langle C_i \rangle_G$ to $\langle C_j \rangle_G$ for some i, j, then we replace P by $P \backslash \{C_i, C_j\} \cup \{C_i \cup C_j\}$, and we continue.

Lemma 14. *Let Q be the minimum nice partition of G. At a given step of Algorithm* Merge*, it holds that: for each $C \in P$, there is a $K \in Q$ such that $\langle C \rangle_G \subseteq K$.*

Proof. By induction on the number of steps executed by the algorithm. This is clear initially. Suppose that this holds at the beginning of the pth step, and let us show that this holds at the end of the pth step. Suppose that this step replaces P by $P' = P \backslash \{C_i, C_j\} \cup \{C_i \cup C_j\}$. By induction hypothesis, there exist $A, B \in Q$ such that $\langle C_i \rangle_G \subseteq A$ and $\langle C_j \rangle_G \subseteq B$. By assumption, G contains a transverse edge between $\langle C_i \rangle_G$ and $\langle C_j \rangle_G$. As two classes of Q are

disconnected in G, it follows that $A = B$, and thus $C_i \cup C_j \subseteq A$. As A is a nice set, we obtain that $\langle C_i \cup C_j \rangle_G \subseteq A$. □

The crucial property of Algorithm MERGE is that, by starting with the interesting vertices, it will end with P consisting of a single class. This property is stated in the following Lemma, whose proof is given in Section 3.5.2. The proof relies on an alternative formulation of the GETSUCCESSORS algorithm, and on the notion of *merge forest* representing the sequence of merges performed by the algorithm.

Lemma 15. *Suppose that* TESTAGREEMENT$(\mathcal{T}, \pi_\top)$ *has returned* (no, π, I)*. Then Algorithm* MERGE *run on* G, I *ends with the partition* $\{I\}$*.*

Note that this fact certifies that the minimum nice partition Q of G is a singleton: by Lemma 14, it follows that $\langle I \rangle_G = V$ is included in a same class of Q. To check that Q has a unique class, we can thus use I as the certificate, and MERGE as the verification algorithm. We will repeatedly use this fact in the following sections. Additionally, for settings where we actually want to use MERGE for verification purposes, we will give a $O(kn)$ implementation in Section 3.4.3 under the name FINDOBSTRUCTION.

3.4.2 Solving the AST-EC problem

The computational complexity of AST-EC does not seem to have been studied before. To motivate the development of a fixed-parameter algorithm to solve the problem, we first prove the problem is NP-complete.

We use a recursive parenthesized notation for trees: if ℓ is a label, ℓ represents a tree with a single leaf labelled ℓ; if $T_1, \ldots, T_k$ are trees, then $(T_1, \ldots, T_k)$ represents the tree whose root is unlabelled and has $T_1, \ldots, T_k$ as child subtrees.

Theorem 22. AST-EC *is* NP*-complete.*

Proof. Membership in NP is clear. The NP-hardness is shown by a reduction from the MULTICUT problem, which is defined as follows. Given a graph $G = (V, E)$, a set of requests $R \subseteq V \times V$, and an integer p, the MULTICUT problem asks if there exists a set S of at most p edges in E, where, for every $uv \in R$, u and v are in different components of $G \backslash S$.

Given an instance $I = (G, R, p)$ of MULTICUT, we construct an instance $I' = (\mathcal{T}, p)$ of AST-EC as follows. The collection $\mathcal{T}$ is defined over $L := V \cup \{x\}$ and consists of: (i) for each edge $e = uv \in E$, a tree $t_e = ((u, v), x)$; (ii) for each pair $p = uv \in R$, a tree $f_p = (u, v, x)$. For each $e \in E$, we let $\gamma(e)$ denote the internal edge of t_e, we let $\alpha(e)$ denote the parent of u, v in t_e, and we let $\beta(e)$ denote the parent of $\alpha(e), x$ in t_e. The notation $\gamma(e)$ induces a bijection $\gamma : E \to \hat{E}(\mathcal{T})$; we will use the notation $\gamma(S)$ and $\gamma^{-1}(S)$ to denote, respectively, the image and the inverse image of a set S under γ.

The reduction is clearly doable in polynomial time. To prove its correctness, we show that: I is a positive instance of MULTICUT iff I' is a positive instance of AST-EC.

Suppose that (G, R) has a minimum multicut $S \subseteq E$ with $|S| \leq p$. Let $S' = \gamma(S)$, we show that $\mathcal{T}/S'$ has an agreement supertree. Let $C_1, \ldots, C_m$ denote the connected components of $G \backslash S$. By minimality of S, each edge of S is between two distinct components of G. For every $1 \leq i \leq m$, let T_i denote a star-tree with leaves labeled by C_i, and let $T = (T_1, \ldots, T_m, x)$. We show that T is an agreement supertree of $\mathcal{T}/S'$. First, for each $e = uv \in E \backslash S$, we have u, v in a same connected component of $G \backslash S$, and thus $t_e/S' = ((u, v), x)$ is a subtree of T. Second, for each $e = uv \in S$, we have u, v in different connected components of $G \backslash S$, and thus $t_e/S' = (u, v, x)$ is a subtree of T. Third, for each $p = uv \in R$, we have u, v in different connected components of $G \backslash S$, and thus $f_e/S' = (u, v, x)$ is a subtree of T. We conclude that T is an agreement supertree of $\mathcal{T}/S'$.

Conversely, suppose that there exists $S \subseteq \hat{E}(\mathcal{T})$ such that $|S| \leq p$ and $\mathcal{T}/S$ has an agreement supertree T. Let $S' = \gamma^{-1}(S)$, we show that S' is a multicut of (G, R). Suppose by contradiction that $G \backslash S'$ contains a path P between two endpoints of a request $uv \in R$. Then $P = u_0 e_1 u_1 \ldots u_{r-1} e_r u_r$ with $u = u_0, v = u_r$. It follows from Lemma 12 that for every $1 \leq i \leq r$, there is an embedding ϕ_i of t_{e_i} into T; let $a_i = \phi_i(\alpha(e_i))$ and $b_i = \phi_i(\beta(e_i))$. Since ϕ_i is an embedding, the nodes $\text{child}_T(b_i, a_i)$ and $\text{child}_T(b_i, x)$ are distinct; let us denote them by p_i, q_i. As u_i occurs in t_{e_i} and $t_{e_{i+1}}$, it follows that $b_{i+1} = b_i, p_{i+1} = p_i, q_{i+1} = q_i$. Denote these nodes by b, p, q. We then have $u_0 <_T p$, $u_r <_T p$ and $x \leq_T q$, which implies that $((u_0, u_r), x)$ is a subtree of T. But by definition of T, it contains $f_{uv} = (u, v, x)$ as a subtree, contradiction. We conclude that there is no such path P, and thus S' is a multicut of (G, R). □

In the remainder of this subsection, we show that AST-EC is fixed-parameter tractable in parameters k and p. Lemma 16 shows that if a call to TestAgreement$(\mathcal{T}, \pi_\top)$ answers negatively, we must contract at least one edge joining an interesting vertex to its parent.

Lemma 16. *Suppose that* TestAgreement$(\mathcal{T}, \pi_\top)$ *has returned a tuple* (no, π, I). *In order to obtain a collection having an agreement supertree, we need to contract one edge* $\{v,\ parent(v)\}$ *with* $v \in I$.

Proof. Assume that we have a set $S \subseteq \hat{E}(\mathcal{T})$ which contains none of the edges $\{v,\ parent(v)\}$ with $v \in I$; we show that $\mathcal{T}/S$ has no agreement supertree. We use the following convention: when contracting an edge $\{u, v\}$ with $v = parent(u)$, the resulting vertex is identified with v. Then, by definition of S, each element of I is still a node of $\mathcal{T}/S$. Define the position π' in $\mathcal{T}/S$ as follows. If $\pi[i] = \perp$, then $\pi'[i] = \perp$. If $\pi[i] = u$ is the parent of two vertices $v, w \in I$, then $\pi'[i]$ is the common parent of v, w in T_i/S. Let $G' = G(\mathcal{T}/S, \pi') = (V', E')$.

Let us consider an execution $\mathcal{E}$ of Algorithm Merge on G and I. We build an execution $\mathcal{E}'$ of Algorithm Merge on G' and I that ends with $\{I\}$ by mirroring each step of $\mathcal{E}$. Let $P_\mathcal{E}, P_{\mathcal{E}'}$ denote the values of P during $\mathcal{E}, \mathcal{E}'$ respectively. We define $\mathcal{E}'$ by induction such that $P_\mathcal{E} = P_{\mathcal{E}'}$ holds at each step. Clearly, this holds at the beginning of $\mathcal{E}, \mathcal{E}'$. Suppose that this holds at the beginning of step s. Then $\mathcal{E}$ picks a transverse edge induced by some label $\ell \in L(\langle C_i \rangle_G) \cap L(\langle C_j \rangle_G)$. The important observation is that given $C \subseteq I$, we have $L(\langle C \rangle_G) \subseteq L(\langle C \rangle_{G'})$ (as $L(\pi[i]) \subseteq L(\pi'[i])$ for every $i \in [k]$). Hence, $\ell \in L(\langle C_i \rangle_{G'}) \cap L(\langle C_j \rangle_{G'})$, and thus $\mathcal{E}'$ can pick a transverse edge joining $\langle C_i \rangle_{G'}$ and $\langle C_j \rangle_{G'}$. This leads to the merge of C_i and C_j, and thus $P_\mathcal{E} = P_{\mathcal{E}'}$ at the end of step s.

Applying the induction hypothesis at the last step of $\mathcal{E}$, and using that $\mathcal{E}$ ends with the partition $\{I\}$ (Lemma 15), we obtain that $\mathcal{E}'$ ends with the partition $\{I\}$. By Lemma 14, if Q is the minimum nice partition of G', we have $\langle I \rangle_{G'} = V'$ in a same component of Q, and thus $\mathcal{T}/S$ has no agreement supertree by Lemmas 11 and 13. □

Since TestAgreement returns a set of at most $2k$ interesting vertices, Lemma 16 leads to an FPT algorithm for AST-EC using a bounded search tree technique.

Theorem 23. AST-EC *can be solved in* $O((2k)^p kn^2)$ *time.*

Proof. We use a recursive algorithm SolveAST-EC$(\mathcal{T}, p)$. The algorithm answers "no" if $p < 0$. Otherwise, it runs TestAgreement$(\mathcal{T}, \pi_\top)$ to decide in $O(kn^2)$ if $\mathcal{T}$ has an agreement supertree. It answers "yes" in case in positive answer. In case of negative answer, it obtains a set I of nodes of $\mathcal{T}$; for each non-leaf vertex $v \in I$, it recursively calls SolveAST-EC$(\mathcal{T}/\{v, parent(v)\}, p-1)$. The algorithm then answers "yes" iff one of the recursive calls does. The correctness of the algorithm follows from Lemma 16, and the running time is $O((2k)^p kn^2)$. □

3.4.3 Solving the AST-TR problem

We will say that a set $C \subseteq L(\mathcal{T})$ is a *conflict among* $\mathcal{T}$ if $\mathcal{T}|C$ has no agreement supertree. Lemma 17 shows that if TestAgreement$(\mathcal{T}, \pi_\top)$ answers negatively, we can obtain a conflict among $\mathcal{T}$ from the set of interesting vertices.

Lemma 17. *Suppose that* TestAgreement$(\mathcal{T}, \pi_\top)$ *has returned a tuple* (no, π, I)*. We can then obtain a set* $C \subseteq L$ *of size at most* $2k-1$ *such that* $\mathcal{T}|C$ *has no agreement supertree.*

Proof. Consider an execution $\mathcal{E}$ of Algorithm Merge on G and I. For each transverse edge $e = uv$ found by $\mathcal{E}$, pick a label $\ell_e \in L(u) \cap L(v)$, and let C be the resulting set of labels. Clearly $|C| \leq 2k-1$. Consider a vertex $v \in I \cap V_i$. Then, during $\mathcal{E}$ consider the first time that $\{v\}$ is merged with another component. This merge corresponds to some label $\ell_v \in L(v) \cap C$. It follows that $\pi[i]$ is still a node of $T_i|C$, let v' denote the child of $\pi[i]$ in $T_i|C$ that contains ℓ_v. Let $I' = \{v' : v \in I\}$, and let $G' = G(\mathcal{T}|C, \pi) = (V', E')$.

We claim that (as in the proof of Lemma 16) the execution $\mathcal{E}$ can be simulated by an execution $\mathcal{E}'$ of Algorithm Merge on G' and I'. Let $P_\mathcal{E}, P_{\mathcal{E}'}$ denote the values of P during $\mathcal{E}, \mathcal{E}'$ respectively. We define $\mathcal{E}'$ by induction such that the following holds at each step: if $P_\mathcal{E} = \{C_1, \ldots, C_p\}$, then $P_{\mathcal{E}'} = \{C'_1, \ldots, C'_p\}$ with $C'_i = \{v' : v \in C_i\}$. Clearly, this holds at the beginning of $\mathcal{E}, \mathcal{E}'$. Suppose that this holds at the beginning of step s. Then $\mathcal{E}$ picks a transverse edge $e = uv$ with $u \in \langle C_i \rangle_G, v \in \langle C_j \rangle_G$. Suppose that $u \in V(T_p)$ and $v \in V(T_q)$. In $T_p|C$ (resp. $T_q|C$), there is a child u' of $\pi[p]$ (resp. a child v' of $\pi[q]$) that contains ℓ_e. The induction hypothesis implies that $u' \in \langle C'_i \rangle_{G'}$ and $v' \in \langle C'_j \rangle_{G'}$, thus $\mathcal{E}'$ can pick the transverse edge $e' = u'v'$ induced by label ℓ_e. This leads to merge C'_i and C'_j and thus the induction hypothesis holds at

the end of step s.

Applying the induction hypothesis at the last step of $\mathcal{E}$, and since $\mathcal{E}$ ends with the partition $\{I\}$ (Lemma 15), we conclude that $\mathcal{E}'$ ends with the partition $\{I'\}$. By Lemma 14, if Q is the minimum nice partition of G', we have $\langle I' \rangle_{G'} = V'$ in a same component of Q, and thus $\mathcal{T}|C$ has no agreement supertree by Lemmas 11 and 13. □

We outline an algorithm FindObstruction that takes as input a set of interesting vertices and returns a conflict among $\mathcal{T}$ of size at most $2k-1$. Suppose that $I = \{v_1, \ldots, v_r\}$. We initialize components $C_1, \ldots, C_r$ with $C_i = \{v_i\}$, and we let $J = \{1, \ldots, r\}$. We use a main loop which performs the $r-1$ steps of Algorithm Merge. At each step, we have $J \subseteq \{1, \ldots, r\}$, and the current partition is represented by the components C_i $(i \in J)$, which are called the *active* components. We maintain for each $\ell \in L$ a variable $J(\ell) = \{i \in J : \ell \in L(\langle C_i \rangle_G)\}$. At a given step, we have to find a label inducing a transverse edge which joins the closure of two active components. This amounts to looking for an ℓ such that $|J(\ell)| \geq 2$. Once such an ℓ has been found, we pick two indices $i, j \in J(\ell)$, and we merge the components C_i, C_j, letting C_i be the newly created component, and updating the variables $J(\ell)$ accordingly. An implementation of FindObstruction is given in the listing of Algorithm 3.

Lemma 18. *Suppose that* TestAgreement$(\mathcal{T}, \pi_\top)$ *returns* (no, π, I). *Then Algorithm* FindObstruction$(\mathcal{T}, \pi, I)$ *returns in* $O(kn)$ *time a conflict among* $\mathcal{T}$ *of size at most* $2k-1$.

Proof. We first argue for the correctness. If $2k \geq n$, then the set L returned by the algorithm is a conflict, as by assumption $\mathcal{T}$ has no agreement supertree. Suppose now that $2k < n$. By Lemma 17, it suffices to show that an execution $\mathcal{E}$ of Algorithm 3 simulates an execution $\mathcal{E}'$ of Algorithm Merge. More precisely, we show the following holds after each step s.

1. There is an execution $\mathcal{E}'$ of s steps of Algorithm Merge which produces the set of components C_i $(i \in J)$ and, the set of labels R is exactly the set of labels corresponding to the transverse edges which induced all the s merges in $\mathcal{E}'$.

2. For each $\ell \in L$, $J(\ell) = \{i \in J : \ell \in L(\langle C_i \rangle_G)\}$.

Algorithm 3: FINDOBSTRUCTION$(\mathcal{T}, \pi, I)$

Input: A collection $\mathcal{T} = \{T_1, \ldots, T_k\}$ of rooted trees, an obstructing position π in $\mathcal{T}$, a set $I = \{v_1, \ldots, v_r\}$ of interesting vertices for π.

Output: A conflict among $\mathcal{T}$.

```
1  if 2k ≥ n then return L
2  R ← ∅ ; J ← {1, ..., r}
3  for i from 1 to r do
4  |  C_i ← {v_i}
5  |  foreach ℓ ∈ L(v_i) do J(ℓ) ← J(ℓ) ∪ {i}
6  for s from 1 to r − 1 do
7  |  Pick ℓ ∈ L such that |J(ℓ)| ≥ 2, and choose i, j ∈ J(ℓ)
8  |  R ← R ∪ {ℓ}
9  |  for ℓ ∈ L do
10 |  |  if j ∈ J(ℓ) then J(ℓ) ← J(ℓ)\{j} ∪ {i}
11 |  for p from 1 to k do
12 |  |  if C_i ∩ V_p ≠ ∅ and C_j ∩ V_p ≠ ∅ then
13 |  |  |  foreach ℓ ∈ L(π[p]) do J(ℓ) ← J(ℓ) ∪ {i}
14 |  C_i ← C_i ∪ C_j, J ← J\{j}
15 return R
```

This is shown by induction on s. The initialization of the variables C_i and $J(\ell)$ in Lines 3-5 ensure that this is true initially. Suppose that this holds at the beginning of step s. The choice of ℓ and i, j in Line 7 ensures that $\ell \in L(\langle C_i \rangle_G) \cap L(\langle C_j \rangle_G)$, and thus there exists a transverse edge e between $\langle C_i \rangle_G$ and $\langle C_j \rangle_G$, induced by the label ℓ. The update of $C_i \leftarrow C_i \cup C_j$ and $J \leftarrow J \backslash \{j\}$ reflect the merge of C_i and C_j, and thus we can simulate step s of Algorithm MERGE which would choose transverse edge e and merge C_i and C_j. This establishes Point 1, and the update of $J(\ell)$ in Lines 9-13 ensures that Point 2 is preserved.

We now justify the running time. Let us assume that $2k < n$, as otherwise the algorithm takes $O(1)$ time. We implement the sets $J(\ell)$ by bit arrays, allowing in constant time the following operations: (i) insertion or deletion of an element, (ii) obtaining the size of the set. It follows that Lines 3-5 take $O(rn) = O(kn)$ time. Let us now analyze the time taken by the s-th iteration of the loop in Lines 6-14. Let K_s denote the set of indices p for which the condition of Line 12 holds. Then Lines 7-10 take $O(n)$ time, Lines 11-13 take $O(k + |K_s|n)$ time, and Line 14 takes $O(k)$ time. Overall, Lines 7-14 take $O(n + |K_s|n)$ time as $k = O(n)$. Observe that the sets K_s are disjoint for $s = 1, \ldots, r-1$, and thus $\sum_s |K_s| = O(k)$. It follows that the

loop of Lines 6-14 take $O(rn + kn) = O(kn)$ time, and we conclude that the whole algorithm runs in $O(kn)$ time. □

Theorem 24. AST-TR *can be solved in* $O((2k)^p kn^2)$ *time.*

Proof. We use a recursive algorithm SolveAST-TR$(\mathcal{T}, p)$. The algorithm answers "no" if $p < 0$. Otherwise, it runs TestAgreement$(\mathcal{T}, \pi^\top)$ to decide in $O(kn^2)$ if $\mathcal{T}$ has an agreement supertree. It answers "yes" in case of positive answer. In case of negative answer, it obtains a position π and a set I of interesting nodes for π. It calls FindObstruction$(\mathcal{T}, \pi, I)$ to obtain in $O(kn)$ time a conflict C among $\mathcal{T}$ of size at most $2k - 1$. Then, for each $\ell \in C$, it recursively calls SolveAST-TR$(\mathcal{T}|(L(\mathcal{T}) \setminus \{\ell\}), p - 1)$, and it answers "yes" if and only if one of the recursive calls does. The correctness follows from Lemma 18, and the running time is $O((2k)^p kn^2)$. □

3.5 Deferred proofs

3.5.1 Proofs of Section 3.3.2

The proof of Theorem 19 relies on three lemmas. Lemma 19 justifies the running time of the algorithm, while Lemmas 20 and 21 prove its correctness.

Lemma 19. *The implementation of* GetSuccessors *given in Algorithm 1 runs in* $O(kn)$ *time.*

Proof. Clearly, line 1 takes $O(n)$ time. Each iteration of the outer loop in lines 2-6 takes a total of $O(n)$ time, since each label can be in the subtree of at most one child of $\pi[i]$, and there are k iterations, for a total of $O(kn)$ time spent on lines 2-6. Line 7 takes $O(1)$ time.

We now argue that the algorithm spends a total of $O(kn)$ time on lines 8-24. The loop in line 8 executes $O(n)$ times. Since any given label can be in a given tree at most once, we have that each $|S_\ell| \leq k$ for each $\ell \in L(\pi)$. Hence the loop in line 10 executes $O(k)$ times for each execution of the outer loop in line 8, and takes constant time per iteration. Since lines 9, 22, and 23 take constant time, we have that the algorithm spends a total of $O(kn)$ time on lines 8-13, and lines 22-23.

We consider lines 14-21 separately. Since we create a total of n positions, and once a position's active flag is set to false it is never again set to true, we have that the while loop of line 14 executes a total of at most n times during the execution of the entire algorithm. The loop in line 16 executes k times and takes $O(1)$ time if the test in line 17 is true, and $O(n)$ time if the test in line 17 is false. However, the test in line 17 can be false at most k times during the entire execution of the algorithm. Hence the algorithm spends a total of $O(kn)$ executing lines 14-17 and $O(kn)$ time executing lines 18-21.

Since there are at most n positions created, line 24 takes $O(n)$ time. This completes the analysis of the runtime of GetSuccessors, showing that it takes $O(kn)$ time. □

Consider an execution of GetSuccessors$(\mathcal{T}, \pi)$, and let $G = G(\mathcal{T}, \pi)$ as before. Observe that Lines 1-6 ensure that for each $\ell \in L$, $S_\ell = \{v \in V : \ell \in L(v)\}$. Suppose that Loop 8-21 examines the labels in the order $\ell_1, \ldots, \ell_n$. Given $0 \leq i \leq n$, let $L_i = \{\ell_1, \ldots, \ell_i\}$. Let G_i denote the graph with vertex set V, and which contains an edge uv iff $L(u) \cap L(v)$ intersects L_i. Let Q_i denote the minimum nice partition of G_i, and let $\mathcal{P}_i$ denote the set of nice partitions of G_i. In the following, we let $\ell = \ell_{i+1}$.

Lemma 20. *Q_i is finer than Q_{i+1}. Furthermore, let S denote the set of classes $C \in Q_i$ which contain a vertex in S_ℓ. Then the classes of S are included in a same class K_ℓ of Q_{i+1}, and $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$, where Q' is the set of classes of Q_i included in K_ℓ.*

Proof. We first show that Q_i is finer than Q_{i+1}. Observe that a nice partition of G_{i+1} is also a nice partition of G_i, as $E(G_i) \subseteq E(G_{i+1})$. It follows that $Q_i = \sqcap_{P \in \mathcal{P}_i} P \sqsubseteq \sqcap_{P \in \mathcal{P}_{i+1}} P = Q_{i+1}$, where the inclusion holds as $\mathcal{P}_{i+1} \subseteq \mathcal{P}_i$. We deduce that each class of Q_i is included in a class of Q_{i+1}. Now, the classes of S must be included in a same class K_ℓ of Q_{i+1}, as two classes of Q_{i+1} are disconnected. Let Q' be the set of classes of Q_i included in K_ℓ. Then, Q_{i+1} is obtained from Q_i by merging together the classes of Q', and possibly some other classes. Suppose by contradiction that $Q_{i+1} \neq Q_i \backslash Q' \cup \{K_\ell\}$, then there is a class K' of Q_{i+1} distinct of K_ℓ and which is not a class of Q_i. Let $C_1, \ldots, C_m$ be the classes of Q_i included in K', with $m \geq 2$. Let $R = Q_{i+1} \backslash \{K'\} \cup \{C_1, \ldots, C_m\}$. Then Q_i is finer than R and thus R is a nice partition of G_i. On the other hand, R is not a nice partition of G_{i+1} by minimality of Q_{i+1}. It follows

that G_{i+1} contains an edge joining two classes C_p, C_q; as R is a nice partition of G_i, this edge must be induced by ℓ. But K_ℓ is the only class of Q_{i+1} which intersects S_ℓ, contradiction. We conclude that $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$. □

A *successor* of π is a position π' such that for every $i \in [k]$, either $\pi'[i] = \perp$, or $\pi'[i] \in V_i$, or $\pi'[i] = \pi[i]$. Given π' successor of π, we define the corresponding component $C_\pi \subseteq V$, such that: if $\pi'[i] = \perp$ then $C_\pi \cap V_i = \emptyset$; if $\pi'[i] = v \in V_i$ then $C_\pi \cap V_i = \{v\}$; if $\pi'[i] = \pi[i]$ then $C_\pi \cap V_i = V_i$. Let Π_i denote the set of active positions of Π at the end of step i of Loop 8-21. The construction of each position π_ℓ in Lines 14-21 ensures that Π_i is a set of successors of π. Let P_i denote the family containing (i) the sets $C_{\pi'}$ for $\pi' \in \Pi_i$, (ii) the singletons $\{v\}$ for $v \in V$ such that there is no $\pi' \in \Pi_i$ with $v \in C_{\pi'}$.

Lemma 21. *For every i $(0 \leq i \leq n)$, it holds that $P_i = Q_i$ at the end of step i of the loop in lines 8-21.*

Proof. Let us show that this holds initially. We have $\Pi_0 = \emptyset$, $V_0 = \emptyset$, and thus P_0 consists of the singletons $\{v\}$ for $v \in V$. This is clearly equal to Q_0 as G_0 is the empty graph. Let us now assume that this holds at the end of step i, let us consider step $i+1$ and let $\ell = \ell_{i+1}$. By the induction hypothesis, we have $P_i = Q_i$. By Lemma 20, we have $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$, where Q' is the set of classes of Q_i included in K_ℓ. Let S denote the set of components of Q_i corresponding (i) to positions added to Z in Lines 11-12, (ii) to singleton components $\{v\}$ for v examined at Line 13. Consider step j of Loop 14-21. At a given step, let π_ℓ^j denote the current value of π_ℓ, let C_ℓ^j denote the corresponding component, let Z^j be the set of elements of Z, and let Z_f^j be the set of elements $\pi \in Z$ with $\mathsf{active}(\pi) = \mathsf{false}$. Let D_ℓ^j denote the set of vertices $v \in C_\ell^j$ such that there is no $\pi' \in \Pi_i$ with $v \in C_{\pi'}$.

Claim 1. At step j of Loop 14-21, we have:

(a) $D_\ell^j \cup \bigcup_{\pi' \in Z_f^j} C_{\pi'}$ is included in C_ℓ^j;

(b) C_ℓ^j is included in $D_\ell^j \cup \bigcup_{\pi' \in Z^j} C_{\pi'}$ and in K_ℓ;

(c) for each $\pi' \in Z^j$, $C_{\pi'} \subseteq K_\ell$.

Proof. Let us verify that this holds for $j = 0$. Observe that $C_\ell^0 = D_\ell^0$, as each vertex v added to C_ℓ^0 is obtained in Line 13 and has $\mathsf{position(v)} = \emptyset$. Point (a) is an equality. Points (b) and (c) follow from the fact that the components of S are included in K_ℓ, according to the definition of K_ℓ in Lemma 20.

Let us now verify it for $j+1$, assuming that it holds for j. Suppose that step $j+1$ examines position $\pi' \in Z^j$. Let Z' be the set of positions added at Line 21 during this step. Lines 16-21 compute π_ℓ^{j+1} such that $C_\ell^{j+1} = \langle C_\ell^j \cup C_{\pi'} \rangle_{G_{i+1}}$. By the induction hypothesis, $D_\ell^j \cup \bigcup_{\pi' \in Z_f^j} C_{\pi'} \subseteq C_\ell^j \subseteq C_\ell^{j+1}$; by definition, $D_\ell^{j+1} \subseteq C_\ell^{j+1}$ and $C_{\pi'} \subseteq C_\ell^{j+1}$; as $Z_f^{j+1} = Z_f^j \cup \{\pi'\}$, we deduce that Point (a) holds. Let us show Point (b). As $C_\ell^j \subseteq K_\ell$ and $C_{\pi'} \subseteq K_\ell$ by the induction hypothesis, it follows that $C_\ell^j \cup C_{\pi'} \subseteq K_\ell$, and thus $C_\ell^{j+1} \subseteq K_\ell$ as K_ℓ is a nice set. Now, observe that for every $v \in C_\ell^{j+1} \backslash C_\ell^j$ we have either $v \in D_\ell^{j+1}$ (if $\mathsf{position(v)} = \emptyset$ at Line 21) or $v \in C_{\pi'}$ for some $\pi' \in Z'$ (if $\mathsf{position(v)} = \{\pi'\}$ at Line 21). It follows that $C_\ell^{j+1} \subseteq C_\ell^j \cup D_\ell^{j+1} \cup \bigcup_{\pi' \in Z'} C_{\pi'} \subseteq D_\ell^{j+1} \cup \bigcup_{\pi' \in Z^j} C_{\pi'}$ (using the induction hypothesis), and thus Point (b) holds. Let us show Point (c). Observe that for each $\pi' \in Z'$, $C_{\pi'}$ intersects C_ℓ^{j+1}. As $C_\ell^{j+1} \subseteq K_\ell$ and as $C_{\pi'}$ is a class of Q_i, it follows that $C_{\pi'} \subseteq K_\ell$. ◊

Suppose that we have reached Line 23. Let Π' be the set of positions in Z at this step, then $\Pi_{i+1} = \Pi_i \backslash \Pi' \cup \{\pi_\ell\}$. Thus, we have $P_{i+1} = P_i \backslash P' \cup \{C_\ell\}$, where C_ℓ is the component corresponding to π_ℓ. Recall that $Q_{i+1} = Q_i \backslash Q' \cup \{K_\ell\}$ and that $P_i = Q_i$. To show that $P_{i+1} = Q_{i+1}$, we will prove that (i) $C_\ell \subseteq K_\ell$, (ii) P_{i+1} is a nice partition of G_{i+1}.

(i) $C_\ell \subseteq K_\ell$. Indeed, by applying Claim 1 at the last step of Loop 14-21, we obtain that $C_\ell \subseteq K_\ell$ by Point (b).

(ii) P_{i+1} is a nice partition of G_{i+1}. We first show that P_{i+1} is a partition of V. Let D_ℓ denote the value of D_ℓ^j at the last step of Loop 14-21. Note that $P_{i+1} = P_i \backslash P' \cup \{C_\ell\}$, where P' contains (i) the singleton components $\{v\}$ for $v \in D_\ell$, (ii) the set of components corresponding to the positions in Z. Let X denote the union of the components in P', observe that $X = D_\ell \cup \bigcup_{\pi \in Z} C_\pi$. By Points (a) and (b) of Claim 1, we have $X = C_\ell$. As P_i is a partition of V, we obtain that P_{i+1} is a partition of V.

We now show that P_{i+1} is nice. Each class of P_{i+1} is nice, as it is either a class of P_i (which is a nice partition of G_i), or the set C_ℓ (which is nice). Suppose that P_{i+1} contains

two classes C, C' that are connected. As P_i is a nice partition of G_i finer than P_{i+1}, it holds that P_{i+1} is a nice partition of G_i. Thus, C and C' are disconnected in G_i, and they must be connected in G_{i+1} by an edge induced by ℓ. But C_ℓ is the only class of P_{i+1} which intersects S_ℓ, a contradiction. □

The proof of Theorem 19 follows directly from Lemmas 19, 20, and 21.

We conclude this section with a proof of Theorem 20. Recall that when the set of successor positions returned by GetSuccessors is a singleton, the algorithm returns a set of vertices I along with the obstructing position π. Theorem 20 states that, in this case, the vertices in I are interesting for π. We have delayed the proof of this theorem until now for expository reasons: It is not until Section 3.4 that we developed the machinery required for the proof.

of Theorem 20. The following properties of Algorithm Merge demonstrate that the set I satisfies the definition of interesting vertices:

- Each vertex in I starts in its own class.
- At each step of the algorithm a transverse edge is found and two of the classes are merged.

Let $K = \{i \in [k] : \pi[i] \neq \perp\}$. By Lemma 15, Algorithm Merge run on G and I returns $\{I\}$. Since there are $|I| = 2|K|$ initial classes, it follows that $2|K| - 1$ transverse edges are needed to merge them into a single class. Furthermore, since each interesting vertex is initially the unique element in its class, it must be an endpoint of the transverse edge found that merged that class with the final class. Thus, the set of transverse edges used by Algorithm Merge prove that I is indeed a set of interesting vertices. □

3.5.2 Proof of Lemma 15

For convenience, we introduce an abstract version of GetSuccessors called Algorithm $\mathcal{A}$. The algorithm takes as input $\mathcal{T}$ and the position π, and it returns a partition P of V, and a set I of interesting vertices. For each $\ell \in L$, the algorithm constructs the set $S_\ell = \{v \in V : \ell \in L(v)\}$. Initially, P consists of the singletons $\{v\}$ for $v \in V$, and $I = \emptyset$. Then, the algorithm successively

examines each label $\ell \in L$. When examining ℓ, it constructs a set $Z \subseteq P$ and $K \subseteq V$ in two steps. It starts with $Z = \emptyset$ and $K = \emptyset$, and it does the following.

- Phase A: for each $C \in P$ intersecting S_ℓ, add C to Z.
- Phase B: while Z contains an unprocessed class C, extract C from Z and do the following. For every $i \in [k]$, if K and C respectively contain a vertex x and a vertex y where $x, y \in V_i$, then add x, y to I, and for every $v \in V_i \backslash \{x, y\}$ add to Z the class $C' \in P$ that contains v. Then let $K \leftarrow \langle K \cup C \rangle_G$.
- At the end of Phase B, let P' be the set of classes of P which have been added to Z. Let $P \leftarrow P \backslash P' \cup \{K\}$.

It is not difficult to see that Algorithm $\mathcal{A}$ is equivalent to Algorithm GetSuccessors, so for every execution of GetSuccessors producing the set of vertices I, there is a corresponding execution of Algorithm $\mathcal{A}$ which produces I. An execution of Algorithm $\mathcal{A}$ can be represented by a *merge forest*. This is an ordered rooted forest $\mathcal{M}$, where each node u of $\mathcal{M}$ is associated to a component K_u produced by the algorithm, such that at each step of Algorithm $\mathcal{A}$ the roots of $\mathcal{M}$ correspond to the classes of P. $\mathcal{M}$ is constructed as follows. At the beginning of Algorithm $\mathcal{A}$, $\mathcal{M}$ consists of one isolated node for each class of P. When Algorithm $\mathcal{A}$ examines label ℓ, let P' be the set produced at the end of Phase B and let K be the class added to P, then $\mathcal{M}$ is updated by adding a vertex u with $K_u = K$, and by adding an arc (u, v) for each root v of $\mathcal{M}$ corresponding to a class $K_v \in P'$. Then, the new children of u correspond to the classes added to Z, and they are ordered according to the first time when they were added to Z.

We state below some simple properties of the merge forest. The following lemma can be proved by a similar argument as in the proof of Lemma 21.

Lemma 22. *Let u be an internal node of $\mathcal{M}$ with children $u_1, \ldots, u_p$. Then $\{K_{u_1}, \ldots, K_{u_p}\}$ is a partition of K_u.*

Let $i \in [k]$, and let u be a node of $\mathcal{M}$. We say that u *merges* V_i if $V_i \subseteq K_u$ but there is no child v of u with $V_i \subseteq K_v$. Suppose that u merges V_i. Given $x \in V_i$, we let $C_x(u)$ denote the child v of u such that $x \in K_v$. We let $Ch(u)$ denote the ordered list of children of u in $\mathcal{M}$.

Lemma 23. *Suppose that u merges V_i. Then I contains two vertices $x, y \in V_i$, and for every $z \in V_i \backslash I$, $C_x(u), C_y(u)$ precede $C_z(u)$ in $Ch(u)$.*

We are now in position to prove Lemma 15.

of Lemma 15. By definition of π and I, it holds that GetSuccessors$(\mathcal{T}, \pi)$ has returned $(\{\pi\}, I)$. It follows that there is a corresponding execution of Algorithm $\mathcal{A}$ on $\mathcal{T}, \pi$ which returns $(\{V\}, I)$. Let $\mathcal{M}$ be the merge forest corresponding to this execution, then $\mathcal{M}$ has a single root r with $K_r = V$. Consider an execution of Algorithm Merge on G, I. We show that as long as P contains at least two classes, the algorithm finds a transverse edge. Suppose that $P = \{C_1, \ldots, C_p\}$ with $p \geq 2$, we thus need to find a transverse edge joining $\langle C_i \rangle_G$ and $\langle C_j \rangle_G$ for some i, j.

We will need the following definitions. Let v be a node of $\mathcal{M}$. Given $x \in K_v$, we color x with color $1 \leq i \leq p$ if $x \in \langle C_i \rangle_G$. We say that v is *split* if K_v contains two colored vertices of different colors. We say that v is *full* if for each $i \in [k]$, if $K_v \cap V_i = \{x\}$ then x is colored. For every $i \in [k]$ such that $V_i \subseteq K_v$, the vertices of $V_i \backslash I$ are called *secondary vertices* of K_v. Given v' child of v in $\mathcal{M}$, we say that v' is a *secondary child* of v if there exists $i \in [k]$ such that $K_{v'} \cap V_i = \{x\}$ with x secondary vertex of K_v; otherwise, we say that v' is a *primary child* of v.

Let S be the set of nodes of $\mathcal{M}$ that are split and full. Observe that the root r of $\mathcal{M}$ is in S. Indeed, r is split as K_r contains all interesting nodes and as $p \geq 2$; r is full as there is no $i \in [k]$ such that $|K_r \cap V_i| = 1$. Let u be a deepest node of S, and let $u_1, \ldots, u_m$ be its ordered list of children. We say that a node v of $\mathcal{M}$ is *monochromatic* (with color c) iff all vertices of K_v have the same color c.

Claim 1. If v is a primary child of u, then v is added during Phase A of Algorithm $\mathcal{A}$. Furthermore, v is full and monochromatic.

Proof. For the first point, observe that if v is added during Phase B of Algorithm $\mathcal{A}$ then K_v contains a secondary vertex of K_u and thus v is a secondary child of u. Let us now show the second point. Suppose by contradiction that v is not full. Then there is some $i \in [k]$ such that $K_v \cap V_i = \{x\}$ with x uncolored. As u is full, we cannot have $K_u \cap V_i = \{x\}$ and thus $V_i \subseteq K_u$.

As x is uncolored, we have $x \in V_i \backslash I$, and thus x is a secondary vertex of K_u. We conclude that v is a secondary child of u, contradiction. It follows that v is full, and by choice of u it cannot be split. Thus, all colored vertices of K_v have the same color c. Now, if $K_v \cap V_i = \{x\}$ then x has color c (as K_v is full), and if $V_i \subseteq K_v$ then the two vertices of $V_i \cap I$ have color c, which implies that all vertices of V_i have color c. It follows that v is monochromatic with color c. $\Diamond$

By Claim 1, each primary child u_i of u is monochromatic with color c_i.

Claim 2. There exist two primary children u_i, u_j of u such that $c_i \neq c_j$.

Proof. By way of contradiction, assume that all primary children have the same color c. We show by induction on $1 \leq j \leq m$ that u_j is monochromatic with color c. This holds if u_j is a primary child of u, so let us assume that u_j is a secondary child of u. Let $J \subseteq K_{u_j}$ be the set of vertices $x \in K_{u_j}$ such that $K_{u_j} \cap V_i = \{x\}$ for some $i \in [k]$, and let $J' \subseteq J$ be the set of vertices of J that are secondary vertices of K_u. Then $J' \neq \emptyset$ as u_j is a secondary child of u. Consider $z \in J'$ and suppose that $K_{u_j} \cap V_i = \{z\}$. As z is a secondary vertex of K_u, we then have $V_i \subseteq K_u$. Then u merges V_i, and by Lemma 23 it follows that I contains two elements $x, y \in V_i$. Let u_p, u_q be the children of u such that $x \in K_{u_p}, y \in K_{u_q}$. We have $p, q < j$ by Lemma 23. We can thus apply the induction hypothesis to obtain that x, y have color c, which implies that z has color c. We obtain that all vertices of J' have color c. On the other hand, all vertices of $J \backslash J'$ are colored, as u is full. We conclude that u_j is full, with some vertex having color c. By choice of u, its child u_j cannot be split. A similar reasoning as in the proof of Claim 1 shows that u_j is monochromatic with color c. This concludes the induction, and we obtain that all children of u are monochromatic with color c. By Lemma 22, we obtain that u is monochromatic with color c, contradicting the assumption that u is split. $\Diamond$

Claim 2 yields two primary children u_i, u_j of different colors c, c'. By Claim 1, they are added during Phase A of Algorithm $\mathcal{A}$, and thus the label that is examined in the iteration of Algorithm $\mathcal{A}$ when K_u is built induces an edge between K_{u_i} and K_{u_j}. We have thus shown the existence of a transverse edge between $\langle C_c \rangle_G$ and $\langle C_{c'} \rangle_G$, which concludes the proof. $\square$

3.6 Concluding Remarks

We have given $O((2k)^p kn^2)$ time algorithms for both the AST-EC and AST-TR problems, thus showing they are fixed-parameter tractable for parameters k and p. We remark here that the bound of $2k-1$ given for the obstruction set of AST-TR (Lemmas 17 and 18) is tight.

Our proof that AST-EC is NP-hard relies on a reduction from the parameterized Multicut problem to the AST-EC problem parameterized by p. As Multicut is fixed-parameter tractable [10, 56], this leaves open the question of whether AST-EC could be fixed-parameter tractable in p only. It is known that AST-TR is fixed-parameter intractable for parameter p [6].

Our focus here was on agreement supertrees. A compatible supertree is one that contains a refinement of each of the input trees. There are natural analogs of AST-EC and AST-TR for compatible supertrees. For binary input trees, compatibility is equivalent to agreement, so the results of [34] imply fixed-parameter tractability. However, for input trees of arbitrary degree, we have established that any upper bound on the cardinality of an obstruction set is at least $c2^k$. Hence, the techniques given here are unlikely to imply efficient fixed-parameter tractability for the analogs of AST-TR and AST-EC to compatible supertrees.

There are also analogs of both AST-EC and AST-TR to unrooted trees. Although Maximum Agreement Supertree (Smast) has been studied for unrooted trees [6, 40], the AST-EC and AST-TR problems for unrooted trees do not seem to have been studied before.

CHAPTER 4. The Tile Assembly Model and Discrete Fractals

In this chapter we give a brief introduction to the Tile Assembly Model (TAM) and Discrete Fractals that is sufficient for understanding the results presented in Chapters 5 and 6. For a more thorough introduction to the TAM see [84, 65]. Our notation for the TAM follows that of [51] but is tailored somewhat to our objectives.

4.1 Notation and Terminology

We work in the discrete Euclidean plane $\mathbb{Z}^2$. We write U_2 for the set of all *unit vectors* in $\mathbb{Z}^2$. We often refer to the elements of U_2 as the cardinal directions, and write $\vec{u}_{\mathrm{N}}$ for $(0,1)$, $\vec{u}_{\mathrm{S}}$ for $(0,-1)$, $\vec{u}_{\mathrm{E}}$ for $(1,0)$, and $\vec{u}_{\mathrm{W}}$ for $(-1,0)$.

For any set $S \subseteq \mathbb{N}^2$, let $S|_m$ denote the set $\{(x,y) \in S : x < m \text{ and } y < m\}$. For $u, v \in \mathbb{N}$, let $S|_m[u,v] = \{(x,y) : (x-u, y-v) \in S|_m\}$, i.e., the set $S|_m$ translated to (u,v).

Let X and Y be sets. We write $[X]^2$ for the set of all 2-element subsets of X. For a partial function $f : X \dashrightarrow Y$, we write $f(x)\!\downarrow$ if $x \in \operatorname{dom} f$ and $f(x)\!\uparrow$ otherwise. We write $X \Delta Y$ for the *symmetric difference* of X and Y. For a *Boolean* expression ϕ, we write $[\![\phi]\!]$ for the value of ϕ, i.e., $[\![\phi]\!] = 1$ when ϕ is true, and $[\![\phi]\!] = 0$ when ϕ is false.

All graphs here are undirected graphs of the form $G = (V, E)$, where $V \subseteq \mathbb{Z}^2$ is a set of *vertices* and $E \subseteq [V]^2$ is a set of *edges*. A *grid graph* is a graph where each $\{\vec{m}, \vec{n}\} \in E$ satisfies $\vec{m} - \vec{n} \in U_2$. If E contains every $\{\vec{m}, \vec{n}\} \in [V]^2$ such that $\vec{m} - \vec{n} \in U_2$, we say it is the *full grid graph* on V, written $G_V^{\#}$. A *cut* of a graph is a partition of V into two subsets. A *binding function* on a graph is a function $\beta : E \to \mathbb{N}$. If β is a binding function on G and C is a cut of G, then the *binding strength of β on C* is

$$\beta_C = \sum \{\beta(e) \mid e \in E \text{ and } e \cap C_0 \neq \emptyset \text{ and } e \cap C_1 \neq \emptyset\},$$

and the *binding strength of* β *on* G is

$$\beta_G = \min\{\beta_C \mid C \text{ is a cut of } G\}.$$

A *binding graph* is an ordered triple (V, E, β), where β is a binding function on (V, E). For $\tau \in \mathbb{N}$, a binding graph (V, E, β) is τ*-stable* when $\beta_{(V,E)} \geq \tau$.

4.2 The Tile Assembly Model

Rothemund and Winfree [65, 84] introduced the Tile Assembly Model (TAM), a constructive version of Wang tiling [82], in order to study the growth of DNA crystals. The TAM models the self-assembly of unit square *tiles* that can be translated, but not rotated, so that each tile has a well defined "side $\vec{u}$" for each $\vec{u} \in U_2$. Each side $\vec{u}$ of t has a *glue* $t(\vec{u}) = (\mathrm{col}_t(\vec{u}), \mathrm{str}_t(\vec{u}))$ where $\mathrm{col}_t(\vec{u}) \in \Sigma^*$ (for some fixed alphabet Σ) is the glue *color*, and $\mathrm{str}_t(\vec{u}) \in \mathbb{N}$ is the glue *strength* (usually 0, 1, or 2). Two tiles with the same glue on each side are of the same *tile type*. Two tiles placed next to each other *interact* if the glues on their abutting sides match in both color and strength. Intuitively, a tile models a DNA double crossover molecule and the glues correspond to the "sticky ends" on the four arms of the molecule.

Figure 4.1 gives an example illustration of a tile. Glue strengths are represented by lines that are dotted for 0, solid for 1, and solid with notches for 2. Glue colors are drawn in the interior of the tile on the corresponding side. In this example $t(\vec{u}_{\mathrm{N}}) = (a, 1)$, $t(\vec{u}_{\mathrm{E}}) = (b, 1)$, $t(\vec{u}_{\mathrm{S}}) = (\lambda, 0)$ where λ represents the empty string, and $t(\vec{u}_{\mathrm{W}}) = (c, 2)$. Also, we sometimes give a label to a tile type. This label does not play a role in the TAM, it is only to make referring to tiles of that type more convenient.

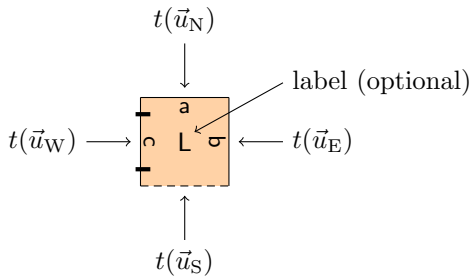


Figure 4.1: An example illustration of a tile.

Let T be a set of tile types. A T-configuration is a partial function $\alpha : \mathbb{Z}^2 \dashrightarrow T$. For $\vec{m}, \vec{n} \in \operatorname{dom}\alpha$, the tiles at these locations *interact* with *strength*

$$\operatorname{str}_\alpha(\vec{m}, \vec{n}) = [\![\vec{n} - \vec{m} \in U_2]\!] \cdot \operatorname{str}_{\alpha(\vec{m})}(\vec{n} - \vec{m}) \cdot [\![\alpha(\vec{m})(\vec{n} - \vec{m}) = \alpha(\vec{n})(\vec{m} - \vec{n})]\!].$$

The *binding graph* of α is $G_\alpha = (\operatorname{dom}\alpha, E, \beta)$, where

$$E = \left\{ \{\vec{m}, \vec{n}\} \in [V]^2 \mid \operatorname{str}_\alpha(\vec{m}, \vec{n}) > 0 \right\},$$

and for all $\{\vec{m}, \vec{n}\} \in E$, $\beta(\{\vec{m}, \vec{n}\}) = \operatorname{str}_\alpha(\vec{m}, \vec{n})$. For $\tau \in \mathbb{N}$, α is *τ-stable* if G_α is τ-stable. We write $\mathcal{A}_T^\tau$ for the set of all τ-stable T-configurations. Let $\alpha, \alpha' \in \mathcal{A}_T^\tau$. If $\operatorname{dom}\alpha \subseteq \operatorname{dom}\alpha'$ and $\alpha(\vec{m}) = \alpha'(\vec{m})$ for all $\vec{m} \in \operatorname{dom}\alpha$, then α is a *subconfiguration* of α' and we write $\alpha \sqsubseteq \alpha'$. If $|\operatorname{dom}\alpha' \setminus \operatorname{dom}\alpha| = 1$, then α' is a *single-tile-extension* of α and we write $\alpha' = \alpha + (\vec{m} \mapsto t)$ where $\{\vec{m}\} = \operatorname{dom}\alpha' \setminus \operatorname{dom}\alpha$ and $t = \alpha'(\vec{m})$. For each $t \in T$, the *τ-t-frontier of* α is

$$\partial_t^\tau \alpha = \left\{ \vec{m} \in \mathbb{Z}^2 \setminus \operatorname{dom}\alpha \mid (\Sigma_{\vec{u} \in U_2} \operatorname{str}_{\alpha + (\vec{m} \mapsto t)}(\vec{m}, \vec{m} + \vec{u})) \geq \tau \right\}, \text{ and}$$

the *τ-frontier of* α is

$$\partial^\tau \alpha = \bigcup_{t \in T} \partial_t^\tau \alpha.$$

We say α is *terminal* when $\partial^\tau \alpha = \emptyset$.

A *tile assembly system* (TAS) is an ordered triple $\mathcal{T} = (T, \sigma, \tau)$ where T is a finite set of tile types called the *tileset*, the *seed assembly* $\sigma \in \mathcal{A}_T^\tau$ is such that $\operatorname{dom}\sigma = \vec{0}$, and $\tau \in \mathbb{N}$ is the *temperature*. In this thesis we will always have that $\tau = 2$. We will also assume that the tile assembly system contains an infinite number of tiles of each type, and each type occurs at the same concentration.

The process starts with the seed assembly and growth occurs by single tiles attaching one at a time. A tile can attach at a site where the summed strength of the glues on sides that interact with the existing structure is at least the temperature. More formally, an *assembly sequence in* $\mathcal{T}$ is a sequence $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ where $\alpha_0 = \sigma$, $k \in \mathbb{Z}^+ \cup \{\infty\}$ and for each $0 \leq i < k$, $\alpha_{i+1} = \alpha_i + (\vec{m} \mapsto t)$ for some $t \in T$ and $\vec{m} \in \partial_t^\tau \alpha_i$. The *result of* $\vec{\alpha}$, written $\operatorname{res}\vec{\alpha}$, is the unique $\alpha \in \mathcal{A}_T^\tau$ satisfying $\operatorname{dom}\alpha = \bigcup_{0 \leq i < k} \operatorname{dom}\alpha_i$ and for each $0 \leq i < k$, $\alpha_i \sqsubseteq \alpha$. We

write $\sigma \longrightarrow \alpha$ if there exists an assembly sequence $\vec{\alpha}$ in $\mathcal{T}$ such that $\alpha = \operatorname{res}\vec{\alpha}$. The set of *producible assemblies* is

$$\mathcal{A}[\mathcal{T}] = \{\alpha \in \mathcal{A}_T^\tau \mid \sigma \longrightarrow \alpha\}$$

and the set of *terminal assemblies* is

$$\mathcal{A}_\square[\mathcal{T}] = \{\alpha \in \mathcal{A}[\mathcal{T}] \mid \partial^\tau \alpha = \emptyset\}.$$

$\mathcal{T}$ is *directed* if $|\mathcal{A}_\square[\mathcal{T}]| = 1$.

A set $X \subseteq \mathbb{Z}^2$ *weakly self-assembles* in a TAS $\mathcal{T}$ if the tileset T can be partitioned into two subsets, conventionally called the "black" tiles and "white" tiles, and in every terminal assembly of $\mathcal{T}$, the set of locations at which black tiles have been placed is exactly X. We say X *weakly self-assembles* if X weakly self-assembles in some TAS.

A set X *strictly self-assembles in* $\mathcal{T}$ if every $\alpha \in \mathcal{A}_\square[\mathcal{T}]$ satisfies $\operatorname{dom}\alpha = X$, i.e., if in every terminal assembly of $\mathcal{T}$ the set of locations at which tiles have been placed is exactly X. We say X *strictly self-assembles* if X strictly self-assembles in some TAS.

Let $\mathcal{T} = (T, \sigma, \tau)$ be a TAS, $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be an assembly sequence in $\mathcal{T}$, and $\alpha = \operatorname{res}\vec{\alpha}$. For each $\vec{m} \in \mathbb{Z}^2$, the $\vec{\alpha}$-*index* of $\vec{m}$ is

$$i_{\vec{\alpha}}(\vec{m}) = \min\{i \in \mathbb{N} \mid \vec{m} \in \operatorname{dom}\alpha_i\}.$$

If $\vec{m}, \vec{n} \in \operatorname{dom}\alpha$ and $i_{\vec{\alpha}}(\vec{m}) < i_{\vec{\alpha}}(\vec{n})$, we say $\vec{m}$ *precedes* $\vec{n}$ *in* $\vec{\alpha}$, and write $\vec{m} \prec_{\vec{\alpha}} \vec{n}$. For $X \subseteq \operatorname{dom}\alpha$, α *restricted to* X, written $\alpha\restriction X$, is the unique T-configuration satisfying $(\alpha\restriction X) \sqsubseteq \alpha$ and $\operatorname{dom}(\alpha\restriction X) = X$.

4.2.1 Local Determinism

Winfree and Soloveichik [76] introduced local determinism as a way to prove that a TAS is directed. Let $\mathcal{T} = (T, \sigma, \tau)$ be a TAS, $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be an assembly sequence in $\mathcal{T}$, and $\alpha = \operatorname{res}\vec{\alpha}$. For each $\vec{m} \in \operatorname{dom}\alpha$, define [76] the sets

$$\mathrm{IN}^{\vec{\alpha}}(\vec{m}) = \{\vec{u} \in U_2 \mid \vec{m} + \vec{u} \prec_{\vec{\alpha}} \vec{m} \text{ and } \operatorname{str}_{\alpha_{i_{\vec{\alpha}}(\vec{m})}}(\vec{m}, \vec{m} + \vec{u}) > 0\}, \text{ and}$$
$$\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) = \{\vec{u} \in U_2 \mid -\vec{u} \in \mathrm{IN}^{\vec{\alpha}}(\vec{m} + \vec{u})\}.$$

Then, $\vec{\alpha}$ is *locally deterministic* [76] if the following three conditions hold.

(1) For all $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$,

$$\sum_{\vec{u} \in \mathrm{IN}^{\vec{\alpha}}(\vec{m})} \mathrm{str}_{\alpha_{i_{\vec{\alpha}}(\vec{m})}}(\vec{m}, \vec{m}+\vec{u}) = \tau.$$

(2) For all $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ and $t \in T \setminus \{\alpha(\vec{m})\}$,

$$\vec{m} \notin \partial_t^{\tau}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))).$$

(3) $\partial^{\tau}\alpha = \emptyset$.

Conceptually, (1) requires that each tile added in $\vec{\alpha}$ "just barely" binds to the existing assembly; (2) holds when the tiles at $\vec{m}$ and $\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m})$ are removed from α, no other tile type can attach to the assembly at location $\vec{m}$; and (3) requires that α is terminal. A TAS is *locally deterministic* if it has a locally deterministic assembly sequence. Soloveichik and Winfree [76] proved every locally deterministic TAS is directed.

4.3 Discrete Fractals

In this thesis we will be concerned with approximating fractal structures with strict self-assembly. Fractals are normally bounded and have the same structure at arbitrarily small scales. The TAM models the self-assembly of tiles, which are discrete objects, so structures that self-assemble are fundamentally discrete. Thus, when considering the self-assembly of a fractal, we use a discrete version which is unbounded and has the same detail at arbitrarily large scales.

Our primary concern will be with a subset of the discrete numerically self-similar fractals. A *discrete fractal* is an unbounded subset F of the Euclidean plane that has the same structure at arbitrarily large scales. In all the examples we consider, F will be a subset of $\mathbb{N}^2$.

Definition 1. *For any integer $p > 1$, let K be a nonempty, proper subset of $\mathbb{N}^2|_p$. The* discrete self-similar fractal with $p \times p$ kernel K *is the set F defined by*

$$\begin{aligned} F|_p &= K, \\ F|_{p^{k+1}} &= \bigcup_{(s,t)\in K} F|_{p^k}[sp^k, tp^k]. \end{aligned}$$

4.3.1 Zeta-Dimension

A fractal dimension is a measure of how completely a fractal fills space. The most commonly used fractal dimension for discrete fractals is zeta-dimension. Although the origins of zeta-dimension lie in eighteenth and nineteenth century number theory, namely Euler's *zeta-function* [26], it has been rediscovered many times by researchers in a variety of fields. See [21] for a review of the origins of zeta-dimension, the development of its basic theory, and the connections between zeta-dimension and classical fractal dimensions.

In this thesis we use the entropy characterization of zeta-dimension [14]. For each $\vec{m} \in \mathbb{Z}^2$, let $||\vec{m}||$ be the Euclidean distance from the origin to $\vec{m}$, i.e., if $\vec{m} = (m_1, m_2)$ then $||\vec{m}|| = \sqrt{m_1^2 + m_2^2}$. For $A \subseteq \mathbb{Z}^2$ and $I \subseteq [0, \infty)$, let $A_I = \{\vec{m} \in A \mid ||\vec{m}|| \in I\}$. Then, the *$\zeta$-dimension* (*zeta-dimension*) of a set $A \subseteq \mathbb{Z}^2$ is

$$\mathrm{Dim}_\zeta(A) = \limsup_{n \to \infty} \frac{\log_2 |A_{[0,n]}|}{\log_2 n}.$$

By routine calculus it follows that

$$\mathrm{Dim}_\zeta(A) = \limsup_{n \to \infty} \frac{\log_2 |A_{[0,2^n)}|}{n}. \tag{4.1}$$

Note that ζ-dimension has the following functional properties of a fractal dimension [21].

Observation 3. *Let $A, B \subseteq \mathbb{Z}^2$. Then,*

(1) $A \subseteq B \implies \mathrm{Dim}_\zeta(A) \leq \mathrm{Dim}_\zeta(B)$ *(monotonicity), and*

(2) $\mathrm{Dim}_\zeta(A \cup B) = \max\{\mathrm{Dim}_\zeta(A), \mathrm{Dim}_\zeta(B)\}$ *(stability).*

It is easy to verify using Definition 1 that if F is a fractal with $p \times p$ kernel K, then $\mathrm{Dim}_\zeta(F) = \log_p |K|$.

4.3.2 Approximate Self-Assembly of a Discrete Fractal

We now formally define the notion of approximating a fractal structure with strict self-assembly.

Definition 2. *Let S be a subset of $\mathbb{Z}^2$. An* in-place approximation of S in the aTAM *is a set $X \supset S$ that strictly self-assembles in the aTAM in such a way that specially labeled tiles appear at exactly the positions corresponding to points in S and X has exactly the same fractal dimension as S.*

That is, the approximation X contains additional elements occupying the "negative space" of S, as illustrated in Fig. 6.5, but the space occupied by these additional elements is negligible in the sense that it does not increase the fractal dimension of the resulting structure. It is also the case that the tile types that occupy the additional space are distinct from the tile types used to assembly the intended fractal structure.

CHAPTER 5. Approximate Self-Assembly of the Sierpinski Triangle

Modified from a paper published in *Theory of Computing Systems*

Jack H. Lutz and Brad Shutters

Abstract

The Tile Assembly Model is a Turing universal model that Winfree introduced in order to study the nanoscale self-assembly of complex DNA crystals. Winfree exhibited a self-assembly that tiles the first quadrant of the Cartesian plane with specially labeled tiles appearing at exactly the positions of points in the Sierpinski triangle. More recently, Lathrop, Lutz, and Summers proved that the Sierpinski triangle cannot self-assemble in the "strict" sense in which tiles are not allowed to appear at positions outside the target structure. Here we investigate the strict self-assembly of sets that approximate the Sierpinski triangle. We show that every set that does strictly self-assemble disagrees with the Sierpinski triangle on a set with fractal dimension at least that of the Sierpinski triangle (≈ 1.585), and that no subset of the Sierpinski triangle with fractal dimension greater than 1 strictly self-assembles. We show that our bounds are tight, even when restricted to supersets of the Sierpinski triangle, by presenting a strict self-assembly that adds communication fibers to the fractal structure without disturbing it. To verify this strict self-assembly we develop a generalization of the local determinism method of Soloveichik and Winfree.

5.1 Introduction

Self-assembly is a process in which simple objects autonomously combine to form complex structures as a consequence of specific, local interactions among the objects themselves. It

occurs spontaneously in nature as well as in engineered systems and is a fundamental principle of structural organization at all scales. Since the pioneering work of Seeman [69], the self-assembly of DNA molecules has developed into a field with rich interactions between the theory of computing (the information processing properties of DNA) and geometry (the structural properties of DNA), and with many applications to nanotechnology [70].

Winfree [84] introduced the Tile Assembly Model (TAM) as a mathematical model of self-assembly in order to study the growth of complex DNA crystals. The TAM is a constructive version of Wang tiling [81, 82] that models the self-assembly of unit square *tiles* that can be translated, but not rotated. A tile has a *glue* on each side that is made up of a *color* and an integer *strength* (usually 0, 1, or 2). Intuitively, a tile models a DNA double crossover molecule and the glues correspond to the "sticky ends" on the four arms of the molecule. Two tiles with the same glue on each side are of the same *tile type*. Two tiles placed next to each other *interact* if the glues on their abutting sides match in both color and strength.

A *tile assembly system* (TAS) is a finite set of tile types, a single tile for the *seed*, and a specified integer *temperature* (usually 2). The process starts with the seed tile placed at the origin, and growth occurs by single tiles attaching one at a time. A tile can attach at a site where the summed strengths of the glues on sides that interact with the existing structure is at least the temperature. The assembly is *terminal* when no more tiles can attach. A TAS is *directed* if it always results in a unique terminal assembly. Winfree proved the TAM is Turing universal [84]. The TAM is described formally in Section 4.2.

This paper is concerned with the self-assembly of fractals. Structures that self-assemble in naturally occurring biological systems are often fractals, which have advantages for materials transport, heat exchange, information processing, and robustness [7]. There are two types of fractals, continuous and discrete. Continuous fractals are typically bounded and exhibit the same structure at arbitrarily small scales. In contrast, discrete fractals are unbounded and exhibit the same structure at arbitrarily large scales. Many fractals have both continuous and discrete "versions" that share the same fractal dimension and other properties [79]. This duality is a topic of ongoing investigation [83, 4, 41, 60].

The TAM models the bottom-up self-assembly of discrete tiles, so structures that self-

assemble in the TAM are discrete. We thus restrict our attention to fractals that are discrete. There are two main notions of the self-assembly of such a fractal. In weak self-assembly one typically causes a two-dimensional surface to self-assemble with the desired fractal appearing as a labeled subset of the surface. In contrast, strict self-assembly requires the fractal, and nothing else, to self-assemble. For many purposes, strict self-assembly is needed in order to achieve the above mentioned advantages of fractal structures.

The Sierpinski triangle is a canonical starting point for many investigations of fractals, and this is certainly true for self-assembly. Winfree [19] showed that the Sierpinski triangle weakly self-assembles, and Rothemund, Papadakis, and Winfree [66] achieved a molecular implementation of this self-assembly. More recently, Lathrop, Lutz, and Summers [51] proved that the Sierpinski triangle cannot strictly self-assemble. Patitz and Summers [62] then exhibited a large class of fractals that cannot strictly self-assemble. It appears to be a challenging question whether these results–or ours–hold more generally. Even the Sierpinski carpet, a natural "next" fractal to consider after the Sierpinski triangle, is a case in point. Kautz and Lathrop [46] have shown that the Sierpinski carpet weakly self-assembles, but its strict self-assembly remains an open question. In fact, at the time of this writing, it is not known whether *any* nontrivial self-similar discrete fractal strictly self-assembles.

This has motivated the development of techniques to approximate self-similar fractals with strict self-assembly. The only previously known technique, introduced by Lathrop, Lutz, and Summers [51], and later generalized by Patitz and Summers [62], enables strict self-assembly of the intended fractal structure by adding communication fibers that shift successive stages of the fractal. However, this results in a structure that only visually resembles, but does not contain, the intended fractal structure.

In this paper we address a quantitative question: given that the Sierpinski triangle $\mathbf{S}$ cannot strictly self-assemble, how closely can strict self-assembly approximate $\mathbf{S}$? That is, if X is a set that *does* strictly self-assemble, how small can the fractal dimension–a measure of how completely a fractal fills space–of the symmetric difference $X \Delta \mathbf{S}$ be? Our first main theorem says that the fractal dimension of $X \Delta \mathbf{S}$ is at least the fractal dimension of $\mathbf{S}$.

To gain further insight, we restrict our attention to subsets of $\mathbf{S}$ and show that here the

limitation is even more severe. Any subset of the Sierpinski triangle that strictly self-assembles must have fractal dimension 0 or 1. Roughly speaking, the axes that bound $\mathbf{S}$ form the largest subset of $\mathbf{S}$ that strictly self-assembles. Hence, $\mathbf{S}$ cannot even be approximated "closely" with strict self-assembly.

Our second main theorem shows that our first main theorem is tight, even when restricted to supersets of $\mathbf{S}$. To prove this we demonstrate the existence of a set X with the following three properties.

(1) $\mathbf{S} \subseteq X$.

(2) The fractal dimension of $X \Delta \mathbf{S}$ is the fractal dimension of $\mathbf{S}$.

(3) X strictly self-assembles in the Tile Assembly Model.

What we have achieved here is a means of *fibering* $\mathbf{S}$ *in place*, i.e., adding the needed communication fibers (the set $X \setminus \mathbf{S}$) without disturbing the set $\mathbf{S}$. To the best of our knowledge, this is the first such construction for a self-similar fractal.

The local determinism method of Soloveichik and Winfree [76] is a common technique for proving a TAS is directed. However, the TAS in the proof of our second main theorem uses a blocking technique that prevents it from being locally deterministic. We thus introduce *conditional determinism*, a generalization of local determinism, to verify this TAS is directed.

The proof techniques used here, along with our blocking technique (and thus our generalization of local determinism), are likely to be useful in the design and analysis of other tile assembly systems that approximate self-similar fractals. Our fibering technique may be a useful example for other contexts where one seeks to enhance the "internal bandwidth" of a set in a distortion-free manner.

The self-assembly of large complex systems is a long-term objective of nanotechnology [64]. DNA tile assembly may–at least in its present form–be too fault-prone to directly achieve this objective. Nevertheless DNA tile assembly and the Tile Assembly Model have given us the beginnings of a general theory of self-assembly, its potentialities, and its limitations. We hope that our results lead to a more general understanding of the self- assembly and approximate self-assembly of fractal structures.

5.2 Preliminaries

5.2.1 Notation and Terminology

We now review finite-tree depth [51]. Let $G=(V,E)$ be a graph and let $D \subseteq V$. For $r \in V$, the *D-r-rooted subgraph* of G is the graph $G_{D,r}=(V_{D,r}, E_{D,r})$, where

$$V_{D,r} = \{v \in V \mid r \text{ is on every path from } v \text{ to (any vertex in) } D\}$$

and $E_{D,r} = E \cap [V_{D,r}]^2$. A *$D$-subtree* of G is a rooted tree B with root $r \in V$ such that $B = G_{D,r}$. The *finite-tree depth* of G *relative to* D is

$$\text{ft}-\text{depth}_D(G) = \sup\{\text{depth}(B) \mid B \text{ is a finite } D\text{-subtree of } G\}.$$

Intuitively, given a set D of vertices of G (which is in practice the domain of the seed assembly), the D-subtree of G is a rooted tree in G that consists of all vertices of G that lie at or on the far side of the root from D.

5.2.2 The Sierpinski Triangle

The Sierpinski triangle, a.k.a. the Sierpinski gasket or the Sierpinski sieve, is a self-similar fractal named after the Polish mathematician Waclaw Sierpiński who first described it [75]. It is formed by starting with a solid triangle and removing the middle fourth. This process is continued ad infinitum on all remaining triangles. See Figure 5.1 for an illustration of the this process.

This continuous version of the Sierpinski triangle is bounded and has the same detail at arbitrarily small scales. Since the TAM models the bottom-up self-assembly of tiles, which are discrete objects, structures that self-assemble in the TAM are fundamentally discrete. Therefore, we shall focus on the strict self-assembly of a discrete version of the Sierpinski triangle that is unbounded and has the same detail at arbitrarily large scales.

Figure 5.1: The first five stages of the continuous Sierpinski triangle.

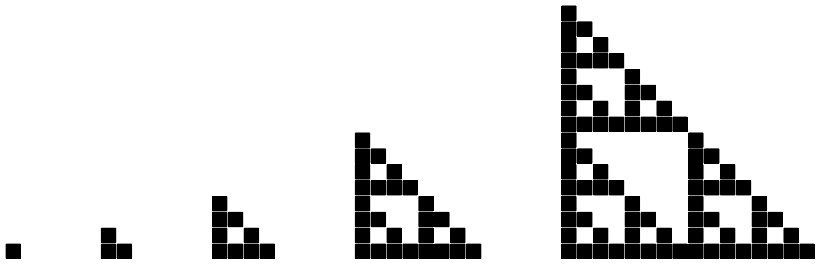


Figure 5.2: Stages 0 through 4 of the discrete Sierpinski triangle.

Formally, the discrete Sierpinski triangle is a set of points in $\mathbb{Z}^2$. Let $V = \{(1,0), (0,1)\}$ and define the sets $\mathbf{S}_0, \mathbf{S}_1, \ldots$ by the recursion

$$\begin{aligned} \mathbf{S}_0 &= \{(0,0)\}, \text{ and} \\ \mathbf{S}_{i+1} &= \mathbf{S}_i \cup (\mathbf{S}_i + 2^i V), \end{aligned} \tag{5.1}$$

where $A + cB = \{\vec{m} + c\vec{n} \mid \vec{m} \in A \text{ and } \vec{n} \in B\}$. The *discrete Sierpinski triangle* is the set

$$\mathbf{S} = \bigcup_{i=0}^{\infty} \mathbf{S}_i. \tag{5.2}$$

We often refer to $\mathbf{S}_i$ as the i^{th} stage of $\mathbf{S}$. Note that $\mathbf{S}$ can also be defined as the nonzero residues modulo 2 of Pascal's triangle [9]. It is also a numerically self-similar fractal [46]. See Figure 5.2 for an illustration.

Using equation 5.1 it is easy to give a formula for the cardinality of the i^{th} stage of $\mathbf{S}$.

Observation 4. *For each $n \in \mathbb{N}$, $|\mathbf{S}_n| = 3^n$.*

Then, using Observation 4 and Equations (4.1) and (5.1), we can easily calculate the ζ-dimension of $\mathbf{S}$.

Observation 5. $\text{Dim}_\zeta(\mathbf{S}) = \log_2 3$.

Winfree [84] proved that $\mathbf{S}$ weakly self-assembles in the TAM. Rothemund, Papadakis, and Winfree [66] later achieved a molecular implementation of this self-assembly. More recently, Lathrop, Lutz, and Summers [51] proved that $\mathbf{S}$ cannot strictly self-assemble in the TAM.

Theorem 25 (Lathrop, Lutz, and Summers [51])**.** $\mathbf{S}$ *cannot strictly self-assemble in the Tile Assembly Model.*

5.3 Limitations on Approximating the Sierpinski Triangle

In this section we present our first main theorem. We show that every set that strictly self-assembles disagrees with $\mathbf{S}$ on a set with ζ-dimension at least that of $\mathbf{S}$. We then show that for subsets of $\mathbf{S}$, the limitation is even more severe.

Our lower bound is proven by establishing a bound on the number of tile types needed for the self-assembly of $\mathbf{S}_n$ (Lemma 26). We then calculate the ζ-dimension of the symmetric difference of a terminal assembly with $\mathbf{S}_n$ (Theorem 26) using a recursive argument, and show that it works out to the ζ-dimension of the $\mathbf{S}$.

We first show that in any τ-stable assembly in the configuration of some stage of $\mathbf{S}$, each tile that attaches during the assembly sequence does so by interacting with exactly one tile. Hence, all of the interactions are between pairs of tiles that interact with a strength of at least the temperature τ on their abutting side.

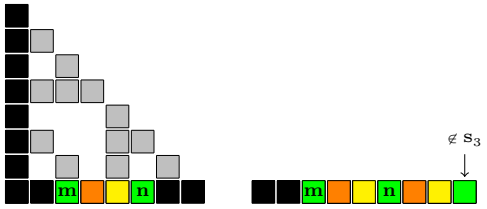
Lemma 24. *Let T be a set of tile types, $\tau, n \in \mathbb{N}$, and $\alpha \in \mathcal{A}_T^\tau$ such that $\operatorname{dom}\alpha = \mathbf{S}_n$. For each $\vec{m} \in \operatorname{dom}\alpha$ and $\vec{u} \in U_2$, if $\vec{m}+\vec{u} \in \operatorname{dom}\alpha$, then $\alpha(\vec{m})(\vec{u}) = \alpha(\vec{m}+\vec{u})(-\vec{u})$ and $\operatorname{str}_{\alpha(\vec{m})}(\vec{u}) \geq \tau$.*

Proof. Assume the hypothesis with T, τ, α, and n as witness. Let $\vec{m} \in \operatorname{dom}\alpha$ and $\vec{u} \in U_2$ such that $\vec{m}+\vec{u} \in \operatorname{dom}\alpha$. It suffices to show that $\operatorname{str}_\alpha(\vec{m}, \vec{m}+\vec{u}) \geq \tau$. Let $G_\alpha = (V, E, \beta)$ be the binding graph of α. Note that since $\operatorname{dom}\alpha = \mathbf{S}_n$, (V, E) is a tree rooted at the origin and since α is τ-stable, $\beta_{(V,E)} \geq \tau$. So, it suffices to show that $\beta((\vec{m}, \vec{m}+\vec{u})) \geq \tau$.

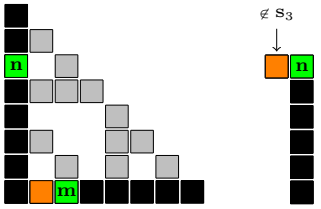
Since (V, E) is a tree, and $\vec{m}$ and $\vec{m}+\vec{u}$ are adjacent in (V, E), either $\vec{m}$ is on the path from the origin to $\vec{m}+\vec{u}$ or $\vec{m}+\vec{u}$ is on the path from the origin to $\vec{m}$. Without loss of generality, assume $\vec{m}$ is on the path from the origin to $\vec{m}+\vec{u}$ (otherwise, the the theorem holds for $\vec{m}' = \vec{m}+\vec{u}$ and $\vec{u}' = -\vec{u}$). Let $C = (C_0, C_1)$ be the unique cut of G such that

$$C_1 = \{\vec{n} \in \operatorname{dom}\alpha \mid \vec{m}+\vec{u} \text{ is on a path in } G \text{ from the origin to } \vec{n}\}, \text{ and}$$
$$C_0 = V \setminus C_1.$$

Then, $\vec{m} \in C_0$ and $\vec{m}+\vec{u} \in C_1$. Furthermore, since (V, E) is a tree, $(\vec{m}, \vec{m}+\vec{u}) \in E$ is the unique edge across C. But then, $\beta_C = \beta((\vec{m}, \vec{m}+\vec{u}))$, and since $\beta_C \geq \beta_{(V,E)} \geq \tau$, $\beta((\vec{m}, \vec{m}+\vec{u})) \geq \tau$. □



(a) Case 1. The figure on the left shows an example α where $\vec{m} = (2, 0)$ and $\vec{n} = (5, 0)$. $\alpha(\vec{m})$ and $\alpha(\vec{n})$ are green, $\alpha(\vec{m} + \vec{u}_{\mathrm{E}})$ is orange, and $\alpha(\vec{m} + 2\vec{u}_{\mathrm{E}})$ is yellow. The figure on the right shows the β corresponding to this α.



(b) Case 3. The figure on the left shows an example α where $\vec{m} = (2, 0)$ and $\vec{n} = (0, 5)$. The tile $\alpha(\vec{m} + \vec{u}_{\mathrm{W}})$ is orange. The figure on the right shows the β corresponding to this α.

Figure 5.3: Illustrating the proof of Lemma 25 for $n = 3$.

Lemma 24 allows us to show that all of the boundary tiles of a terminal assembly in the configuration of some stage of $\mathbf{S}$ are each a unique tile type via a pumping argument used in the proof of Lemma 25.

Lemma 25. *If $\mathbf{S}_n$ strictly self-assembles in a TAS (T, σ, τ), then $|T| \geq 2^{n+1} - 1$.*

Proof. Assume the hypothesis with $n \in \mathbb{N}$ and TAS $\mathcal{T} = (T, \sigma, \tau)$ as witness. Let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. If $n = 0$ the lemma is trivially true, so assume $n > 1$. Let $A = A_h \cup A_v$, where $A_h = \{(i, 0) \mid 0 \leq i < 2^n\}$ and $A_v = \{(0, i) \mid 0 \leq i < 2^n\}$. Conceptually, A_h (and A_v) represent the left (and right) boundary of $\mathbf{S}_n$. Let $T_A = \{\alpha(\vec{m}) \mid \vec{m} \in A\}$ be the set of all tile types placed at locations in A. Clearly, $T_A \subseteq T$, so it suffices to show that $|T_A| \geq 2^{n+1} - 1$. Suppose $|T_A| < 2^{n+1} - 1$. By (5.1) and $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, $\vec{m} \in \operatorname{dom} \alpha$ for all $\vec{m} \in A$. Then, there exist a $\vec{m}, \vec{n} \in A$ such that $\vec{m} \neq \vec{n}$ and $\alpha(\vec{m}) = \alpha(\vec{n})$. Either $\vec{m}, \vec{n} \in A_h$, $\vec{m}, \vec{n} \in A_v$, or $\vec{m} \in A_h \setminus \{\vec{0}\}$ and $\vec{n} \in A_v \setminus \{\vec{0}\}$. In each case we show that $\mathbf{S}_n$ does not strictly self-assemble in $\mathcal{T}$.

Case 1. Suppose $\vec{m}, \vec{n} \in A_h$. Without loss of generality, let $\vec{m} = (i, 0)$ and $\vec{n} = (j, 0)$ where $0 \leq i < j < 2^n$. Let β be the unique T-configuration such that for all $\vec{k} = (k_1, k_2) \in \mathbb{N}$,

$$\beta(\vec{k}) = \begin{cases} \uparrow & \text{if } k_1 > 2^n \text{ or } k_2 \neq 0 \\ \alpha(\vec{k}) & \text{if } k_1 < j \\ \alpha(\vec{m} + ((k_1 - j) \bmod (j - i), 0)) & \text{otherwise.} \end{cases}$$

See Figure 5.3a for an illustration. By (5.1) and $j > 0$, $\vec{n} + \vec{u}_{\mathrm{W}} \in \operatorname{dom} \alpha$. So, by Lemma 24, $\alpha(\vec{n})(\vec{u}_{\mathrm{W}}) = \alpha(\vec{n} + \vec{u}_{\mathrm{W}})(\vec{u}_{\mathrm{E}})$ and $\operatorname{str}_{\alpha(\vec{n})}(\vec{u}_{\mathrm{W}}) \geq \tau$. But since $\alpha(\vec{n}) = \alpha(\vec{m})$,

$\alpha(\vec{m})(\vec{u}_{\mathrm{W}}) = \alpha(\vec{n} + \vec{u}_{\mathrm{W}})(\vec{u}_{\mathrm{E}})$ and $\mathrm{str}_{\alpha(\vec{m})}(\vec{u}_{\mathrm{W}}) \geq \tau$. So, $\beta \in \mathcal{A}[\mathcal{T}]$. Then, there exists a $\gamma \in \mathcal{A}_{\square}[\mathcal{T}]$ such that $\beta \sqsubseteq \gamma$ and since $\beta((2^n, 0))\downarrow$, $\gamma((2^n, 0))\downarrow$. But $(2^n, 0) \notin \mathbf{S}_n$. So, $\mathbf{S}_n$ does not strictly self-assemble in $\mathcal{T}$.

Case 2. The case for $\vec{m}, \vec{n} \in A_v$ is similar to Case 1.

Case 3. Suppose $\vec{m} \in A_h \setminus \{\vec{0}\}$ and $\vec{n} \in A_v \setminus \{\vec{0}\}$. Let $\vec{m} = (i, 0)$ and $\vec{n} = (0, j)$, where $i, j \in \{1, \ldots, 2^n - 1\}$. Let β be the unique T-configuration such that for all $\vec{k} = (k_1, k_2) \in \mathbb{N}$,

$$\beta(\vec{k}) = \begin{cases} \alpha(\vec{k}) & \text{if } k_1 = 0 \text{ and } k_2 \leq j \text{ or } k_2 = 0 \text{ and } k_1 \leq i \\ \alpha(\vec{m} + \vec{u}_{\mathrm{W}}) & \text{if } k_1 = -1 \text{ and } k_2 = j \\ \uparrow & \text{otherwise.} \end{cases}$$

See Figure 5.3b for an illustration. By (5.1) and $i > 0$, $\vec{m} + \vec{u}_{\mathrm{W}} \in \mathrm{dom}\,\alpha$. So, by Lemma 24, $\alpha(\vec{m})(\vec{u}_{\mathrm{W}}) = \alpha(\vec{m} + \vec{u}_{\mathrm{W}})(\vec{u}_{\mathrm{E}})$ and $\mathrm{str}_{\alpha(\vec{m})}(\vec{u}_{\mathrm{W}}) \geq \tau$. But since $\alpha(\vec{m}) = \alpha(\vec{n})$, $\alpha(\vec{n})(\vec{u}_{\mathrm{W}}) = \alpha(\vec{m} + \vec{u}_{\mathrm{W}})(\vec{u}_{\mathrm{E}})$ and $\mathrm{str}_{\alpha(\vec{n})}(\vec{u}_{\mathrm{W}}) \geq \tau$. So, $\beta \in \mathcal{A}[\mathcal{T}]$. Then, there exists a $\gamma \in \mathcal{A}_{\square}[\mathcal{T}]$ such that $\beta \sqsubseteq \gamma$ and since $\beta((-1, j))\downarrow$, $\gamma((-1, j))\downarrow$. But $(-1, j) \notin \mathbf{S}_n$. So, $\mathbf{S}_n$ does not strictly self-assemble in $\mathcal{T}$. □

Even if we only require that $\mathbf{S}_n$ appear *somewhere* in the terminal assembly (not necessarily at the origin), we still have an exponential lower bound on the minimum number of tile types needed. To show this we use the *ruler function* $\rho : \mathbb{Z}^+ \to \mathbb{N}$ defined by the recurrence $\rho(2k+1) = 0$ and $\rho(2k) = \rho(k) + 1$ for all $k \in \mathbb{N}$. The value of $\rho(n)$ is the exponent of the largest power of 2 that divides n, or equivalently, $\rho(n)$ is the number of 0's lying to the right of the rightmost 1 in the binary representation of n [32]. Now, for each $i \in \mathbb{N}$, the width of the longest horizontal bar rooted at $(0, i)$ and the height of the tallest vertical bar rooted at $(i, 0)$ in $\mathbf{S}$ is $2^{\rho(i)} - 1$ [51].

Lemma 26. *Let $n \in \mathbb{N}$ and $\vec{m} \in \mathbb{Z}^2$. If $\mathcal{T} = (T, \sigma, \tau)$ is a TAS such that for every $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$, $\mathrm{dom}\,\alpha \cap (\vec{m} + \{0, \ldots, 2^n - 1\}^2) = \vec{m} + \mathbf{S}_n$, then $|T| \geq 2^n - 2$.*

Proof. Assume the hypothesis with $n \in \mathbb{N}$, $\vec{m} = (m_1, m_2) \in \mathbb{Z}^2$, and TAS $\mathcal{T} = (T, \sigma, \tau)$ as witness. Let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. Suppose $|T| < 2^n - 2$. We will construct a TAS $\mathcal{T}' = (T', \sigma', \tau)$ in which S_n strictly self-assembles but with $|T'| \leq 2^{n+1} - 2$, thus contradicting Lemma 25.

The TAS $\mathcal{T}' = (T', \sigma', \tau)$ is constructed as follows. We will assume that none of the glue colors on tiles in T use digits. Thus, we can safely assume that the following algorithm doesn't introduce any *unwanted* interactions.

(1) For every $\vec{n} \in \vec{m} + \left\{1, 2, \ldots, 2^{n-1} - 1\right\}^2$, if $\alpha(\vec{n}) \downarrow$, then $\alpha(\vec{n}) \in T'$.

(2) Let $i = 2^{n-1}$. There exists tiles $h_i, v_i \in T'$ such that

$$h_i(\vec{u}_{\mathrm{N}}) = v_i(\vec{u}_{\mathrm{N}}) = \alpha(\vec{m} + (2^{n-1}, 1))(\vec{u}_{\mathrm{S}}),\ v_i(\vec{u}_{\mathrm{S}}) = h_i(\vec{u}_{\mathrm{W}}) = (i-1, \tau),$$
$$h_i(\vec{u}_{\mathrm{E}}) = v_i(\vec{u}_{\mathrm{E}}) = \alpha(\vec{m} + (1, 2^{n-1}))(\vec{u}_{\mathrm{W}}),\ \text{and } v_i(\vec{u}_{\mathrm{W}}) = h_i(\vec{u}_{\mathrm{S}}) = (0, 0).$$

(3) For each $0 < i < 2^{n-1}$, there exists tiles $v_i, h_i \in T'$ such that

$$v_i(\vec{u}_{\mathrm{W}}) = h_i(\vec{u}_{\mathrm{S}}) = (0,0),\quad v_i(\vec{u}_{\mathrm{E}}) = \begin{cases} \alpha(\vec{m} + (2^{n-1} - 2^{\rho(i)} + 1, 2^{n-1})) & \rho(i) > 0 \\ (0,0) & \rho(i) = 0, \end{cases}$$

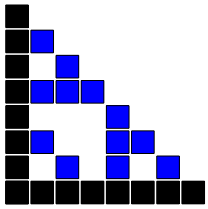
$$v_i(\vec{u}_{\mathrm{N}}) = h_i(\vec{u}_{\mathrm{E}}) = (i, \tau),\quad h_i(\vec{u}_{\mathrm{N}}) = \begin{cases} \alpha(\vec{m} + (2^{n-1}, 2^{n-1} - 2^{\rho(i)} + 1)) & \rho(i) > 0 \\ (0,0) & \rho(i) = 0, \end{cases}$$

and $v_i(\vec{u}_{\mathrm{S}}) = h_i(\vec{u}_{\mathrm{W}}) = (i-1, \tau)$.

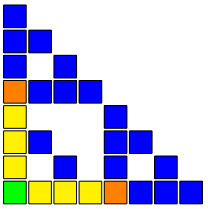
(4) There exists a tile $\sigma' \in T'$ such that

$$\sigma'(\vec{u}_{\mathrm{N}}) = \sigma'(\vec{u}_{\mathrm{E}}) = (0, \tau) \text{ and } \sigma'(\vec{u}_{\mathrm{S}}) = \sigma'(\vec{u}_{\mathrm{W}}) = (0,0).$$

Each tile type added to T' in step (1) is also a tile type in T, so, by assumption, we add at most $2^n - 3$ tile types to T' in step (1). Then, 2 tile types are added to T' in step (2),



(a) The assembly α for some arbitrary $\vec{m}$. Tile types added to T' in step (1) are blue.



(b) The assembly α' such that $\operatorname{dom}\alpha' = \mathbf{S}_n$. Tile types added to T' in step (2) are orange, step (3) yellow, and step (4) green.

Figure 5.4: Illustrating the proof of Lemma 26 for $n = 3$.

$2^n - 2$ tile types are added to T' in step (3). and 1 tile type is added to T' in step (4). Thus, $|T'| \leq 2^{n+1} - 2$. But, by using equation (5.1) and the ruler function properties, it is easy to verify that $\mathbf{S}_n$ strictly self-assembles in $\mathcal{T}'$. This contradicts Lemma 25. Thus, no such TAS exists. □

We now have the necessary machinery to prove our first main theorem which says that every set that strictly self-assembles disagrees with $\mathbf{S}$ on a set with fractal dimension at least that of $\mathbf{S}$. Hence, $\mathbf{S}$ cannot even be approximated closely with strict self-assembly.

Theorem 26. *If $X \subseteq \mathbb{Z}^2$ strictly self-assembles, then $\mathrm{Dim}_\zeta(X \Delta \mathbf{S}) \geq \mathrm{Dim}_\zeta(\mathbf{S})$.*

Proof. Assume the hypothesis with $X \subseteq \mathbb{Z}^2$ and TAS $\mathcal{T} = (T, \sigma, \tau)$ as witness. Let $V = \{(0,0),(0,1),(1,0)\}$ and $n = \lceil \log_2(|T|+2)+1 \rceil$. Since X strictly self-assembles in $\mathcal{T}$, for every $\alpha \in \mathcal{A}_\square[\mathcal{T}]$, $X = \mathrm{dom}\,\alpha$. Let $d : \mathbb{Z}^2 \times \mathbb{N} \to \mathbb{N}$ where

$$d(\vec{m}, k) = \left| \left(X \cap \left(\vec{m} + \{0, \ldots, 2^{n+k} - 1\}^2\right)\right) \Delta \left(\vec{m} + \mathbf{S}_{n+k}\right) \right| \tag{5.3}$$

for all $\vec{m} \in \mathbb{Z}^2$ and $k \in \mathbb{N}$. Then, by (5.1), $d(\vec{m}, k) \geq \sum_{\vec{v} \in V} d(\vec{m} + 2^{n+k-1}\vec{v}, k-1)$. Since $|T| < 2^n - 2$, by Lemma 26, for all $\vec{m} \in \mathbb{Z}^2$, $X \cap \left(\vec{m} + \{0, \ldots, 2^n - 1\}^2\right) \neq \vec{m} + \mathbf{S}_n$. So, for all $\vec{m} \in \mathbb{Z}^2$, $d(\vec{m}, 0) \geq 1$. Then, the recurrence solves to

$$d(\vec{m}, k) \geq 3^k \tag{5.4}$$

for all $\vec{m} \in \mathbb{Z}^2$. So,

$$\mathrm{Dim}_\zeta(X \Delta \mathbf{S}) \overset{(4.1)}{=} \limsup_{n \to \infty} \frac{\log |X_{[0,2^n)}|}{n} \overset{(5.3)}{=} \limsup_{k \to \infty} \frac{\log d(\vec{0}, k)}{n+k} \overset{(5.4)}{=} \limsup_{k \to \infty} \frac{\log 3^k}{n+k} = \log_2 3.$$

By Observation 5, the theorem holds. □

To gain further insight, we now consider the strict self-assembly of subsets of $\mathbf{S}$, and show that here the limitation is even more severe. We first give an upper bound on the number of tiles located within a given distance of the seed tile in any strict self-assembly of a subset of $\mathbf{S}$. We use the a theorem from [51] that for any structure to strictly self-assemble, the number of tile types used is at least the finite-tree depth of the structure.

Theorem 27 (Lathrop, Lutz, and Summers [51])**.** *If $X \subseteq \mathbb{Z}^2$ strictly self-assembles in a TAS (T, σ, τ), then $|T| \geq \text{ft}-\text{depth}_{\text{dom}\,\sigma}(G_X^{\#})$.*

It easily follows that in any strict self-assembly of a subset of $\mathbf{S}$, not too many tiles can be placed far from the boundary.

Corollary 7. *If $X \subseteq \mathbf{S}$ strictly self-assembles in a TAS (T, σ, τ), then for all $\vec{m} = (m_1, m_2) \in \mathbb{Z}^2$ such that $m_1 \geq |T|$ and $m_2 \geq |T|$, $\vec{m} \notin X$.*

Lemma 27. *If $X \subseteq \mathbf{S}$ strictly self-assembles in a TAS (T, σ, τ), then for every $n \in \mathbb{N}$, $|X_{[0,n]}| \leq 2|T|(n+1)$.*

Proof. Assume the hypothesis with $X \subseteq \mathbb{Z}^2$ and $\mathcal{T} = (T, \sigma, \tau)$ as witness. Let $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$ and let $n \in \mathbb{N}$. If $n \leq |T|$ the theorem is trivially true, so assume $n > |T|$. Let $A = \{0, \ldots, |T|-1\}^2$, $B = \{0, \ldots, |T|-1\} \times \{|T|, \ldots, n\}$, $C = \{|T|, \ldots, n\} \times \{0, \ldots, |T|-1\}$, and $D = \{|T|, \ldots, n\}^2$. It is clear that A, B, C, D is a partition of $\{0, \ldots, n\}^2$. Then,

$$
\begin{aligned}
|X_{[0,n]}| &= |\{0, \ldots, n\}^2 \cap \text{dom}\,\alpha| && \text{since } X \subseteq \mathbb{N}^2 \\
&= |A \cap \text{dom}\,\alpha| + |B \cap \text{dom}\,\alpha| + |C \cap \text{dom}\,\alpha| && \text{by Corollary 7} \\
&\leq |T|^2 + 2|T|(n - |T| + 1) \\
&\leq 2|T|(n+1).
\end{aligned}
$$

□

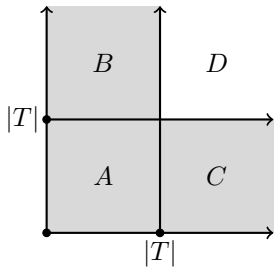


Figure 5.5: Illustrating the partition used in the proof of Lemma 27.

Theorem 28. *If $X \subseteq \mathbf{S}$ strictly self-assembles, then $\text{Dim}_{\zeta}(X) \in \{0, 1\}$.*

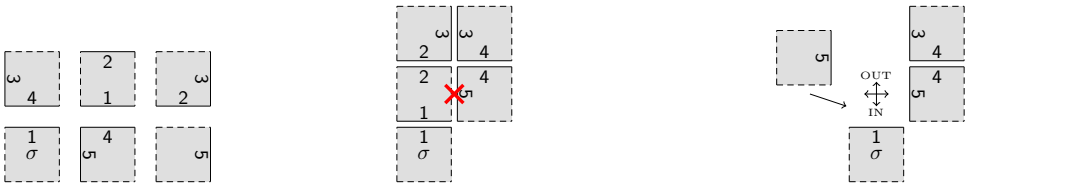
Proof. Assume the hypothesis with $X \subseteq \mathbf{S}$ and TAS (T, σ, τ) as witness. By Lemma 27, $|X_{[0,n]}| \leq 2|T|(n+1)$. Then, $\text{Dim}_{\zeta}(X) \leq 1$. But, the binding graph of any $\alpha \in \mathcal{A}_T^{\tau}$ must be

connected and any infinite connected structure has ζ-dimension at least 1. It follows that either $\mathrm{Dim}_\zeta(X) = 1$ or X is finite, in which case X has ζ-dimension 0. So, $\mathrm{Dim}_\zeta(X) \in \{0, 1\}$. □

Note that boundary of $\mathbf{S}$ is a subset of $\mathbf{S}$ that strictly self-assembles and has ζ-dimension 1. A single tile placed at the origin is a subset of $\mathbf{S}$ that strictly self-assembles and has ζ-dimension 0. Hence, Theorem 28 is trivially tight.

5.4 Conditional Determinism

The method of local determinism introduced by Soloveichik and Winfree [76] is a common technique for showing that a TAS is directed. However, there exists very natural constructions that are directed but not locally deterministic. Consider the TAS $\mathcal{T}_B = (T_B, \sigma_B, 1)$ of Figure 5.6. Clearly, there is only one assembly sequence $\vec{\alpha}$ in $\mathcal{T}_B$ such that $\mathrm{res}\,\vec{\alpha}$ is terminal. Hence, $\mathcal{T}_B$ is directed. However, $\vec{\alpha}$ fails condition (2) of local determinism at the location $(0, 1)$. The culprit is the blocking technique used by this TAS which is marked by a red X in Figure 5.6b. Since $\vec{\alpha}$ is the only possible locally deterministic assembly sequence in $\mathcal{T}_B$, then $\mathcal{T}_B$ is not a locally deterministic TAS. Thus, new techniques are needed to show this TAS is directed.



(a) The tile set T_B.

(b) $\mathrm{res}\,(\vec{\alpha})$. The site of the blocking is marked with a red X.

(c) $\vec{\alpha}$ fails condition (2) of local determinism for the location $(0, 1)$.

Figure 5.6: The TAS $\mathcal{T}_B = (T_B, \sigma_B, 1)$ which uses a blocking technique.

In this section we give sufficient conditions for proving such a TAS is directed. First, we introduce some new notation. For $\vec{m}, \vec{n} \in \mathbb{Z}^2$, if $\vec{m} \prec_{\vec{\alpha}} \vec{n}$ for every assembly sequence $\vec{\alpha}$ in a TAS $\mathcal{T}$, then we say $\vec{m}$ *precedes* $\vec{n}$ *in* $\mathcal{T}$, and we write $\vec{m} \prec_{\mathcal{T}} \vec{n}$. For each $\vec{m} \in \mathbb{Z}^2$, we define the set

$$\mathrm{DEP}^{\mathcal{T}}(\vec{m}) = \{\vec{u} \in U \mid \vec{m} \prec_{\mathcal{T}} \vec{m} + \vec{u}\}.$$

Now, let $\mathcal{T}$ be a TAS, $\vec{\alpha}$ an assembly sequence in $\mathcal{T}$, and $\alpha = \operatorname{res}\vec{\alpha}$. Then, $\vec{\alpha}$ is *conditionally deterministic* if the following three conditions hold.

(1) For all $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$, $\sum_{\vec{u}\in \mathrm{IN}^{\vec{\alpha}}(\vec{m})} \mathrm{str}_{\alpha_{i_{\vec{\alpha}}(\vec{m})}}(\vec{m}, \vec{m}+\vec{u}) = \tau$.

(2) For all $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ and all $t \in T \setminus \{\alpha(\vec{m})\}$,

$$\vec{m} \notin \partial_t^{\tau}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + (\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})))))).$$

(3) $\partial^{\tau}\alpha = \emptyset$.

Note that conditions (1) and (3) are the same as in the definition of local determinism. Conceptually, (1) requires that each tile added in $\vec{\alpha}$ "just barely" binds to the existing assembly; (2) holds when the tiles at $\vec{m}$ and $\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}) + \mathrm{DEP}^{\mathcal{T}}(\vec{m})$ are removed from α, no other tile type can attach to the assembly at location $\vec{m}$; and (3) requires that α is terminal. A TAS is *conditionally deterministic* if it has a conditionally deterministic assembly sequence.

Our first theorem shows that conditional determinism is a weaker notion than local determinism.

Theorem 29. *Every locally deterministic TAS is conditionally deterministic.*

Proof. Let $\mathcal{T}$ be a locally deterministic TAS with $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ as witness. Let $\alpha = \operatorname{res}(\vec{\alpha})$. It suffices to show that $\vec{\alpha}$ is a conditionally deterministic assembly sequence. Since conditions (1) and (3) in the definitions of both local determinism and conditional determinism are the same, it suffices to show that condition (2) in the definition of conditional determism holds for $\vec{\alpha}$. Since $\vec{\alpha}$ is locally deterministic, by condition (2) of local determinism, for all $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ and all $t \in T \setminus \{\alpha(\vec{m})\}$,

$$\vec{m} \notin \partial_t^{\tau}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))).$$

Then, since $\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \subseteq \mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})$, it follows that for all $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ and all $t \in T \setminus \{\alpha(\vec{m})\}$,

$$\vec{m} \notin \partial_t^{\tau}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + (\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})))))).$$

Hence, $\vec{\alpha}$ is a conditionally deterministic assembly sequence in $\mathcal{T}$. □

We now show that although conditional determinism is weaker than local determinism, it is strong enough to show a TAS is directed.

Theorem 30. *Every conditionally deterministic TAS is directed.*

Proof. Our proof is similar to the proof in [76] that every locally deterministic TAS is directed. Let $\mathcal{T} = (T, \sigma, \tau)$ be a conditionally deterministic TAS with $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ as witness. Let $\alpha = \text{res}(\vec{\alpha})$ and note that $\alpha \in \mathcal{A}_{\square}[\mathcal{T}]$. To see that $\mathcal{T}$ is directed, it suffices to show that for all $\beta \in \mathcal{A}_{\square}[\mathcal{T}]$, $\beta \sqsubseteq \alpha$.

Let $\beta \in \mathcal{A}_{\square}[\mathcal{T}]$. Then, there is an assembly sequence $\vec{\beta} = (\beta_j \mid 0 \leq j < l)$ in $\mathcal{T}$ such that $\beta_0 = \sigma$ and $\beta = \text{res}(\vec{\beta})$. To see that $\beta \sqsubseteq \alpha$, it suffices to show that for each $0 \leq j < k$, the following conditions hold:

(1) $\text{IN}^{\vec{\beta}}(\text{dom}\,\beta_{j+1} \setminus \text{dom}\,\beta_j) = \text{IN}^{\vec{\alpha}}(\text{dom}\,\beta_{j+1} \setminus \text{dom}\,\beta_j)$, and

(2) $\beta(\text{dom}\,\beta_{j+1} \setminus \text{dom}\,\beta_j) = \alpha(\text{dom}\,\beta_{j+1} \setminus \text{dom}\,\beta_j)$.

Suppose there exists a $0 \leq j < k$ such that either condition (1) or condition (2) fails. Let i be the smallest such j. To prove the theorem, it suffices to show no such i exists. Let $\vec{b}_i = \text{dom}\,\beta_{i+1} \setminus \text{dom}\,\beta_i$. Consider any $\vec{u} \in \text{IN}^{\vec{\beta}}(\vec{b}_i)$. It is clear that $-\vec{u} \in \text{OUT}^{\vec{\beta}}(\vec{b}_i + \vec{u})$, so $-\vec{u} \notin \text{IN}^{\vec{\beta}}(\vec{b}_i + \vec{u})$. Either $\vec{b}_i + \vec{u} \in \text{dom}\,\sigma$ or there exists an $h < i$ such that $\vec{b}_h = \vec{b}_i + \vec{u}$.

Case 1. Suppose $\vec{b}_i + \vec{u} \in \text{dom}\,\sigma$. Then, since both $\vec{\alpha}$ and $\vec{\beta}$ are assembly sequences in $\mathcal{T}$, $\text{IN}^{\vec{\beta}}(\vec{b}_i + \vec{u}) = \text{IN}^{\vec{\alpha}}(\vec{b}_i + \vec{u}) = \emptyset$. Then, $-\vec{u} \notin \text{IN}^{\vec{\alpha}}(\vec{b}_i + \vec{u})$. So, $\vec{u} \notin \text{OUT}^{\vec{\alpha}}(\vec{b}_i)$. Also, $i_{\vec{\beta}}(\vec{b}_i + \vec{u}) = 0$, so $\vec{b}_i \not\prec_{\mathcal{T}} \vec{b}_i + \vec{u}$. Then, $\vec{u} \notin \text{DEP}^{\mathcal{T}}(\vec{b}_i)$.

Case 2. Suppose there exists an $h < i$ such that $\vec{b}_h = \vec{b}_i + \vec{u}$. Then, by condition (1), $\text{IN}^{\vec{\beta}}(\vec{b}_i + \vec{u}) = \text{IN}^{\vec{\alpha}}(\vec{b}_i + \vec{u})$. Then, $-\vec{u} \notin \text{IN}^{\vec{\alpha}}(\vec{b}_i + \vec{u})$. So, $\vec{u} \notin \text{OUT}^{\vec{\alpha}}(\vec{b}_i)$. Also, $h \prec_{\vec{\beta}} i$, so $\vec{b}_i \not\prec_{\mathcal{T}} \vec{b}_i + \vec{u}$. Then, $\vec{u} \notin \text{DEP}^{\mathcal{T}}(\vec{b}_i)$.

In either case,

$$\text{IN}^{\vec{\beta}}(\vec{b}_i) \cap (\text{OUT}^{\vec{\alpha}}(\vec{b}_i) \cup \text{DEP}^{\mathcal{T}}(\vec{b}_i)) = \emptyset. \tag{i}$$

Since for all $\vec{m} \in \text{dom}\, \beta_i, \beta_i(\vec{m}) = \alpha(\vec{m})$, then for all $\vec{u} \in U_2$,

$$\text{str}_{\beta_{i+1}}(\vec{b}_i, \vec{b}_i + \vec{u}) \leq \text{str}_{\alpha}(\vec{b}_i, \vec{b}_i + \vec{u}). \tag{ii}$$

Then, by (i) and (ii),

$$\Sigma_{\vec{u} \in \text{IN}^{\vec{\beta}}(\vec{b}_i)} \text{str}_{\beta_{i+1}}(\vec{b}_i, \vec{b}_i + \vec{u}) \leq \Sigma_{\vec{u} \in U_2 \setminus (\text{OUT}^{\vec{\alpha}}(\vec{b}_i) \cup \text{DEP}^{\mathcal{T}}(\vec{b}_i))} \text{str}_{\alpha}(\vec{b}_i, \vec{b}_i + \vec{u}).$$

But, by property (2) of conditional determinism, the only type of tile that can attach to β_i at location $\vec{b}_i$ is $\alpha(\vec{b}_i)$. Thus, $\beta(\vec{b}_i) = \alpha(\vec{b}_i)$.

So it must be the case that $\text{IN}^{\vec{\beta}}(\vec{b}_i) \neq \text{IN}^{\vec{\alpha}}(\vec{b}_i)$. By property (1) of conditional determinism, there must be some $\vec{u} \in \text{IN}^{\vec{\beta}}(\vec{b}_i) \setminus \text{IN}^{\vec{\alpha}}(\vec{b}_i)$. Since $\vec{u} \in \text{IN}^{\vec{\beta}}(\vec{b}_i)$, $\vec{b}_i + \vec{u} \in \text{dom}\, \beta_i$, so $\beta(\vec{b}_i + \vec{u}) = \alpha(\vec{b}_i + \vec{u})$. We've already established that $\beta(\vec{b}_i) = \alpha(\vec{b}_i)$. So, By property (2) of conditional determinism, it must be the case that $i_{\vec{\alpha}}(\vec{b}_i + \vec{u}) > i_{\vec{\alpha}}(\vec{b}_i)$. So, $\vec{u} \notin \text{IN}^{\vec{\alpha}}(\vec{b}_i)$. But then $-\vec{u} \in \text{IN}^{\vec{\alpha}}(\vec{b}_i + \vec{u})$, and so $\vec{u} \in \text{OUT}^{\vec{\alpha}}(\vec{b}_i)$. But, by (i), this is impossible. Therefore, no such i exists. □

It is now a straightforward task to show that the TAS of Figure 5.6 is directed.

5.5 Fibering the Sierpinski Triangle in Place

In this section we present our second main theorem. We construct a TAS in which a superset of $\mathbf{S}$ with the same ζ-dimension strictly self-assembles. Thus, our first main theorem is tight, even when restricted to supersets of $\mathbf{S}$. To prove this we define a new fractal, the *laced Sierpinski triangle*, denoted $\mathbf{L}$. We show that $\mathbf{S} \subseteq \mathbf{L}$, $\text{Dim}_{\zeta}(\mathbf{L} \Delta \mathbf{S}) = \text{Dim}_{\zeta}(\mathbf{S})$, and that $\mathbf{L}$ strictly self-assembles in the Tile Assembly Model.

Formally, the laced Sierpinski triangle is a set of points in $\mathbb{Z}^2$. Our goal is to define the sets $\mathbf{L}_0, \mathbf{L}_1, \ldots$ such that each $\mathbf{L}_i$ is the i^{th} stage in our construction of $\mathbf{L}$. We will break each $\mathbf{L}_i$ up into disjoint subsets representing the different "types" of fibers added to $\mathbf{S}$ that allow $\mathbf{L}$ to strictly self-assemble. Let $V = \{(0,1),(1,0)\}$, $W = \{(0,0)\} \cup V$, and

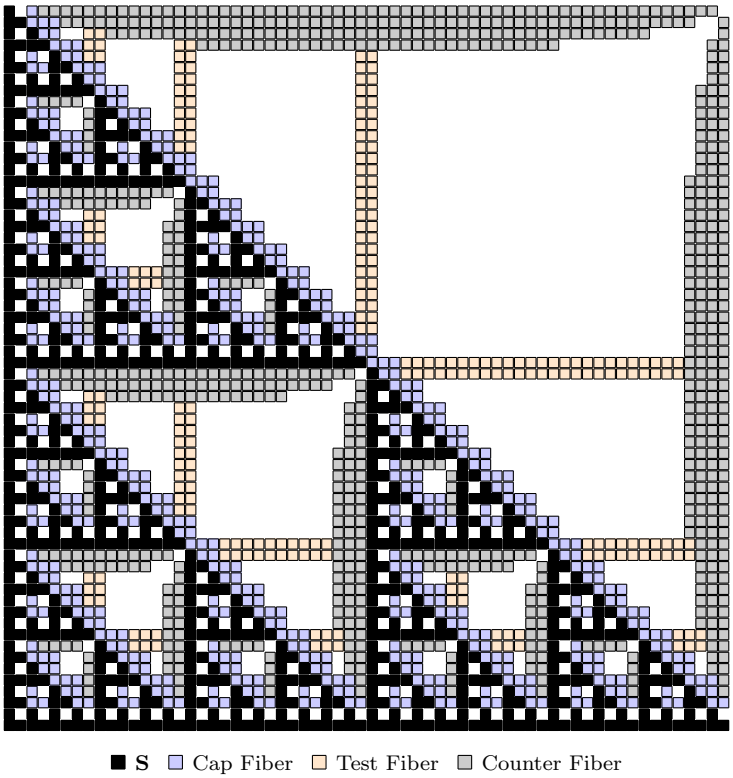


Figure 5.7: Stage 6 of the laced Sierpinski triangle.

$X = \{(0,2),(2,0),(-1,1),(1,-1)\} \cup W$. Then, we define the sets $C_0, C_1, \ldots$ by

$$C_i = \begin{cases} \emptyset & \text{if } i < 2 \\ 2^{i-1}(1,1) + W & \text{if } i = 2 \\ (2^{i-1}(1,1) + X) \cup \bigcup\limits_{w \in W} (2^{i-1}w + C_{i-1}) & \text{otherwise.} \end{cases} \tag{5.5}$$

Intuitively, each C_i is the set of *cap fibers* in $\mathbf{L}_i$. For each $i \in \mathbb{N}$, let

$$\Phi_i = \bigcup_{j=1}^{i-2} \bigcup_{k=2^j+1}^{2^i-2^j} \left\{(2^i - j, k), (k, 2^i - j)\right\}. \tag{5.6}$$

Note that $\Phi_i = \emptyset$ for $i < 3$. Then, we define the sets $N_0, N_1, \ldots$ by

$$N_i = \begin{cases} \emptyset & \text{if i} < 3 \\ \Phi_i \cup N_{i-1} \cup \bigcup\limits_{v \in V} (2^{i-1}v + (N_{i-1} - \Phi_{i-1})) & \text{otherwise.} \end{cases} \tag{5.7}$$

Intuitively, N_i is the set of *counter fibers* that run along the top and right sides of the empty triangles that form in the negative space around the interior of $\mathbf{S}_{i+1}$. For each $i \in \mathbb{N}$, let

$$\Psi_i = \bigcup_{j=3}^{i-1} \bigcup_{k=2^i-2^j+3}^{2^i-j} \left\{(2^j-1,k),(2^j,k),(k,2^j-1),(k,2^j)\right\}. \tag{5.8}$$

Note that $\Psi_i = \emptyset$ for $i < 4$. Then, we define the sets $T_0, T_1, \ldots \subseteq \mathbb{Z}^2$ by

$$T_i = \begin{cases} \emptyset & \text{if i} < 4 \\ \Psi_i \cup T_{i-1} \cup \bigcup_{v \in V} (2^{i-1}v + (T_{i-1} - \Psi_{i-1})) & \text{otherwise.} \end{cases} \tag{5.9}$$

Intuitively, T_i is the set of *test fibers* between the counter fibers and cap fibers in $\mathbf{L}_i$. Now, for each $i \in \mathbb{N}$, let

$$\mathbf{L}_i = \mathbf{S}_i \cup C_i \cup N_i \cup T_i. \tag{5.10}$$

Then, the *laced Sierpinski triangle* is the set

$$\mathbf{L} = \bigcup_{i=0}^{\infty} \mathbf{L}_i. \tag{5.11}$$

We often refer to $\mathbf{L}_i$ as the i^{th} stage of $\mathbf{L}$. See Figure 5.7 for an illustration. From equation (5.10), it is clear that $\mathbf{L}$ is a superset of $\mathbf{S}$.

Observation 6. $\mathbf{S} \subseteq \mathbf{L}$.

We now show that the ζ-dimension of $\mathbf{L} \Delta \mathbf{S}$ (hence also of $\mathbf{L}$) is the same as the ζ-dimension of $\mathbf{S}$.

Theorem 31. $\mathrm{Dim}_\zeta(\mathbf{L} \Delta \mathbf{S}) = \mathrm{Dim}_\zeta(\mathbf{S})$.

Proof. Since $\mathbf{S} \subseteq \mathbf{L}$, it suffices to show that $\mathrm{Dim}_\zeta(\mathbf{L} \setminus \mathbf{S}) = \mathrm{Dim}_\zeta(\mathbf{S})$. By (5.10), for each $n \in \mathbb{N}$, $|(\mathbf{L} \setminus \mathbf{S})_{[0,2^n)}| = |C_n| + |N_n| + |T_n|$. By (5.5),

$$|C_n| = \begin{cases} 0 & \text{if n} < 2 \\ 3 & \text{if n} = 2 \\ 3|C_{n-1}| + 7 & \text{otherwise.} \end{cases}$$

Solving this recurrence for $n \geq 2$ gives $|C_n| = 6.5 \cdot 3^{n-2} - 3.5$. By (5.6), for $n \geq 2$, $|\Phi_n| = 2^{n+1}(n-3) + 8$. Then, by (5.7),

$$|N_n| = \begin{cases} 0 & \text{if n} < 3 \\ 8 & \text{if n} = 3 \\ 3|N_{n-1}| + |\Phi_n| - 2|\Phi_{n-1}| & \text{otherwise.} \end{cases}$$

Solving this recurrence for $n \geq 2$ gives $|N_n| = 4 \cdot 3^{n-1} - 2^{n+2} + 4$. By (5.8), for $n \geq 3$, $|\Psi_n| = 2^{n+2} - 2n^2 - 6n + 4$. Then, by (5.9),

$$|T_n| = \begin{cases} 0 & \text{if n} \leq 3 \\ 3|T_{n-1}| + |\Psi_n| - 2|\Psi_{n-1}| & \text{otherwise.} \end{cases}$$

Solving this recurrence for $n \geq 3$ gives $|T_n| = 3^{n-3}(n + 16.5) - 3n^2 - 9n + 34.5$. Then,

$$\begin{aligned} \text{Dim}_\zeta(\mathbf{L}\Delta\,\mathbf{S}) &= \limsup_{n\to\infty} \frac{\log_2(|C_n| + |N_n| + |T_n|)}{n} \\ &= \limsup_{n\to\infty} \frac{\log_2(3^{n-3}(n+72) - 2^{n+2} - 3n(n+3) + 35)}{n} \\ &= \log_2 3 \\ &= \text{Dim}_\zeta(\mathbf{S}) \text{ by Observation 5.} \end{aligned}$$

□

It remains to show that $\mathbf{L}$ strictly self-assembles. Our proof is constructive in that we exhibit a TAS in which $\mathbf{L}$ strictly self-assembles. We begin by explaining the general techniques used to fiber $\mathbf{S}$ in place, i.e., strictly self-assembly a superset of $\mathbf{S}$ without disturbing the set $\mathbf{S}$, and then delve into the details of a TAS implementing those techniques.

Conceptually, the communication fibers added to $\mathbf{S}$ enable a superset of $\mathbf{S}_{i+1}$ to strictly self-assemble when given a superset of $\mathbf{S}_i$ as input. By (5.1), $\mathbf{S}_{i+1}$ can be constructed by placing a copy of $\mathbf{S}_i$ on top and to the right of itself. This is achieved by copying the left boundary of $\mathbf{S}_i$ to the right of $\mathbf{S}_i$, and the bottom boundary of $\mathbf{S}_i$ to the top of $\mathbf{S}_i$. These communication fibers are divided into three functional groups. To ensure that the newly added bars are of the proper length, *counter* fibers control their attachment. The counter fibers increment until they have reached the same height as the middle point of the largest diagonal in $\mathbf{S}_i$, and then decrement

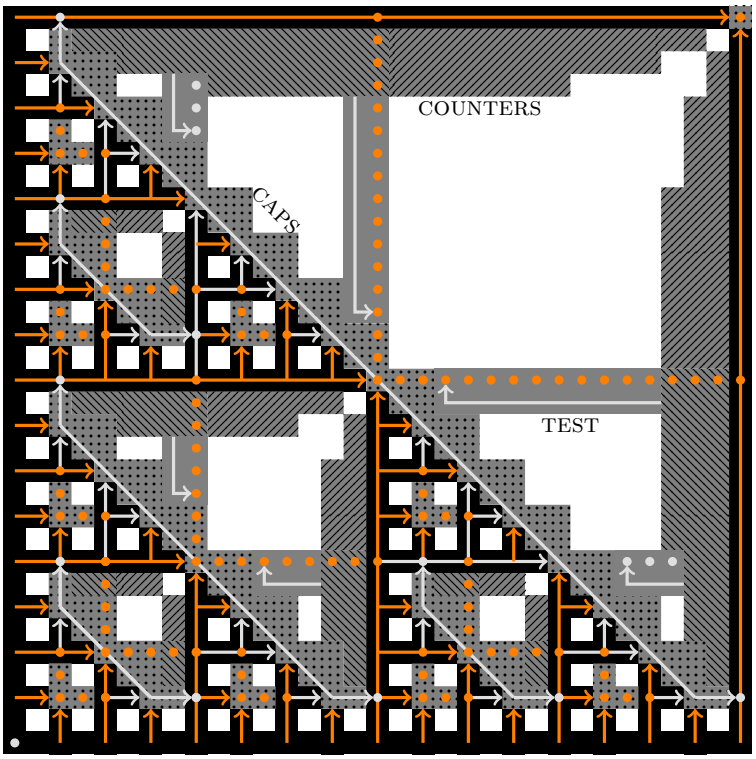


Figure 5.8: Fibering **S** in place.

to zero. To know where the middle point is, the counter fibers initiate the attachment of *test fibers* which grow back to $\mathbf{S}_i$, test whether the middle point is reached, and return the result to the counters. However, if $\mathbf{S}_i$ has not yet fully attached, the test fibers will read from the wrong location. Nor can the test fibers wait until $\mathbf{S}_i$ has completed attaching before returning to the counters, because the test fibers would have to know where to wait! The solution to this is the *diagonal cap* fibers that attach along the largest diagonal in **S** on the side opposite the seed. The purpose of the diagonal cap fibers is to force the necessary part of $\mathbf{S}_i$ to complete attaching before its middle is read by the test fibers. Then, a blocking technique can be used for the test fibers. The bottom row of the test fibers runs from the counters until blocked by the cap fibers. This attachment forms a path on which information can propagate from the diagonals back to the counters in a controlled manner. This is achieved by the *diagonal cap* fibers that attach along the largest diagonal in **S** on the side opposite the seed. They force the necessary part of $\mathbf{S}_i$ to complete attaching before the counters for $\mathbf{S}_{i+1}$ can begin to attach. Then, a blocking technique is used for the test fibers. The bottom row of the test fibers runs

from the counters until blocked by the cap fibers. This attachment forms a path on which information can propagate from the diagonals back to the counters in a controlled manner. See Figure 5.8 for an illustration. We now describe how the self-assembly determines the center of $\mathbf{S}_i$. A location is at the center of $\mathbf{S}_i$ when it sits directly above the left boundary of the $\mathbf{S}_{i-1}$ structure on the right part of $\mathbf{S}_i$ and directly to the right of the bottom boundary of the $\mathbf{S}_{i-1}$ structure on the top part of $\mathbf{S}_i$. This is computed in our construction by assigning to each bar of $\mathbf{S}$ a boolean value that is true (represented in Figure 5.8 by orange) only if it meets the criteria above. Every new bar that attaches to an existing bar will carry a true value unless it is the unique bar that attaches at the halfway point. Then, when two true bars meet, it is always at a location in the middle of the largest diagonal of some stage of $\mathbf{S}$. When this is the case, it is noted by the diagonal cap fibers so it can be passed to the test fibers. Note that every bar that attaches on the boundary has a true value.

We now construct a TAS $\mathcal{T}_{\mathbf{L}}$ that implements the techniques described. Let $\mathcal{T}_{\mathbf{L}} = (T_{\mathbf{L}}, \sigma_{\mathbf{L}}, 2)$ be a TAS such that the set $T_{\mathbf{L}}$ has ninety-five tile types as illustrated in Figure 5.9, and $\sigma_{\mathbf{L}}$ is a tile of type $\boxed{\texttt{S}}$ from Figure 5.9.

There are five tile types to assemble the boundary of $\mathbf{S}$, two for the bottom boundary and two for the left boundary. The bottom boundary is assembled by a tile with a west glue color of `0` attaching to the east side of $\boxed{\texttt{S}}$ and a tile with a west glue color of `1` attaching to the east side of it. This process continues ad infinitum. The left boundary assembles in a similar fashion. See Figure 5.10 for an illustration.

The are thirty-two tile types to assemble the horizontal and vertical bars in the interior of $\mathbf{S}$, sixteen for the vertical bars and sixteen for the horizontal bars. Here, we focus on the assembly of a vertical bar. A horizontal bar assembles in a similar fashion. The glue colors on the north and south sides of the tiles in a vertical bar are made up of the characters `+`, `-`, `*`, `T`, and `F`. Tiles used for the bottom half of a vertical bar use the `+` and cause the counter that assembles next to the bar to increment. Tiles used for the top half of a vertical bar use the `-` and cause the counter to decrement. A tile with a `*` in its south glue color also has a `*` for its east glue color. The `*` will be used by the cap tiles to know when to stop attaching. The `T` or `F` in the north and south glue colors propagates through the entire bar. When a location

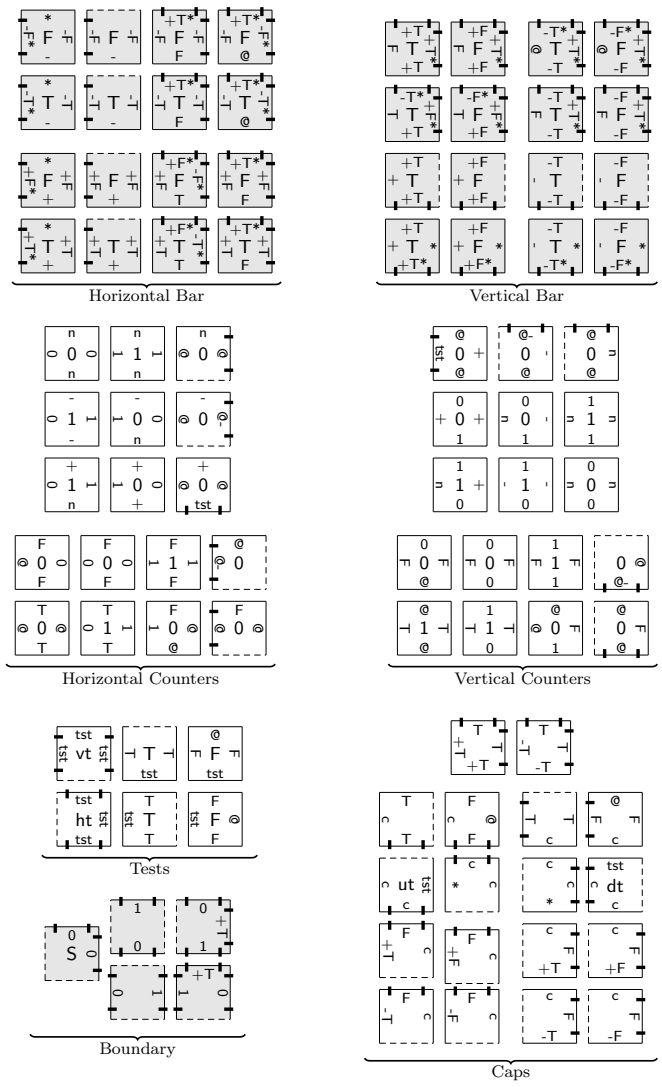


Figure 5.9: The tile set $T_{\mathbf{L}}$ of the TAS $\mathcal{T}_{\mathbf{L}}$.

$\vec{m}$ has a tile with a T in the glue color of both its south and west sides it means that $\vec{m}$ is the middle point of the largest diagonal to which it belongs and the cap tiles start their assembly at location $\vec{m}$. The west side of alternating tiles in the vertical bar have a glue color of either T, F, or @. These glues allow the vertical bar to receive feedback from the counters. A T glue color tells the bar that it is at half of its intended height, at which point the bar switches from instructing the counter to increment to instructing the counter to decrement. An @ glue color instructs the bar to complete its assembly. An F glue allows the bar continue assembling. The east side of these tiles also starts the growth of new horizontal bars. If the tile abuts a glue of

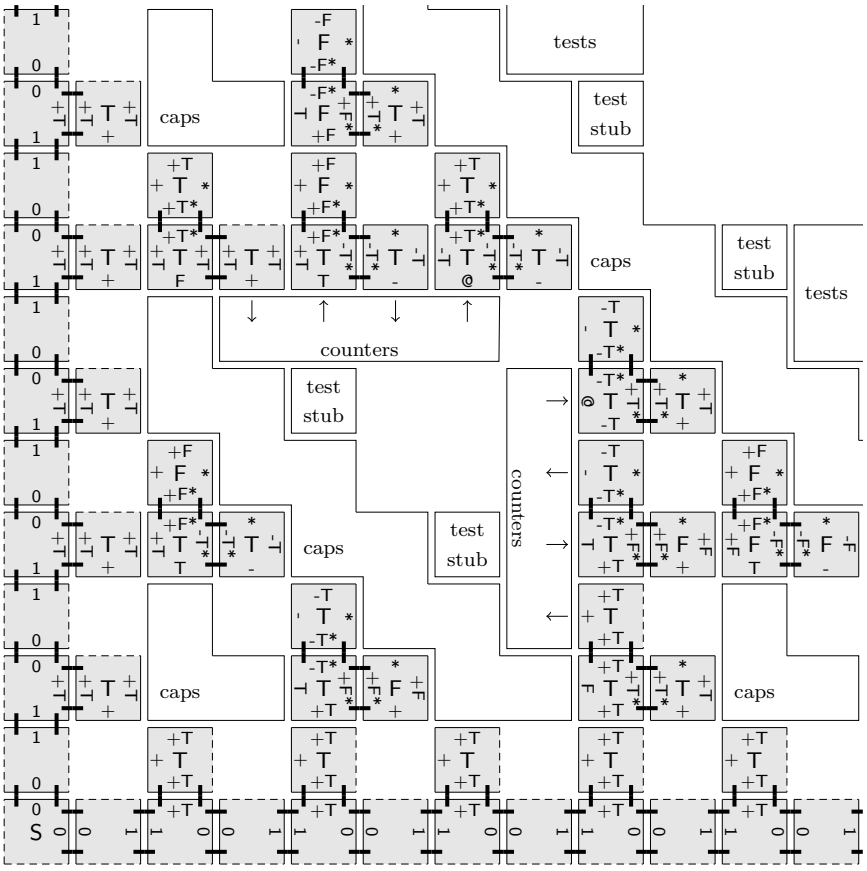


Figure 5.10: Example assembly of the horizontal and vertical bars of **S**.

color T (or F) on the counter, then it negates this value for the horizontal bar originating on its east side. See Figures 5.10 and 5.12 for an illustration.

The cap tiles are fibers that sit on top of the diagonals of **S**. There are eighteen tile types to assemble the cap tiles. Each diagonal of **S** has a vertical (horizontal) bar directly to the right (above) it. The height (length) of these bars can be computed directly from knowledge of the location of the middle of this diagonal. The test tiles will handle the two way communication between the caps and the counters so that this information can be used in the assembly of the bars. The cap tiles also delay the assembly of these bars until the assembly of the relevant part of the diagonal has been completed. This allows for the blocking behavior needed for proper assembly of these two-way communication fibers. The middle point of the diagonal will always be at the unique location that has a tile with a T in the glue color on both its south and west sides. When tiles meeting this criteria are present, a cap tile will attach to the assembly at

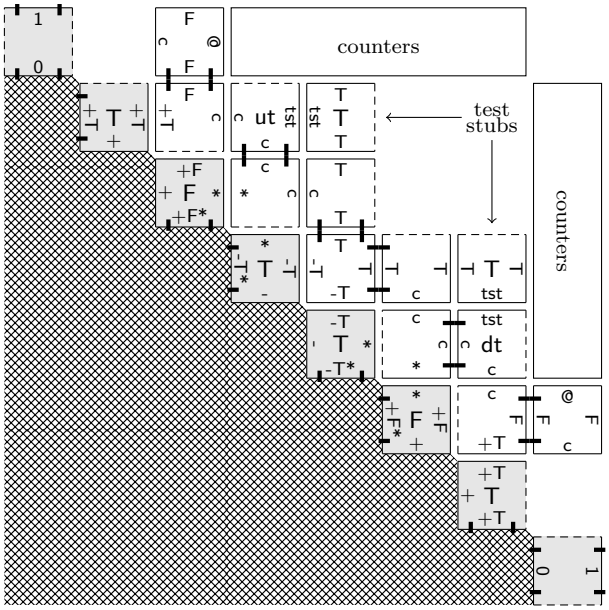


Figure 5.11: Example assembly of the cap fibers in **L**.

that location. This will trigger the assembly of cap tiles both up and down the diagonal. The growth of the cap tiles are controlled by the * glue colors on the tops (right sides) of the vertical (horizontal) bars making up the diagonal. They first attach to a tile having a * in its glue color on the abutting side, and then to a tile having a T or F in its glue color on the abutting side, then the process repeats. When a * glue color is not present, the cap tiles cease their assembly. See Figure 5.11 for an illustration.

There are thirty-four tile types to assemble the binary counter fibers that assemble adjacent to the bars of **S**, seventeen for the vertical counters and seventeen for the horizontal counters. The counters assemble in a zig-zag fashion. Alternating rows go from east to west (zig) and west to east (zag). The zig row either increments or decrements the value of the counter depending on the glue color on the abutting side of the tile on the bar of **S**. During the increment phase of the counter, if the counter is at a value that is one less than a power of two, then it initiates the two-way communication with the cap fibers by presenting a tst glue. The result of the test, either a T or F glue color, is propagated back to the bar. If the counter is not at a power of two, then the zag row returns the value F. During the decrement phase of the counter, if the counter is at a power of two, the zag row is triggered by a @- glue color, which instructs the

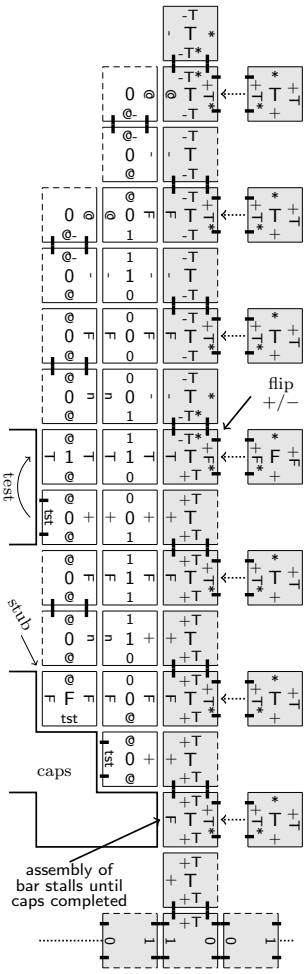


Figure 5.12: Example assembly of the counter fibers in **L**.

counter to shrink in width by one. See Figure 5.12 for an illustration.

There are six tile types to assemble the test fibers. Three for the vertical tests and three for the horizontal tests. These fibers allow for the two-way communication between the counter fibers and cap fibers. The request made by the counter fibers is sent along the bottom row of the test fibers and the response is returned along the top row of the test fibers. Because the counters do not assemble until the assembly of the corresponding caps have completed, we can be sure that the bottom row will not continue indefinitely – it will be blocked by the cap fibers. See Figure 5.13 for an illustration. We conclude with the following theorems which prove that **L** strictly self-assembles in the Tile Assembly Model.

We now show that $\mathcal{T}_{\mathbf{L}}$ satisifies the conditions for the generalization of local determinism

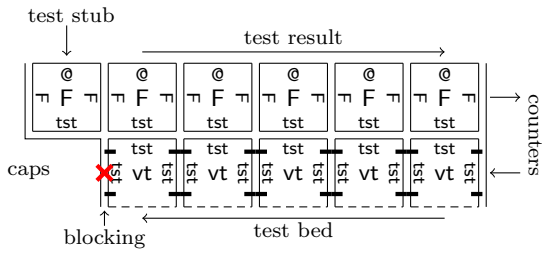


Figure 5.13: Example assembly of the test fibers in $\mathbf{L}$.

we introduced in Section 5.4.

Theorem 32. *$\mathcal{T}_{\mathbf{L}}$ is conditionally deterministic.*

Proof. Let $\vec{\alpha} = (\alpha_i \mid 0 \leq i < k)$ be any assembly sequence in $\mathcal{T}_{\mathbf{L}}$ such that $\operatorname{res}(\vec{\alpha})$ is terminal. It should be clear that there is such an assembly sequence and that $k = \infty$. First, we make the following observations that the reason that $\mathcal{T}_{\mathbf{L}}$ is not locally deterministic is because of the locations in α at which there is a tile of type $\boxed{\mathsf{ut}}$ or $\boxed{\mathsf{dt}}$ (of Figure 5.9).

1. For each $0 \leq i < k$, the unique tile type $t = \alpha(\vec{m})$, where $\vec{m} \in \operatorname{dom}\alpha_{i+1} \setminus \operatorname{dom}\alpha_i$, attaches to α_i with a strength of exactly 2.

2. For each location $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ such that $\alpha(\vec{m}) = \boxed{\mathsf{ut}}$, either

$$\vec{m} + \vec{u}_{\mathrm{N}} \in \partial^{\tau}_{\boxed{\mathsf{ht}}}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m})))))$$

   or

$$\partial^{\tau}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))) = \emptyset,$$

   and for all $t \in T_{\mathbf{L}} \setminus \{\boxed{\mathsf{ut}}, \boxed{\mathsf{ht}}\}$,

$$\partial^{\tau}_{t}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))) = \emptyset.$$

3. For each location $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ such that $\alpha(\vec{m}) = \boxed{\mathsf{dt}}$,

$$\vec{m} + \vec{u}_{\mathrm{E}} \in \partial^{\tau}_{\boxed{\mathsf{vt}}}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m})))))$$

   or

$$\partial^{\tau}(\alpha \upharpoonright (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))) = \emptyset,$$

and all $t \in T_{\mathbf{L}} \setminus \{\boxed{\mathsf{ut}}, \boxed{\mathsf{ht}}\}$,

$$\partial_t^{\mathcal{T}}\left(\alpha \restriction (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))\right) = \emptyset.$$

4. For each location $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ such that $\alpha(\vec{m}) \notin \{\boxed{\mathsf{ut}}, \boxed{\mathsf{dt}}\}$, and all $t \in T_{\mathbf{L}} \setminus \{\alpha(\vec{m})\}$,

$$\partial_t^{\mathcal{T}}\left(\alpha \restriction (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m}))))\right) = \emptyset.$$

5. $\partial^{\tau}\alpha = \emptyset$.

Thus, $\vec{\alpha}$ satisfies conditions (1) and (3) of both local determinism and conditional determinism. What prevents $\vec{\alpha}$ from satisfying condition (2) of local determinism is the second and third observation above. So, it suffices to show that

1. For each location $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ such that $\alpha(\vec{m}) = \boxed{\mathsf{ut}}$,

$$\partial^{\tau}(\alpha \restriction (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + (\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})))))) = \emptyset, \text{ and}$$

2. For each location $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ such that $\alpha(\vec{m}) = \boxed{\mathsf{dt}}$,

$$\partial^{\tau}(\alpha \restriction (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + (\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})))))) = \emptyset.$$

We will argue that (2) holds. The argument that (1) holds is similar. Let $\vec{m} \in \operatorname{dom}\alpha \setminus \operatorname{dom}\alpha_0$ such that $\alpha(\vec{m}) = \boxed{\mathsf{dt}}$. By construction, it must be the case that either $\alpha(\vec{m} + \vec{u}_{\mathrm{E}}) \uparrow$ or $\alpha(\vec{m} + \vec{u}_{\mathrm{E}}) \downarrow$. If $\alpha(\vec{m} + \vec{u}_{\mathrm{E}}) \uparrow$ then it follows that

$$\partial^{\tau}(\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + \mathrm{OUT}^{\vec{\alpha}}(\vec{m})))) = \emptyset, \text{ so}$$
$$\partial^{\tau}(\alpha \restriction (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + (\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})))))) = \emptyset.$$

If $\alpha(\vec{m} + \vec{u}_{\mathrm{E}}) \downarrow$ then the tile at $\alpha(\vec{m} + \vec{u}_{\mathrm{E}})$ must have attached along the bottom row of the test fibers initiated by the vertical bar directly to the right of $\vec{m}$ (i.e., $\alpha(\vec{m} + \vec{u}_{\mathrm{E}}) = \boxed{\mathsf{vt}}$). However, as illustrated in Figure 5.12, the second tile from the bottom uses the bottom right location of these caps as an input side. Hence, the vertical bar can not assemble above this point until all of the down caps along this diagonal have assembled. Thus, $\vec{m} \prec_{\mathcal{T}_{\mathbf{L}}} \vec{m} + \vec{u}_{\mathrm{E}}$. Hence, $\vec{m} + \vec{u}_{\mathrm{E}} \in \mathrm{DEP}^{\mathcal{T}}(\vec{m})$. Thus,

$$\partial^{\tau}(\alpha \restriction (\operatorname{dom}\alpha \setminus (\{\vec{m}\} \cup (\vec{m} + (\mathrm{OUT}^{\vec{\alpha}}(\vec{m}) \cup \mathrm{DEP}^{\mathcal{T}}(\vec{m})))))) = \emptyset. \qquad \square$$

We now show that $\mathbf{L}$ strictly self-assembles in $\mathcal{T}_{\mathbf{L}}$.

Theorem 33. $\mathbf{L}$ *strictly self-assembles in* $\mathcal{T}_{\mathbf{L}}$.

Proof. We say some set of locations $X \subseteq \mathbf{L}$ "properly assembles" if the intended tile type was placed at each location in the set, and no tile is placed at a location in $\mathbb{Z}^2 \setminus \mathbf{L}$. By Theorem 32 and Theorem 30, $|\mathcal{A}_{\square}[\mathcal{T}_{\mathbf{L}}]| = 1$. Pick the unique $\alpha \in \mathcal{A}_{\square}[\mathcal{T}_{\mathbf{L}}]$. It suffices to show that $\operatorname{dom}\alpha = \mathbf{L}$. We make the following claims about $\mathcal{T}_{\mathbf{L}}$.

(1) If $\mathbf{L}_i$ assembles properly, then $\mathbf{S}_{i+1}$ assembles properly. To see this note that $\mathbf{S}_i \subseteq \mathbf{L}_i$. Then, the same mechanics used to assemble $\mathbf{S}_i$ are used to assemble $(2^i, 0) + \mathbf{S}_i$ and $(0, 2^i) + \mathbf{S}_i$. Then, by (5.1), $\mathbf{S}_{i+1}$ assembles properly.

(2) If $\mathbf{S}_i$ assembles properly, then C_i assemble properly. To see this Let $\vec{m} = (2^{i-1}, 2^{i-1})$. Note that the tallest (widest) vertical (horizontal) bar of $\mathbf{S}_i$ originates from the boundary and hence propagates a $\mathtt{T}$ glue color throughout its assembly. Then, $\vec{m} + \vec{u}_{\mathrm{S}}$ and $\vec{m} + \vec{u}_{\mathrm{W}}$ have a glue color of $\mathtt{T}$ on its $\vec{u}_{\mathrm{N}}$ and $\vec{u}_{\mathrm{E}}$ sides respectively. Thus, C_i are allowed to begin their assembly at location $\vec{m}$ and since all of the smaller horizontal and vertical bars of $\mathbf{S}_i$ assemble properly, the caps will assemble up and down the longest diagonal in $\mathbf{S}_i$.

(3) If C_i assembles properly, then the largest horizontal and vertical bar of $\mathbf{S}_{i+1}$ along with N_i and T_i assemble properly. To see this note that the longest vertical and widest horizontal bar of $\mathbf{S}_{i+1}$ cannot grow very far until C_i has completed assembling (see Figure 5.12 for an illustration). At this point the proper assembly of these bars depends upon the proper assembly of N_i. But for N_i to assemble properly only depends on T_i to assemble properly, which in turn depends on C_i to assemble properly.

Our proof by induction easily follows from these claims. It is easy to see that $\mathbf{L}_0$, $\mathbf{L}_1$, $\mathbf{L}_2$, $\mathbf{L}_3$ properly assemble in $\mathcal{T}_{\mathbf{L}}$. It suffices to show that if $\mathbf{L}_i$ properly assembles in $\mathcal{T}_{\mathbf{L}}$, then $\mathbf{L}_{i+1}$ properly assembles in $\mathcal{T}_{\mathbf{L}}$. Suppose $\mathbf{L}_i$ has properly assembled in $\mathcal{T}_{\mathbf{L}}$. Then, by claim (1), $\mathbf{S}_{i+1}$ assembles properly. Then, by claim (2), C_{i+1} assemble properly. Then, by claim (3), N_{i+1} and T_{i+1} assemble properly. Hence $\mathbf{L}_{i+1}$ assembles properly. □

It is also interesting to note that $\mathbf{S}$ also weakly self-assembles in $\mathcal{T}_{\mathbf{L}}$.

Observation 7. **S** *weakly self-assembles in* $\mathcal{T}_{\mathbf{L}}$.

Instructions for simulating $\mathcal{T}_{\mathbf{L}}$ with the ISU TAS [61] are available at [54].

We conclude this section by presenting our second main theorem which shows that the bound given in our first main theorem, Theorem 26 is tight.

Theorem 34. *There exists a set* $X \subseteq \mathbb{Z}^2$ *with the following properties.*

(1) $\mathbf{S} \subseteq X$.

(2) $\text{Dim}_{\zeta}(X \Delta \mathbf{S}) = \text{Dim}_{\zeta}(\mathbf{S})$.

(3) X strictly self-assembles in the Tile Assembly Model.

Proof. Let $X = \mathbf{L}$. By Observation 6, condition (1) is satisfied. By Theorem 31 and Observation 3, condition (2) is satisfied. By Theorem 33 condition (3) is satisfied. □

5.6 Open Questions

Our results show that in the case of the Sierpinski triangle, no set "close" to the Sierpinski triangle strictly self-assembles. Given that no self-similar fractal is known to strictly self-assemble, a natural question is whether there exists a self-similar fractal that can be approximated closely. Is there a set X that strictly self-assembles and a self-similar fractal F such that $\text{Dim}_{\zeta}(X \Delta F) < \text{Dim}_{\zeta}(F)$?

We demonstrated a distortion-free fibering technique that enables a superset of the Sierpinski triangle to strictly self-assemble without increasing the fractal dimension of the intended structure. However, this technique depends on properties unique to the Sierpinski triangle and does not generalize to a large class of fractals. Is there a distortion-free fibering technique that generalizes to a large class of fractals without increasing the fractal dimension of the intended structure?

We gave an extension of local determinism sufficient for showing a blocking tile assembly system is directed. However, the relative order of when certain tiles attach *in every assembly sequence* must be established. In contrast, local determinism requires only the analysis of a

single assembly sequence. Is there a set of conditions that only requires the analysis of a single assembly sequence that is sufficient for showing a blocking tile assembly system is directed?

CHAPTER 6. Self-Assembling Rulers for Approximating Generalized Sierpinski Carpets

Modified from a paper to be published in *Algorithmica*

Steven M. Kautz and Brad Shutters

Abstract

Discrete self-similar fractals have been used as test cases for self-assembly, both in the laboratory and in mathematical models, ever since Winfree exhibited a tile assembly system in which the Sierpinski triangle self-assembles. For *strict* self-assembly, where tiles are not allowed to be placed outside the target structure, it is an open question whether any self-similar fractal can self-assemble. This has motivated the development of techniques to approximate fractals with strict self-assembly. Ideally, such an approximation would produce a structure with the same fractal dimension as the intended fractal, but with specially labeled tiles at positions corresponding to points in the fractal. We show that the Sierpinski carpet, along with an infinite class of related fractals, can approximately self-assemble in this manner. Our construction takes a set of parameters specifying a target fractal and creates a tile assembly system in which the fractal approximately self-assembles. This construction introduces *rulers* and *readers* to control the self-assembly of a fractal structure without distorting it. To verify the fractal dimension of the resulting assemblies, we prove a result on the dimension of sets *embedded* into discrete fractals. We also give a conjecture on the limitations of approximating self-similar fractals.

6.1 Introduction

Fractal structures are ubiquitous in nature but difficult to engineer using top-down techniques. The bottom-up approach of tile self-assembly, relying on Brownian motion and cooperative binding to allow simple objects to assemble into complexes, could prove to be a useful technology for engineering fractal structures. Carbone and Seeman [16] have stated that "generating fractal structures by self-assembly is a major challenge for nanotechnology." In this paper, our motivation is to use fractals as challenging test cases for self-assembly with the hope of discovering general self-assembly techniques that may prove useful in other constructions.

We will work in the abstract Tile Assembly Model (aTAM), a constructive version of Wang tiling [81, 82] introduced by Rothemund and Winfree as a mathematical model of self-assembly [65, 84]. There are two main notions of the self-assembly of a fractal in the aTAM. In *weak self-assembly*, one typically causes a two-dimensional surface to self-assemble with the desired fractal appearing as a labeled subset of the surface. Winfree [84] exhibited a tile assembly system in which the Sierpinski triangle weakly self-assembles. Kautz and Lathrop [46] showed that an infinite class of related fractals, including the Sierpinski carpet, can weakly self-assemble. However, in weak self-assembly the result is not a fractal structure, but rather a surface upon which the fractal pattern is "painted." In contrast, *strict self-assembly* requires only the fractal, and nothing else, to self-assemble. For structures with low fractal dimension, strict self-assembly requires significantly fewer physical tiles. However, due to the aperiodic arrangement of fractals, their strict self-assembly is fundamentally a more challenging problem than their weak self-assembly. Indeed, Lathrop, Lutz, and Summers [51] proved that the Sierpinski triangle cannot strictly self-assemble in the aTAM; and Patitz and Summers [62] extended this result to an infinite class of related fractals. It is an open question whether *any* fractal *can* strictly self-assemble in a model of self-assembly such as the aTAM.[1] This has motivated the development of techniques to approximate fractal structures with strict self-assembly.

Lathrop, Lutz, and Summers [51] and Patitz and Summers [62] developed a technique to

[1] Carbone and Seeman proposed an approach to the self-assembly of the Sierpinski carpet using specially designed molecules and a process requiring physical intervention at each successive stage of the fractal [15]. However, there is no direct translation of their approach into a model of self-assembly such as the aTAM.

approximate fractal structures by introducing communication fibers that shift the successive stages of the fractal. Although this technique results in a structure with the same fractal dimension as the intended fractal, the fractal pattern cannot be observed in specially labeled tiles. In fact, the resulting structure does not even contain the intended fractal structure. Ideally, an approximation of a fractal F would be an *in-place* approximation, i.e., a set $X \supset F$ with the same fractal dimension as F that strictly self-assembles in such a way that those tiles corresponding to F are specially labeled. Lutz and Shutters [53] exhibited a construction in which the Sierpinski triangle approximately self-assembles in-place. Prior to the results presented here, this was the only known non-trivial self-similar fractal to have an in-place approximation.

Our main result is that every generalized Sierpinski carpet has an in-place approximation in the aTAM. We exhibit a construction that takes a set of parameters specifying a target fractal and creates a tile assembly system in which the fractal approximately self-assembles.

Our construction introduces *rulers* and *readers* to control the self-assembly of a fractal structure without distorting it. Many tile assembly systems make use of optimal counters [17] for controlling a growing assembly [1, 51, 62, 67, 76]. For example, a set of counter tiles may grow eastward along the north side of another structure and count as it grows (as in Fig. 6.6a) to determine the width of the southern structure. If the final value N of the counter needs to be advertised not on the east edge of the counter, but on the north, this can be achieved by adding well-known rotator tiles that rotate the value 90 degrees counterclockwise, but this extends the length of the counter by $\log N$ tiles. The rotation can also be achieved by specially marking the glue of the $\log N$-to-last tile of the southern structure to begin rotation "in place" with the counting, but this requires that the southern structure come "pre-marked," which is not always possible. We introduce rulers and readers to solve this problem. They should be useful in other constructions. We provide a simulation [47] of our construction in the ISU TAS [61].

Constructing an in-place approximation X of a fractal F requires analysis of the set of added points $X \setminus F$ to show that the dimension of X is the same as that of F. In order to characterize the set $X \setminus F$ we develop a notion of *embedded fractals* and prove a very general,

and somewhat surprising, new result about their fractal dimensions. Given a subset G of the first quadrant and a fractal F, we define the *embedding of G in F* as the set X that is obtained, loosely speaking, by filling each empty square in F with a translated copy of G that is truncated to fill the available space. (Fig. 6.4 illustrates the embedding of the Sierpinski triangle in the Sierpinski carpet.) The goal is to use knowledge of the dimension of G to analyze the dimension of the approximation X. Our theorem establishes that the dimension of the set $X \setminus F$, and therefore of the set X, is always the maximum of the dimensions of F and G. As a simple example, if G is a single point, then the embedding X consists of F with just one point added to each empty square. Since G has dimension zero, we can conclude that the dimension of the set of added points $X \setminus F$ is equal to the dimension of F itself. Similarly, the structure in Fig. 6.4 has dimension $\log_3 8$, the same as that of the Sierpinski carpet. To the best of our knowledge, this is the first time that embedded fractals have been considered. We conclude by showing that this theorem provides strong evidence of a conjecture on the limitations of approximating fractal structures in the aTAM.

6.2 Preliminaries

6.2.1 Notation and Terminology

In this chapter we will build tile assembly systems by specifying a sequence of instructions used to create the tile types in the tile set of the system. To do so, we will use a special notation for a tile type as in Fig. 6.1. Strengths are represented by notches and colors by strings on the corresponding side and rendered in red when the value is specified externally (in a construction). Tiles may optionally contain a character or colored dot in the middle for labeling purposes.

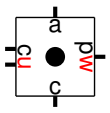
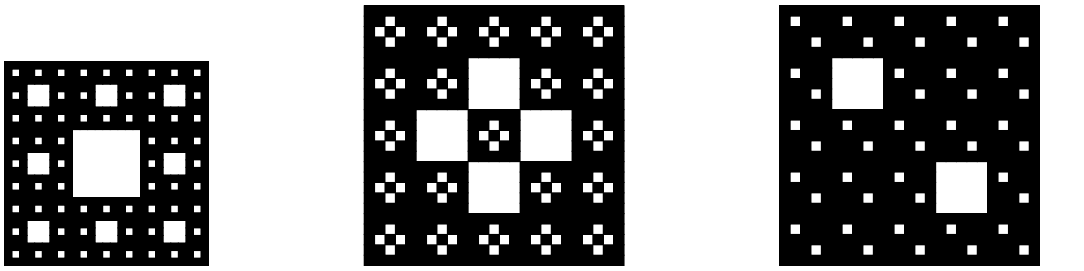


Figure 6.1: Example illustration of a tile used in a construction.



(a) $F|_{3^3}$, $a = b = c = 1$, $p = 3$ (b) $F|_{5^2}$, $a = b = c = 1$, $p = 5$ (c) $F|_{5^2}$, $a = c = 1$, $b = 3$, $p = 5$

Figure 6.2: Some generalized Sierpinski carpets.

6.2.2 Generalized Sierpinski Carpets

Certain fractals can be derived from the following numerical relationship, shown in [46] and independently in [63].

Theorem 35. *Let a, b, $c \geq 0$ and let p be a prime. Let $M : \mathbb{N}^2 \to \{0, 1, \ldots p-1\}$ be defined by $M[0,0] \equiv 1$, $M[0,j] \equiv a^j$ for $j > 0$, $M[i,0] \equiv b^i$ for $i > 0$, and*

$$M[i,j] \equiv aM[i,j-1] + bM[i-1,j] + cM[i-1,j-1] \quad \text{for } i,j > 0,$$

where the equivalences are modulo p. Define $S \subseteq \mathbb{N}^2$ by $(x,y) \in S \iff M[x,y] \not\equiv 0 \bmod p$. Then S is a discrete self-similar fractal with $p \times p$ kernel $S|_p$.

Definition 3. *Let $p > 2$ be a prime and let a, b, and c be positive integers not congruent to zero modulo p. If a, b, and c are not congruent to 0 modulo p, the fractal defined by Theorem 35 is a* generalized Sierpinski carpet*.*

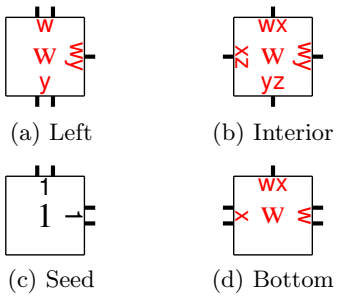
Fig. 6.2 shows some examples of generalized Sierpinski carpets.

Theorem 36. *Every generalized Sierpinski carpet weakly self-assembles.*

Theorem 36 was shown in [46]. The proof exhibits a general construction of the required tileset from the parameters a, b, c, and p. Fig. 6.3 illustrates this construction, where the "black" tiles are those with nonzero labels and the "white" tiles have label $w = 0$.

WeakCarpetTileset(a, b, c, p)
parameters: integers a, b, c and a prime p

include (c) as the seed.
foreach $j \in \{0, \ldots, p-1\}$
  include (a) with $y = j$ and $w = by \bmod p$.
  include (d) with $x = j$ and $w = ax \bmod p$.
foreach $(x, y, z) \in \{0, ..., p-1\}^3$
  include (b) with $w = (ax + by + cz) \bmod p$.



(a) Left (b) Interior (c) Seed (d) Bottom

Figure 6.3: Weak self-assembly of a generalized Sierpinski carpet.

6.3 Embedded Fractals

We now present a general result on sets embedded in discrete fractals, which will be used in Section 6.4 to establish the fractal dimension of the assemblies resulting from our construction. One of the tasks involved in applying Definition 2 is to show that the additional elements $X \setminus S$ do not increase the dimension of the approximation. In many cases $X \setminus S$ can be described by taking restrictions $G|_i$ of a known set G and replicating and translating them into empty $i \times i$ regions of S. This idea is captured in the following definition.

Definition 4. *Let F be a discrete fractal with $p \times p$ kernel K and let G be any nonempty subset of the first quadrant. Assume in the following that $0 \leq s, t < p$. Then* the embedding of G in F *is the set W defined by the recurrence*

$$W|_p = F|_p \cup \bigcup_{(s,t) \notin K_F} G|_1[s,t],$$
$$W|_{p^{k+1}} = \bigcup_{(s,t) \in K_F} W|_{p^k}[sp^k, tp^k] \cup \bigcup_{(s,t) \notin K_F} G|_{p^k}[sp^k, tp^k].$$

For example, the illustration in Fig. 6.4 shows the Sierpinski triangle embedded in the Sierpinski carpet. Our goal is to use knowledge of the dimension of G to analyze the dimension of the entire structure W. The result below characterizes the dimension of the added points, $W \setminus F$, and hence the dimension of W, when the restrictions $G|_i$ are replicated and translated into the empty regions of F. Notice that G itself does not need to be a fractal.

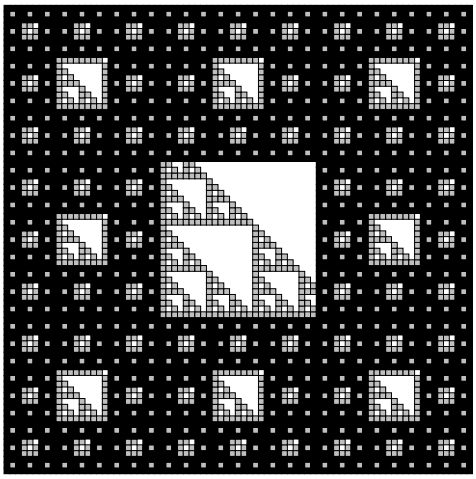


Figure 6.4: The Sierpinski triangle embedded in the Sierpinski carpet.

Theorem 37. *Let F be a self-similar fractal with kernel K, G a nonempty subset of $\mathbb{N}^2$, and W the embedding of G in F. Then,* $\mathrm{Dim}_\zeta(W \setminus F) = \max(\mathrm{Dim}_\zeta(F), \mathrm{Dim}_\zeta(G))$.

By Lemma 29 below, we can also conclude the following, where F, G, and W are as in Theorem 37.

Corollary 8. $\mathrm{Dim}_\zeta(W) = \max(\mathrm{Dim}_\zeta(F), \mathrm{Dim}_\zeta(G))$.

It is also possible to define a *reflected embedding* as in Definition 4 in which the elements of $G|_{p^k}$ are reflected across the negative main diagonal $y = p^k - 1 - x$, and the proof of Theorem 37 still applies. What is seen in Fig. 6.5 is actually the union of two embeddings into the Sierpinski carpet S, one for a set G representing the "rulers" and "readers" (described in Section 6.4 and shown in gray, blue, and red in the figure) and one for a reflected embedding of a set H representing the "decrementers" (shown in yellow). By Observation 3, the dimension of the resulting set is still $\max(\mathrm{Dim}_\zeta(S), \mathrm{Dim}_\zeta(G), \mathrm{Dim}_\zeta(H))$. Since $\mathrm{Dim}_\zeta(G) = 1$ by Theorem 40, and likewise $\mathrm{Dim}_\zeta(H) = 1$, we can conclude that the dimension of the set S with the embedded rulers, readers, and decrementers is still $\mathrm{Dim}_\zeta(S)$.

The remainder of this section is devoted to the proof of Theorem 37.

Lemma 28. $\mathrm{Dim}_\zeta(W \setminus F) \geq \mathrm{Dim}_\zeta(F)$.

Proof. Assume that F has $p \times p$ kernel K and let $d = |K|$. Since G is nonempty there is some M such that $G|_{p^M}$ is nonempty, and hence

$$\left|W|_{p^{k+1}}\right| = d\left|W|_{p^k}\right| + (p^2 - d)\left|G|_{p^k}\right| \geq d\left|W|_{p^k}\right| + 1$$

for all $k \geq M$. It follows that for all $k > M$

$$\left|W|_{p^k}\right| \geq d^k + \sum_{i=0}^{k-M} d^i \geq d^k + d^{k-M}$$

and since $\left|F|_{p^k}\right| = d^k$,

$$\left|W|_{p^k} \setminus F|_{p^k}\right| \quad \geq \quad d^{k-M}$$

and

$$\lim_k \frac{\log_p d^{k-M}}{k} = \log_p d = \mathrm{Dim}_\zeta(F).$$

□

Lemma 29. $\mathrm{Dim}_\zeta(W \setminus F) = \mathrm{Dim}_\zeta(W)$

Proof. Since $F \subseteq W$, $W = W \cup F = W \setminus F \cup F$,

$$\mathrm{Dim}_\zeta(W) = \mathrm{Dim}_\zeta(W \setminus F \cup F) = \max(\mathrm{Dim}_\zeta(W \setminus F), \mathrm{Dim}_\zeta(F)) = \mathrm{Dim}_\zeta(W \setminus F)$$

using Lemma 28 for the last equality. □

Lemma 30. *Let S be a subset of the plane with $\mathrm{Dim}_\zeta(S) = \log_p d$ for some $p > 1$. Then for any $\epsilon > 0$, $\left|S|_{p^k}\right| \geq d^{(1-\epsilon)k}$ for infinitely many k.*

Proof. Suppose that for some $\epsilon_0 > 0$, $\left|S|_{p^k}\right| < d^{(1-\epsilon_0)k}$ for all sufficiently large k; then

$$\mathrm{Dim}_\zeta(S) = \limsup_k \frac{\log_p \left|S|_{p^k}\right|}{k} \leq \limsup_k \frac{\log_p d^{(1-\epsilon_0)k}}{k} = (1 - \epsilon_0)\log_p d.$$

□

Lemma 31. *Let S be a subset of the plane with $\mathrm{Dim}_\zeta(S) = \log_p d$ for some $p > 1$. Then for any $\epsilon > 0$, $\left|S|_{p^k}\right| \leq d^{(1+\epsilon)k}$ for all but finitely many k.*

Proof. Suppose that for some $\epsilon_0 > 0$, $\left|S|_{p^k}\right| > d^{(1+\epsilon_0)k}$ for infinitely many k; then

$$\mathrm{Dim}_\zeta(S) = \limsup_k \frac{\log_p \left|S|_{p^k}\right|}{k} \geq \limsup_k \frac{\log_p d^{(1+\epsilon_0)k}}{k} = (1+\epsilon_0)\log_p d.$$

□

Lemma 32. *Let a, b, c be positive constants and f a monotone function such that $f(k) \geq b^k$ for infinitely many k. Define the recurrences*

$$v(k+1) = av(k) + cb^k,$$

$$w(k+1) = aw(k) + cf(k).$$

Then there exists a sequence $k_0 < k_1 < \cdots$ and a constant $M \geq 1$ such that $Mw(k_{t+1}) \geq v(k_t)$ for all $t \geq 0$.

Proof. Let $k_0 < k_1 < \cdots$ be a sequence of integers such that $f(k_t) \geq b^{k_t}$ for all $t \geq 0$. We can assume wlog that $k_{t+1} - k_t \geq k_t - k_{t-1}$ for all $t > 0$. Using the fact that f is monotone, so $f(k_t + i) \geq b^{k_t}$ for all i, it is not difficult to verify that

$$w(k_{t+1}) \geq a^{(k_{t+1}-k_t)}w(k_t) + cb^{k_t}\sum_{i=0}^{k_{t+1}-k_t-1} a^i = a^{(k_{t+1}-k_t)}w(k_t) + cb^{k_t}\left(\frac{a^{(k_{t+1}-k_t)}-1}{a-1}\right)$$

and that

$$v(k_{t+1}) \leq a^{(k_{t+1}-k_t)}v(k_t) + cb^{k_{t+1}}\left(\frac{a^{(k_{t+1}-k_t)}-1}{a-1}\right).$$

Choose $M \geq 1$ so that $v(k_0) \leq Mw(k_1)$. If we assume for an induction that $v(k_{t-1}) \leq Mw(k_t)$, then

$$\begin{aligned}
v(k_t) &\leq a^{(k_t-k_{t-1})}v(k_{t-1}) + cb^{k_t}\left(\frac{a^{(k_t-k_{t-1})}-1}{a-1}\right) \\
&\leq a^{(k_t-k_{t-1})}Mw(k_t) + cb^{k_t}\left(\frac{a^{(k_t-k_{t-1})}-1}{a-1}\right) \\
&\leq a^{(k_{t+1}-k_t)}Mw(k_t) + cb^{k_t}\left(\frac{a^{(k_{t+1}-k_t)}-1}{a-1}\right) \\
&\leq a^{(k_{t+1}-k_t)}Mw(k_t) + Mcb^{k_t}\left(\frac{a^{(k_{t+1}-k_t)}-1}{a-1}\right) \\
&\leq Mw(k_{t+1}).
\end{aligned}$$

□

We are now ready to prove Theorem 37.

Proof. (Of Theorem 37) Since $\mathrm{Dim}_\zeta(W \setminus F) = \mathrm{Dim}_\zeta(W)$ by Lemma 29, and $\mathrm{Dim}_\zeta(W \setminus F) \geq \mathrm{Dim}_\zeta(F)$ by Lemma 28, to complete the proof it will suffice to show that

$$\mathrm{Dim}_\zeta(W) \leq \max(\mathrm{Dim}_\zeta(F), \mathrm{Dim}_\zeta(G)) \text{ and} \tag{6.1}$$

$$\mathrm{Dim}_\zeta(W) \geq \mathrm{Dim}_\zeta(G). \tag{6.2}$$

Let $a = |K|$ and $c = (p^2 - a)$. $\left|W|_{p^{k+1}}\right|$ is defined by the recurrence

$$w(k+1) = aw(k) + c\left|G|_{p^k}\right|$$

where $\left|W|_p\right| = p^2$ if $(0,0) \in G$ and $\left|W|_p\right| = a$ if $(0,0) \notin G$, and

$$\mathrm{Dim}_\zeta(W) = \limsup_k \frac{\log_p w(k)}{k}. \tag{6.3}$$

We first show the upper bound (6.1). Consider the relation

$$v(k+1) = av(k) + cb^k, \tag{6.4}$$

which has a solution of the form $c_1 a^{k-1} + c_2 a^k + c_3 b^k$ for some constants c_1, c_2, c_3. It follows that

$$\lim_k \frac{\log_p v(k)}{k} = \begin{cases} \log_p a & \text{if } a \geq b \\ \log_p b & \text{if } b > a \end{cases}$$

Choose d such that $\mathrm{Dim}_\zeta(G) = \log_p d$ and recall that $\mathrm{Dim}_\zeta(F) = \log_p a$. Given any $\epsilon > 0$, let $b = d^{(1+\epsilon)}$. Note that if $d < a$ we can assume wlog that ϵ is small enough that $d^{(1+\epsilon)} < a$ also. By Lemma 31, there exists a k_0 such that $\left|G|_{p^k}\right| \leq b^k$ for all $k \geq k_0$. Choose a constant $M \geq 1$ such that $w(k_0) \leq Mv(k_0)$, and hence $w(k) \leq Mv(k)$ for all $k \geq k_0$. It follows that

$$\limsup_k \frac{\log_p w(k)}{k} \leq \lim_k \frac{\log_p Mv(k)}{k}.$$

In the case $\mathrm{Dim}_\zeta(F) > \mathrm{Dim}_\zeta(G)$, we have $a > b$, so the limit is $\log_p a = \mathrm{Dim}_\zeta(F)$. If $\mathrm{Dim}_\zeta(G) \geq \mathrm{Dim}_\zeta(F)$, the limit is $(1+\epsilon)\log_p d$, and since ϵ was arbitrary, the limit is bounded by $\log_p d = \mathrm{Dim}_\zeta(G)$. Thus

$$\mathrm{Dim}_\zeta(W) \leq \max(\log_p a, \log_p d) = \max(\mathrm{Dim}_\zeta(F), \mathrm{Dim}_\zeta(G)).$$

We now turn to the lower bound (6.2). Without loss of generality we can assume that $\mathrm{Dim}_\zeta(F) < \mathrm{Dim}_\zeta(G)$ and verify that in this case, $\mathrm{Dim}_\zeta(W) \geq \mathrm{Dim}_\zeta(G)$. Let $\epsilon > 0$. As before choose d so that $\mathrm{Dim}_\zeta(G) = \log_p d$, let $b = d^{(1-\epsilon)}$ and define v as in 6.4. By Lemma 30, $\left|G|_{p^k}\right| \geq b^k$ for infinitely many k. Since $\left|G|_{p^k}\right|$ is also a monotone function of k, by Lemma 32 there is a sequence $k_0 < k_1 < \cdots$ and a positive constant $m = \frac{1}{M}$ such that $w(k_{t+1}) \geq mv(k_t)$ for all $t \geq 0$. Since

$$\lim_t \frac{\log_p mv(k_t)}{k_t} = \log_p b = (1-\epsilon)\log_p d$$

we have

$$\limsup_t \frac{\log_p v(k_t)}{k_t} \geq (1-\epsilon)\log_p d.$$

Since ϵ was arbitrary it follows that $\mathrm{Dim}_\zeta(W) \geq \log_p d = \mathrm{Dim}_\zeta(G)$. □

6.4 Approximating Generalized Sierpinski Carpets

In this section we present our main result:

Theorem 38. *Every generalized Sierpinski carpet F has an in-place approximation in the aTAM.*

Our proof includes a construction that, given parameters a, b, c, and p specifying a generalized Sierpinski carpet, produces a TAS in which the approximating set X strictly self-assembles. The majority of this section is devoted to explaining this construction. We also note that in the tile types given in the constructions that follow, tile types that will be placed at positions corresponding to points in the intended fractal structure will be illustrated with a black dot in the middle.

A key observation, one that follows from the definition of a generalized Sierpinski carpet, is that in an assembly produced by the construction of Fig. 6.3, all the "white" tiles (tiles with label zero) occur in nonoverlapping, nonadjacent $p^k \times p^k$ blocks of "white" tiles. The main idea of the constructions we present here is to replace the blocks of "white" tiles in the TAS produced by the construction of Fig. 6.3 with a set of communication tiles that are sufficient for the "black" tiles to self-assemble, but which occupy a relatively insignificant fraction of

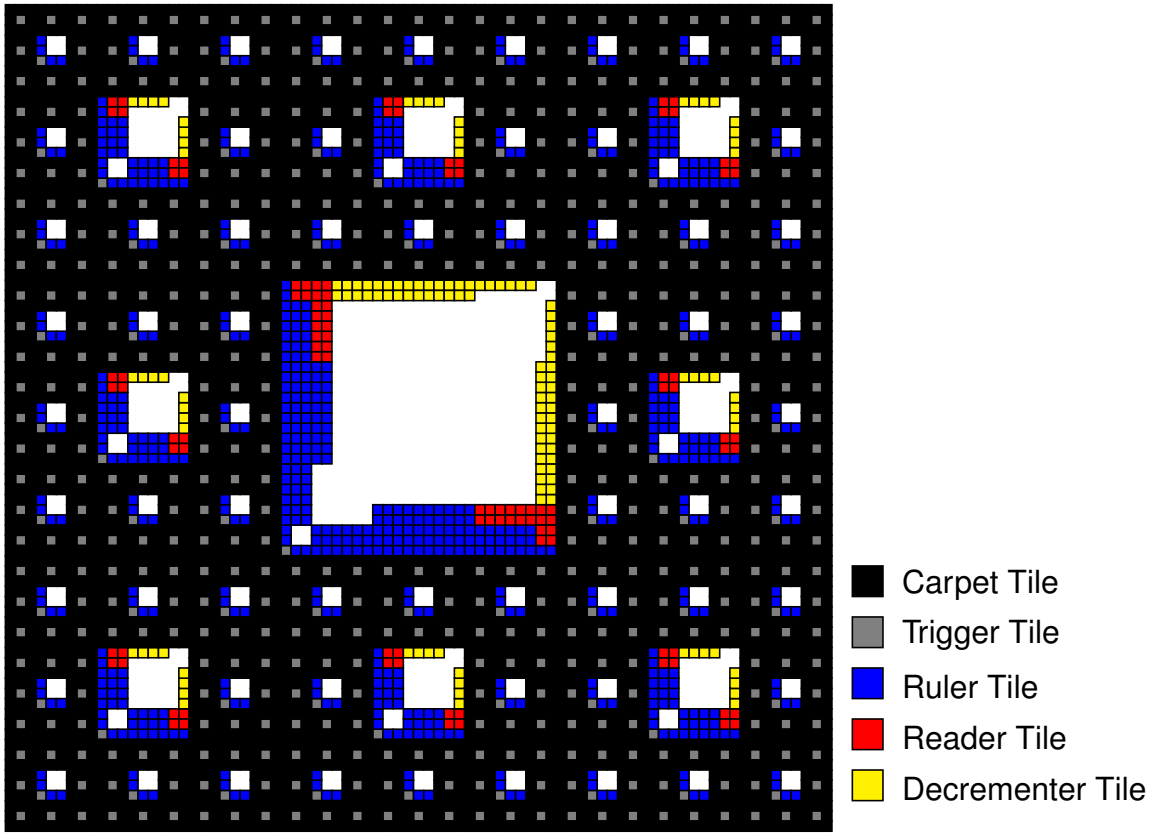


Figure 6.5: Rulers, readers, and decrementers embedded in the Sierpinski carpet.

the original $p^k \times p^k$ block of "white" tiles. These communication tiles are divided into three functional groups: *rulers*, *readers*, and *decrementers*. The ruler tiles assemble against the bottom and left sides of the block in such a way that the reader tiles can determine the size of the right and top sides of the square. The decrementer tiles then use this value to complete the square. See Fig. 6.5 for an illustration. Each of the ruler, reader, and decrementer tiles will need two sets of tile types, one for the tiles that assemble along the bottom and right sides of the square (which we call the "horizontal" tile types) and one for the tiles that assemble along the left and top sides of the square (which we call the "vertical" tile types). We will define the horizontal tile types along with a transformation $\mathbf{V}$ such that given a tileset T, $\mathbf{V}(T)$ consists of the tiles in T with the south and west glues swapped and the north and east glues swapped.

6.4.1 The Ruler Tiles

Most nontrivial tile assembly systems make use of counters [17] for controlling the growing assembly, for example see [1, 53, 51, 62, 67, 76]. When the value on the counter needs to be read perpendicular to the direction of the counter's growth, a turn operation is needed (see Fig. 6.6). However, this turn operation requires the counter to continue for $\log_b n$ more steps where b is the base of the counter and n is the current value. For the self-assembly of squares and other simple shapes this does not pose a problem because the counter can be signaled $\log_b n$ steps prior to the location at which its value is needed. But for the self-assembly of complex fractal structures, this is usually difficult and often impossible.

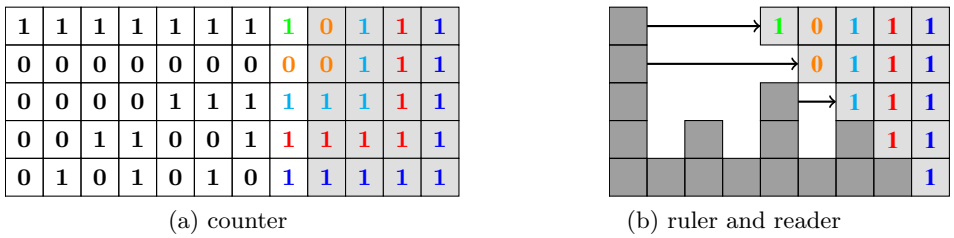
We thus introduce rulers, which are the key to our constructions, as a new mechanism for measuring distances in a growing assembly. By exploiting the interplay between the geometry of the assembling structure and the coding of the glues on the tiles, the rulers will allow for this desired corner turning operation without the need for additional space in the direction of the measurement.

The behavior of the ruler tiles is encapsulated by the ruler function [32]. For each integer $b \geq 2$, the *base b ruler function* $\rho_b : \mathbb{Z}^+ \to \mathbb{N}$ is defined by the recurrence

$$\rho_b(n) = \begin{cases} 0 & \text{if } n = bk + i \text{ for some } k \in \mathbb{N} \text{ and } 0 < i < b \\ \rho_b(k) + 1 & \text{if } n = bk \text{ for some } k \in \mathbb{Z}^+. \end{cases} \tag{6.5}$$

Intuitively, $\rho_b(n)$ is the exponent of the largest power of b that divides n.

It is an easy proof that the ruler structure of Fig. 6.6 cannot strictly self-assemble similar to the proof in [51] that the Sierpinski triangle cannot strictly self-assemble. Our goal then is



(a) counter (b) ruler and reader

Figure 6.6: Corner turn operation for a base 2 counter and ruler.

to develop a notion of a ruler that *does* strictly self-assemble. We do so by adding tiles to the structure to allow the ruler bars to communicate with their neighbors. We will also double the height of the bars for reasons that will become clear when we we discuss the reading mechanism later in this section. For each integer $b \geq 2$ and $n \in \mathbb{Z}^+$, let

$$B_{b,n} = \bigcup_{i=0}^{2\rho_b(n)} \{(n,i)\} \quad \text{and} \quad T_{b,n} = \bigcup_{i=1}^{t_b(n)} \bigcup_{j=n-b^i+1}^{n-1} \{(j,2i-1),(j,2i)\}$$

where $t_b(n) = \rho_b(n) - [\![n \in \{b^k : k \in \mathbb{N}\}]\!]$. Intuitively, each $B_{b,n}$ is a bar of height $2\rho_b(n)$ and $T_{b,n}$ is the set of communication fibers added to the structure to allow $B_{b,n}$ to properly assemble. Then, we define sets

$$R_{b,n} = \bigcup_{i=1}^{n} B_{b,i} \cup T_{b,i} \quad \text{and} \quad R_b = \bigcup_{n=1}^{\infty} R_{b,n}.$$

We refer to $R_{b,n}$ as the n^{th} stage of a *base b ruler* and R_b as the *infinite base b ruler*.

Theorem 39. *For each base $b \geq 2$, there exists a tile set T_b such that for all $n \in \mathbb{Z}^+$, there is a seed assembly σ for which $R_{b,n}$ strictly self-assembles in the tile assembly system $(T_b, \sigma, 2)$.*

Proof. The following discussion shows that there exists an assembly sequence for $\mathcal{T}_{b,n}$ in which $R_{b,n}$ strictly self-assembles. It is an easy excercise to show that this assembly sequence is locally deterministic, and thus directed. □

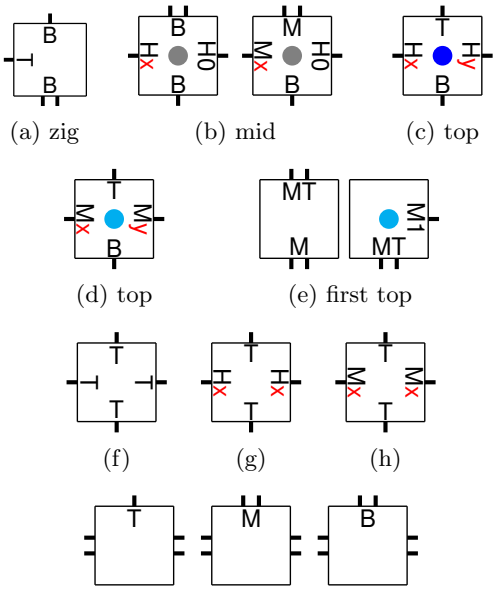
For a given base b and $n \in \mathbb{Z}^+$, let $\mathcal{T}_{b,n} = (T_b, \sigma, 2)$ where $T_b = \mathsf{RulerTileset}(b)$, $\operatorname{dom}\sigma = \{1,\ldots,n\} \times \{0\}$ and σ is constructed from the tile types in Fig. 6.7i such that for each $1 \leq i \leq n$, the tile at $(i,0)$ has a B as its north glue color when $i = b$, M as its north glue color when $i = kb$ for some $k > 1$, and T as its north glue color otherwise.

To explain the assembly sequence of $\mathcal{T}_{b,n}$, we decompose the bars of the ruler into series. Each series consists of a group of $b-1$ bars, all of the same height. The series begins with the first bar at that height following a bar of greater height. The series ends when a bar of greater height appears. We also give special attention to the first series of a given height to appear in the ruler. As a concrete example, see Fig. 6.7j which illustrates an assembly sequence for $\mathcal{T}_{3,54}$.

An assembly sequence in $\mathcal{T}_{b,n}$ mainly consists of "bar" tiles (i.e. Fig.s 6.7a – 6.7e) attaching to the seed configuration σ. The bar tiles that attach to the seed configuration at location

RulerTileset(b)
parameters: a base b

1 include (a), (e) & (f).
2 **foreach** $x \in \{0, \ldots, b-1\}$
3 $\quad y \leftarrow x+1$.
4 $\quad$ **if** $y = b$ **then** include (b).
5 $\quad$ **if** $y < b$ **then** include (c).
6 $\quad$ **if** $x > 0$ & $y < b$ **then** include (d).
7 $\quad$ include (g).
8 $\quad$ **if** $x > 0$ **then** include (h).



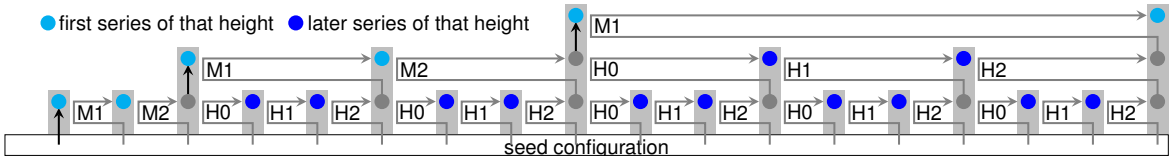


Figure 6.7: Tileset for a ruler.

$(b, 1)$ are of the type 6.7e. For each location $(kb, 1)$ where $k > 1$, the bar that assembles at that location consists of alternating tiles of type 6.7a and then one of the types in 6.7b – 6.7d, or, if B is the first bar of that height, both of the types in 6.7e. To determine which tile should attach, the 6.7a tile type initiates the assembly of communication tiles (i.e. 6.7f) which assemble to the nearest bar L to the left of the assembling bar B that is at least the current height h of B. Once these communication tiles find L, they allow L to return a signal representing its "bar type" using tile types of 6.7g – 6.7h. The type of tile to attach next to B is determined from the returned signal as follows.

- Hx $(0 \leq x < b-1)$: Then L is either a bar of height greater than $h+1$ (when $x = 0$) or L is the x^{th} bar in a series of height $h+1$ (when $x > 0$). In either case, B is the $x+1^{st}$ bar in a series of height $h+1$. B's assembly completes with a 6.7c tile type (with $y = x+1$) attaching.

- Hx ($x = b-1$): Then L is the last bar in a series of height $h+1$, so B's height is greater than $h+1$. B's assembly continues to height $h+2$ by the left tile type of 6.7b attaching and then a 6.7a tile type attaching and the communication process repeats.
- Mx ($x < b-1$): Then L is the x^{th} bar and B is the $x+1^{st}$ bar in the *first* series of height $h+1$. B's assembly completes with a 6.7d attaching (with $y = x+1$).
- Mx ($x = b-1$): Then L is the last bar in the *first* series of height $h+1$, so B is the first bar of height $h+3$. B's assembly continues to height $h+3$ with the two 6.7e tile types attaching.

Our motivation in developing rulers is to embed them in fractal structures to enable strict self-assembly. By Theorem 37 of Section 6.3, if we are to do so without increasing the fractal dimension of the resulting structure, it is important that rulers have a small ζ-dimension. We show that the ζ-dimension of a ruler is the same as that of a line.

Theorem 40. *For each $b \geq 2$, the ζ-dimension of R_b is 1.*

Proof. Since the ζ-dimension of any infinite connected set is at least 1, it suffices to show that the ζ-dimension of each $R_b \leq 1$. Let $V_2(n)$ be as in (4.1). It is clear that

$$R_b \cap V_2(n) \subseteq \{0, \ldots, n\} \times \{0, \ldots, 2\lfloor \log_b n \rfloor\}.$$

Then,

$$\mathrm{Dim}_\zeta(R_b) \overset{(4.1)}{=} \limsup_{n\to\infty} \frac{\log |R_b \cap V_2(0,n)|}{\log n} \leq \limsup_{n\to\infty} \frac{\log(3n \log n)}{\log n} = 1.$$

□

We now discuss how the rulers are embedded in the self-assembly of a generalized Sierpinski carpet. We replace the seed configuration of $\mathcal{T}_{b,n}$ with a set of ground tiles that bind to the "black" tiles along the south and west sides of each block of "white" tiles in the carpet. These tiles are defined in Fig. 6.8. Each ruler will begin its assembly on the north or east side of a ruler seed tile, defined in Fig. 6.11e and illustrated in Fig. 6.5. The ground tiles continue to attach along the bottom or left side of the square, but cannot extend past the square, a fact

which depends on the nature of the glues that can occur on the "black" tiles.

The ground tiles for a base b ruler assemble in cycles of b tile types. The first cycle has glue colors beginning with R on the tile types east and west sides. The remaining cycles have glue colors beginning with S on the tile types east and west sides. At the end of the first cycle, the tile type that attaches has a M for a glue color on the north side, causing the initial bar to assemble. The remaining cycles all end with a tile type having a B for a glue color on the north side, causing a bar to assemble that uses communication with previous bars to determine its intended height.

We will find it useful later to isolate the property that characterizes the seed for a self-assembling ruler.

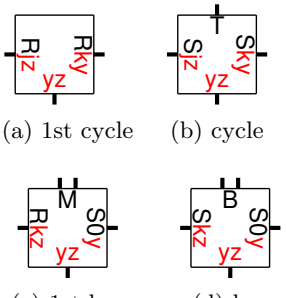
Definition 5. *Assume a fixed base b.*

1. *Let $s = \{t_i\}_{0 \le i < k}$ be a horizontal sequence of k tiles. The sequence s is a* valid ruler ground sequence *if for all $i < k$, the tile t_i has a* B *as its north glue color when $i = b$,* M *as its north glue color when $i = kb$ for some $k > 1$, and* T *as its north glue color otherwise.*

2. *Let $s = \{t_i\}_{0 \le i < k}$ be a vertical sequence of k tiles. The sequence s is a* valid vertical ruler ground sequence *if for all $i < k$, the tile t_i has a* B *as its east glue color when $i = b$,* M *as its east glue color when $i = kb$ for some $k > 1$, and* T *as its east glue color otherwise.*

GroundTileset(p, y, z)
parameters: a prime p and integers y, z

1 **foreach** $j \in \{0, 1, \ldots, p-2\}$
2 $\quad k \leftarrow j + 1$.
3 $\quad$ include (a) & (b).
4 $k \leftarrow p - 1$.
5 include (c) & (d).



(a) 1st cycle (b) cycle

(c) 1st bar (d) bar

Figure 6.8: Tileset for ground of ruler embedded in a carpet.

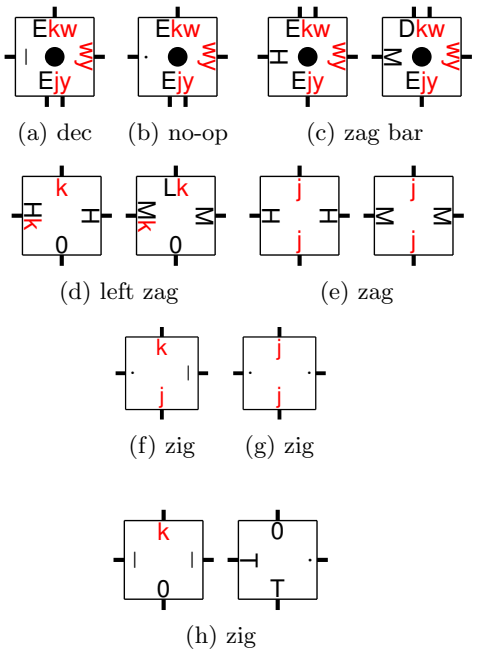
6.4.2 The Reader Tiles

Once a ruler embedded along the bottom edge of the square has completed its assembly, its value needs to be read so that the right side of the square can properly assemble. The purpose of the reader is to broadcast the ruler's value along the north sides of the reader tiles once they have assembled to the height of the highest bar. In principle, the growth of a reader is similar to the growth of a ruler bar, in that its height is controlled by communication with the ruler bars to its left, using the same communication tiles used within the ruler (i.e. Fig. 6.7f – 6.7h). The difference is that each time a signal is returned, another base p digit is added to the left of the growing reader. The readers used in the present construction are particularly simple because the length of the embedded ruler, not including the ruler seed, is always one less than a power of p. However, since the height of the right side of the square is the same as the length of the bottom, we want the value on the reader to be the length of the ruler, *less* the height of the reader itself. This is accomplished by combining a decrement operation with the growth of the rightmost tiles of the reader. An example is shown in Fig. 6.9i for a base 3 ruler of length 80; note that the value encoded in the north glues of the reader tiles is 74 (in base 3), which is 81 less 7, the height of the reader itself.

The growth of a reader is initiated by a reader trigger tile which has one of the three forms shown in Fig.s 6.11g, 6.11i, and 6.11k. The remaining tiles for the reader are shown in Fig. 6.9. The tiles in Fig. 6.9a – 6.9c will form the reader bar along its rightmost side; note that the glues on the east sides of these tiles are consistent with the glues of the "black" tiles from the construction of Fig. 6.3. The reader then assembles in a *zig-zag* fashion and its assembly is controlled by the ruler bar. Each time a tile attaches to the reader bar, the value on the reader is decremented by 1. If the value on the reader bar is either 0 or 1 and when necessary a decrement signal is sent along the zig tiles to decrement the higher order bits of the value. Each group of two tiles that attach to the reader bar represent a trip to the ruler and back. The zag row returns which type of ruler bar was found. If an H bar was found, the reader continues. If an M bar was found, the reader completes and initiates the assembly of the decrementer tiles.

ReaderTileset(b,w,y)
parameters: a prime b and integers w, y

1 **foreach** $j \in \{0, \ldots, b-1\}$
2 $k \leftarrow (j-1) \bmod b$.
3 **if** $j \leq 1$ **then** include (a).
4 **if** $j > 1$ **then** include (b).
5 include (c), (g), & (e).
7 **if** $j \geq 1$ **then** include (f).
8 $k \leftarrow b-1$.
8 include (h) & (d).

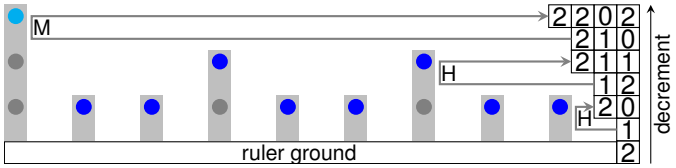


(a) dec (b) no-op (c) zag bar

(d) left zag (e) zag

(f) zig (g) zig

(h) zig



(i) Example assembly of a base 3 reader.

Figure 6.9: Tileset for reader tiles embedded into a carpet.

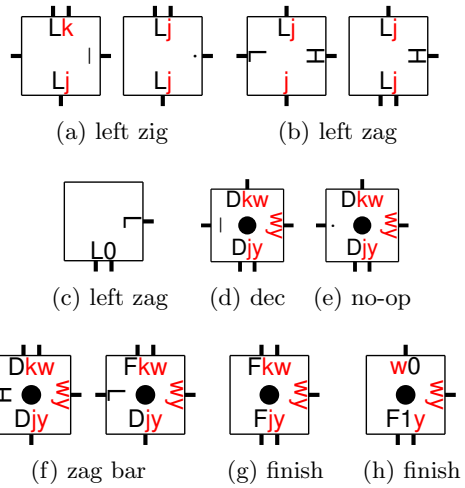
6.4.3 The Decrementer Tiles

Once the reader is assembled, we have the length p^k of the ruler, less the height of the reader itself, encoded in the glues on the north sides of the reader tiles. A tileset that assembles upwards from the reader to exactly the desired height p^k is constructed in Fig. 6.10. The decrementer works in the same zig-zag fashion as the reader. We also note that the non-bar tile types of the reader (i.e. Fig. 6.9d – 6.9e) are used. The decrementer shrinks in width whenever the most significant bit becomes 0, and the topmost p tiles of the main decrementer bar are hardcoded. Note that there are multiple forms for Fig. 6.10d–6.10h, the tiles along the right side of the decrementer, because their east glues have to interact with the glues of the "black" tiles from the original construction of Fig. 6.3.

We now can formalize the fact that given an appropriate ruler ground sequence of length

DecrementTileset(p, w, y)
parameters: a prime p and integers w, y

1 **foreach** $j \in \{0, \ldots, p-1\}$
2 $k \leftarrow (j-1) \bmod p$.
3 **if** $j \leq 1$ **then** include (d).
4 **if** $j > 1$ **then** include (e).
5 include (f).
6 **if** $j \geq 2$ **then** include (g).
7 **if** $j > 0$ **then** include (a) & (b) .
8 include (h) & (c).



(a) left zig (b) left zag (c) left zag (d) dec (e) no-op (f) zag bar (g) finish (h) finish

Figure 6.10: Tileset for decrementer tiles embedded into carpet.

p^k plus a reader trigger, a ruler, reader, and decrementer of height p^k will strictly self-assemble above the ruler ground sequence.

Lemma 33. *(a) Let α be an assembly and let $i, j \geq 0$. Suppose that $\alpha(i, j)$ is a ruler trigger as in Fig. 6.11(e), $\alpha(i+1, j), \ldots, \alpha(i+p^k-1, j)$ is a valid ruler ground sequence, and $t = \alpha(i+p^k, j)$ is a reader trigger as shown in Fig. 6.11(f)–(g). Assume also that α is undefined for all (i', j') with $i' > i$ and $i' \geq j' > j$. Then there exists an assembly sequence with result α', where $\alpha \to \alpha'$, for a ruler, reader and decrementer of height p^k, where $\alpha'(i+p^k, j), \alpha'(i+p^k, j+1), \ldots, \alpha'(i+p^k, j+p^k-1)$ contains the rightmost bar of the reader/decrementer.*

(b) (Same as (a), but for a vertical ruler.)

6.4.4 Putting it All Together

The construction of the complete tileset for approximating a generalized Sierpinski carpet is shown in Fig. 6.11. The set we intend to assemble using the tileset of Fig. 6.11 is illustrated in Fig. 6.5, where the locations of the tiles of Fig. 6.11 are shown in black. Each block of "white" tiles from the original assembly produced by the construction of Fig. 6.3 is removed,

along with the "black" tiles directly adjacent to its right and upper sides, and the block is replaced by a pair of rulers, readers, and decrementers forming the border of the block.

CarpetApproximationTileset(a, b, c, p)
parameters: a prime p and integers a, b, c

1 $n \leftarrow p-1$.
2 include (a) which is the seed tile.
3 **foreach** $j \in \{1, 2, \ldots, n\}$
4 include (b) with $y=j$ and $w=by \bmod p$.
5 include (c) with $x=j$ and $w=ax \bmod p$.
6 **foreach** $(x, y, z) \in \{0, 1, \ldots, n\}^3$
7 $w \leftarrow (ax+by+cz) \bmod p$.
8 **if** $w>0$, $x>0$ and $y>0$ **then**
9 include (d).
10 **if** $w=0$, $x>0$ and $y>0$ **then**
11 include (e).
12 **if** $w=0$, $x=0$ and $y>0$ **then**
13 include **GroundTileset**(p,y,z).
14 **if** $w=0$, $x>0$ and $y=0$ **then**
15 include **V**(**GroundTileset**(p,x,z)).
16 **if** $w>0$, $x=0$, $y>0$ and $z>0$ **then**
17 include (f) & (g).
18 **if** $w>0$, $x>0$, $y=0$ and $z>0$ **then**
19 include (h) & (i).
20 **if** $w>0$, $x=0$, $y=0$ and $z>0$ **then**
21 include (j) & (k).
22 **if** $w>0$, $x=0$, $y>0$, $z=0$ **then**
23 include **ReaderTileset**(p,w,y)
24 include **DecrementTileset**(p,w,y).
25 **if** $w>0$, $x>0$, $y=0$, $z=0$ **then**
26 include **V**(**ReaderTileset**(p,w,x))
27 include **V**(**DecrementTileset**(p,w,x)).
28 include **RulerTileset**(p)
29 include **V**(**RulerTileset**$((p))$.

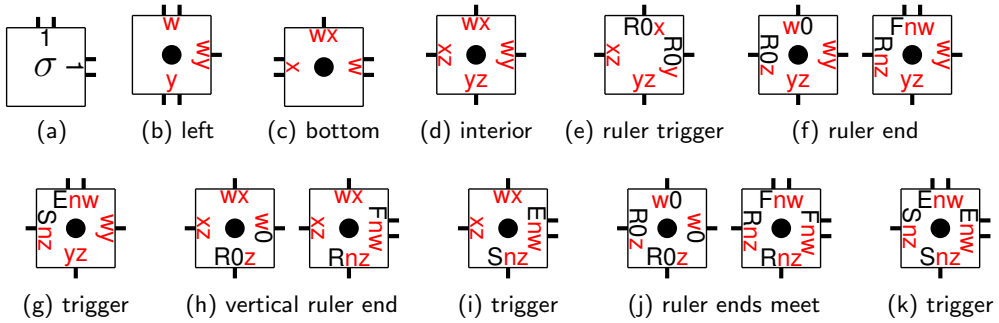


Figure 6.11: Constructing a tileset for approximating a generalized Sierpinski carpet.

The tiles in Fig. 6.11a – 6.11d are identical to those from the construction of Fig. 6.3, excluding those tiles with $w=0$, $x=0$, or $y=0$. These tiles are added in lines 2 – 9. The case $w=0$, $x>0$, and $y>0$ indicates the lower left hand corner of a "white" block, and the tile type of Fig. 6.11e added in line 11 is used to trigger both a horizontal and vertical ruler. If $w=0$, $x=0$, and $y>0$, the position of the tile is along the bottom of a white block, where the tile types added in lines 12 – 13 assemble into a ruler ground. The case $w=0$, $x>0$, and $y=0$ is similar for a vertical ruler. If $w>0$, $x=0$, $y>0$, and $z>0$ the tile's position is at the bottom left corner of a black block abutting a white block on its left. If this block is of size 1×1 or

$p \times p$ then the height of the right side of the white block is hardcoded into the tile set by the tile types of Fig. 6.11f. Otherwise, we need to trigger a horizontal reader to assemble the right side of the white block to the proper height, which is triggered by the tile type of Fig. 6.11g. The case $w > 0$, $x > 0$, $y = 0$, and $z > 0$ is similar for a vertical reader using the tile type of Fig. 6.11i. A special situation arises when $w > 0$, $x = 0$, $y = 0$, and $z > 0$, indicating the lower left corner of a black block with a white block to its left and a white block below. This causes two ruler ends to meet, so we need the special tiles of Fig. 6.11k to deal with triggering both vertical and horizontal readers for the two blocks at once. All of these cases are handled by the tiles added in lines 16 – 21. If $w>0$, $x=0$, $y>0$ and $z=0$, then the position is along the right side of a white block and is part of the main bar for the reader and decrementer. Likewise, if $w>0$, $x>0$, $y=0$ and $z=0$ the position is along the top side of a white block forming the reader and decrementer for a vertical reader. These tiles are added in lines 22 – 27. Finally, we add the tile types for the ruler bars in line 28 – 29.

6.4.5 Proof of Correctness

Let F be a generalized Sierpinski carpet, let X denote the set informally described above and pictured in Fig. 6.5, and let $\mathcal{T}$ denote the TAS $(T, \sigma, 2)$, where T is the tileset defined by Fig. 6.11 and σ is the seed tile of Fig. 6.11a. Theorem 38 is established via the following steps:

1. Precisely define the set X.
2. Define a T-assembly sequence whose terminal assembly is X.
3. Show that $\mathcal{T}$ is directed, i.e., the terminal assembly is unique.
4. Show that $F \subset X$ and $\text{Dim}_\zeta(F) = \text{Dim}_\zeta(X)$.

Steps 1 and 2 are established by Lemma 39(b). Verification of step 3 is routine. For step 4 we appeal to Corollary 8.

We first define some notation and terminology that will be useful in the proof. Let $\vec{u}_N$, $\vec{u}_S$, $\vec{u}_E$, $\vec{u}_W$ denote the four unit vectors in the plane that are parallel to the coordinate axes, viewed as four cardinal directions. Given a tile t, and one of these unit vectors $\vec{u}$, let $\text{col}_t(\vec{u})$ denote the glue color on the side of t in direction $\vec{u}$. Note that in our construction the glue

color is frequently an integer or pair of integers, as seen in Fig. 6.3, where a pair such as (w, y) is shown as "wy" in the illustrations.

Given a TAS $\mathcal{T} = (T, \sigma, \tau)$ and an assembly $\alpha : \mathbb{Z}^2 \to T$, let $\alpha[x, y]$ denote the subassembly of α with its origin at (x, y), that is, $\alpha[x, y](\vec{m}) = \alpha(\vec{v}+\vec{m})$, where $\vec{v} = (x, y)$ and $\vec{m} \in \mathbb{N}^2$. Note that in context we treat $\alpha[x, y]$ as an "alias" for α, that is, if in the description of an assembly sequence a tile t is added to $\alpha[x, y]$ at location $\vec{m}$, we assume the tile t is added to α at location $\vec{v}+\vec{m}$. Similarly, we write $\alpha[x, y \mid k]$ to denote a square $k \times k$ block of tiles consisting of $\alpha[x, y]$ restricted to $\vec{m} \in \{0, 1, \ldots, k-1\}^2$. We refer to $\alpha[x, y \mid k]$ as a *block of size k in α*. Note that as for a subassembly $\alpha[x, y]$, $\alpha[x, y \mid k]$ need not be defined for all locations $\vec{m} \in \{0, 1, \ldots, k-1\}^2$.

Given an assembly or block α, Let M_α denote the matrix of *labels* for the tiles in α, that is, $M_\alpha(\vec{m})$ is the label on $\alpha(\vec{m})$ if $\alpha(\vec{m})$ is defined and has a label, and $M_\alpha(\vec{m})$ is undefined otherwise. For the the tile types of Figures 6.9, 6.10, and 6.11 that are illustrated with a black dot, the label is the parameter "w". For all other tile types, the label is zero.

Given a block β of size k, we define the *south border* as the sequence of tiles $\{t_i\}_{0 \le i < k} \in T^k$ where $t_i = \beta(i, 0)$, provided that $\beta(i, 0)$ is defined for all $0 \le i < k$. Let Σ denote the set of possible glue colors for the tileset T. The *south border glue sequence* is the sequence $\{\text{col}_{t_i}(\vec{u}_N)\}_{0<i<k} \in \Sigma^{n-1}$, i.e., the sequence of glue colors on the *north* sides of the tiles in the south border (excluding the tile at the lower left corner). Similarly we define the *west border* by $t_i = \beta(0, i)$ and the *west border glue sequence* as the corresponding sequence of glue colors on the east sides, $\{\text{col}_{t_i}(\vec{u}_E)\}_{0<i<k}$. The *north border* is the sequence $\{t_i\}_{0 \le i < k}$ defined by $t_i = \beta(i, k-1)$ and the *north border glue sequence* is $\{\text{col}_{t_i}(\vec{u}_N)\}_{0 \le i < k}$. The *east border* is the sequence $\{t_i\}_{0 \le i < k}$ defined by $t_i = \beta(k-1, i)$ and the *east border glue sequence* is $\{\text{col}_{t_i}(\vec{u}_E)\}_{0 \le i < k}$.

Throughout the discussion that follows, let $p > 2$ be a prime and let a, b, and c be integers not congruent to zero modulo p.

Our work will be simplified by exploiting the numerical self-similarity of the generalized Sierpinski carpets and the known facts about their self-assembly. Let T_L denote the tileset defined by Fig. 6.3, let $\mathcal{T}_L = (T_L, \sigma_L, 2)$, and let α_L be its terminal assembly. Assume that each tile is labeled by an integer in $\{0, 1, \ldots, p-1\}$ which is the value w as shown in Fig. 6.3.

As noted in Section 6.2.2, the set of points

$$S = \{(x, y) | M_{\alpha_L}(x, y) \neq 0\}$$

is a generalized Sierpinski carpet. We will refer to the tiles in T_L as the *legacy tiles* and α_L as the *legacy assembly* of S. We define a *legacy n-block* as follows:

- The block $\alpha_L[0, 0 \mid p]$ is a legacy 1-block of size p.
- For $k \geq 1$ and $0 \leq s, t < p$, $\alpha_L[sp^k, tp^k \mid p^k]$ is a legacy n-block of size p^k, where $n = M_{\alpha_L}(s, t)$.

Let $\Sigma = \{0, 1, \ldots, p-1\} \cup \{0, 1, \ldots, p-1\}^2$ denote the set of all glue colors in the legacy tileset. Note that for all tiles except the seed and the border tiles, the glue color is an ordered pair of integers.

We review a number of basic facts about the legacy assembly and the legacy n-blocks, proved in [46]. The first lemma shows that a legacy n-block can be viewed as a legacy 1-block to which a scaling factor of n has been applied to the labels. The next lemma describes the border tiles and border glue sequences.

Lemma 34. *If β_0 is a legacy n-block of size p^k and β_1 is legacy 1-block of size p^k, then $M_{\beta_0}(x, y) \equiv n M_{\beta_1}(x, y) \pmod{p}$.*

Lemma 35. *1. Let β be a legacy n-block of size p^k, and let $\{t_i\}_{0 \leq i < p^k}$ be the south border. The labels on the south border tiles are $M_\beta(t_i) = na^i \mod p$ for $i = 0, 1, \ldots p^k - 1$, and the south border glue sequence is*

$$\begin{aligned} \mathrm{col}_{t_i}(\vec{u}_N) &= (na^i \mod p, na^{i-1} \mod p) \text{ for } 1 \leq i < p^k \\ \mathrm{col}_{t_0}(\vec{u}_N) &= \begin{cases} n & \text{if } t_0 \text{ is a left edge tile} \\ (n, x) & \text{otherwise} \end{cases} \end{aligned} \tag{6.6}$$

for some $x < p$.

2. Let β be a legacy n-block of size p^k, and let $\{t_j\}_{0 \leq j < p^k}$ be the west border. The labels on the west border tiles are $M_\beta(t_j) = nb^j \mod p$ for $j = 0, 1, \ldots p^k - 1$, and the west border

glue sequence is

$$\begin{aligned} \mathrm{col}_{t_j}(\vec{u}_E) &= (nb^j \mod p, nb^{j-1} \mod p) \text{ for } 1 \le j < p^k \\ \mathrm{col}_{t_0}(\vec{u}_E) &= \begin{cases} n & \text{if } t_0 \text{ is a bottom edge tile} \\ (n, y) & \text{otherwise} \end{cases} \end{aligned} \tag{6.7}$$

for some $y < p$.

3. *Let* β *be a legacy* n*-block of size* p^k*, and let* $\{t_i\}_{0 \le i < p^k}$ *be the north border and let* $n_i = M_\beta(t_i)$ *denote the label on* t_i*, for* $0 \le i < p^k$*. Then* $n_0 = n$ *and the labels on the north border tiles satisfy the relation* $bn_i + cn_{i-1} \equiv 0 \pmod{p}$ *for* $1 \le i < p^k$*, and the north border glue sequence is*

$$\begin{aligned} \mathrm{col}_{t_i}(\vec{u}_N) &= (bn_i, cn_{i-1}) \text{ for } 1 \le i < p^k, \text{ and} \\ \mathrm{col}_{t_0}(\vec{u}_N) &= \begin{cases} n & \text{if } t_0 \text{ is a left edge tile} \\ (n, x) & \text{otherwise} \end{cases} \end{aligned} \tag{6.8}$$

for some $x < p$.

4. *Let* β *be a legacy* n*-block of size* p^k*, and let* $\{t_j\}_{0 \le j < p^k}$ *be the east border and let* $n_j = M_\beta(t_j)$ *denote the label on* t_j*, for* $0 \le j < p^k$*. Then* $n_0 = n$ *and the labels on the east border tiles satisfy the relation* $an_j + cn_{j-1} \equiv 0 \pmod{p}$ *for* $1 \le j < p^k$*, and the east border glue sequence is*

$$\begin{aligned} \mathrm{col}_{t_j}(\vec{u}_E) &= (an_j, cn_{j-1}) \text{ for } 1 \le j < p^k, \text{ and} \\ \mathrm{col}_{t_0}(\vec{u}_E) &= \begin{cases} n & \text{if } t_0 \text{ is a bottom edge tile} \\ (n, y) & \text{otherwise} \end{cases} \end{aligned} \tag{6.9}$$

for some $y < p$.

It is a consequence of the facts listed above that for given n and k, all legacy n-blocks of size p^k must be identical except for the south and west border tiles, since every such block has the same south and west border glue sequences, and these completely determine the remaining tiles in the block. Note also that for any legacy n-block β of size p^k,

$$M_\beta(0,0) = M_\beta(p^k - 1, 0) = M_\beta(0, p^k - 1) = M_\beta(p^k - 1, p^k - 1) = n.$$

Another fact we will need is that 0-blocks are never adjacent and never occupy the border of a larger block. Parts (a) and (b) can be found in [46] and (c) is proven in [63].

Lemma 36. *Let β be a legacy 1-block of size p, that is, β is the kernel of a generalized Sierpinski carpet. Then*

(a) for all $0 \leq i < p$, $M_\beta(i,0) \neq 0$ and $M_\beta(i,p-1) \neq 0$.

(b) for all $0 \leq j < p$, $M_\beta(0,j) \neq 0$ and $M_\beta(p-1,j) \neq 0$.

(c) for any $0, i, j < p$, if $M_\beta(i,j) = 0$ then $M_\beta(i+1,j) \neq 0$, $M_\beta(i-1,j) \neq 0$, $M_\beta(i,j+1) \neq 0$, and $M_\beta(i,j-1) \neq 0$

Given a sequence s of p^k tiles, we say that

1. s has a *valid n-block south border glue sequence* if it satisfies 6.6.
2. s has a *valid n-block west border glue sequence* if it satisfies 6.7.
3. s has a *valid n-block north border glue sequence* if it satisfies 6.8.
4. s has a *valid n-block east border glue sequence* if it satisfies 6.9.

The following observation is a simple consequence of Lemma 35, asserting that valid south or west glue sequences are periodic.

Lemma 37. *Suppose s is a sequence of p^k tiles with a valid n-block south (respectively, west) border glue sequence. Let $0 < j < k$ and let $i \leq p^k - p^j$. Let m be the label on the tile $s(i)$. Then the subsequence $s(i), \ldots, s(i+p^j-1)$ has a valid m-block south (respectively, west) border glue sequence.*

We next return to the assembly of the *readers* and *decrementers* described in Lemma 33. Note that the south glues of the ruler ground tile set of Fig. 6.8 are designed to match the glues of n-block north border glue sequence, so that given a block β with a valid n-block north border glue sequence, plus a ruler trigger as in Fig. 6.11e, a ruler ground sequence will self-assemble above the north border of β. Likewise, a vertical ruler ground sequence will self-assemble along

the east side of a block with a valid east border glue sequence. A block of size p^k consisting of just the ruler trigger and the horizontal and vertical ruler ground sequences will be called a *pre-0-block* of size p^k with parameters (x, y), where x and y are the values appearing in the colors of the ruler trigger tile as shown in Fig. 6.11e.

The next step is to observe that once the reader and decrementer assembles on the ruler ground sequence, the the main reader and decrementer bar has a valid west border glue sequence and therefore interacts correctly with the interior tiles of Fig. 6.11d. Likewise, the reader and decrementer bar for the vertical ruler has a valid south border glue sequence.

Lemma 38. *(a) Assume the notation and hypotheses of Lemma 33(a). Assume also that the reader trigger t has south glue $\mathrm{col}_t(\vec{u}_S) = (v, y)$ for some $v, y < p$. Then the reader and decrementer bar has a valid w-block west border glue sequence, where $w = bv + cy$.*

(b) Assume the notation and hypotheses of Lemma 33(b). Assume also that the reader trigger t has west glue $\mathrm{col}_t(\vec{u}_W) = (u, x)$ for some $u, x < p$. Then the reader and decrementer bar has a valid w-block south border glue sequence, where $w = au + cx$.

Based on the above we can define the following:

A *partial* 0*-block* of size p^k and parameters (x, y, v) is the result of applying Lemma 38(a) to a pre-0-block with parameters (x, y), where (v, y) is the south glue of the reader trigger. (Note that although we refer to the result as a block of size p^k, in fact it includes the west border of the adjacent block to the right.)

A *full* 0*-block* of size p^k and parameters (x, y, u, v) is the result of applying Lemma 38(b) to a partial 0-block with parameters (x, y, v), where (u, x) is the west glue of the reader trigger.

Finally, a 0*-block* is the restriction of a full 0-block to $0 \le i, j < p^k$.

For $n \neq 0$, let β_L be a legacy n-block. A *pre n-block of size p^k* is a block β of size p^k such that

1. For $0 \le i < p$, $\beta(i, 0)$ is defined and $M_\beta(i, 0) = M_{\beta_L}(i, 0)$.
2. The south border of β has a valid n-block south glue sequence.
3. For $0 \le j < p$, $\beta(0, j)$ is defined and $M_\beta(0, j) = M_{\beta_L}(0, j)$.

4. The west border of β has a valid n-block west glue sequence.

5. For all $i, j > 0$, $\beta(i, j)$ is undefined.

For $n \neq 0$, an n-block β is defined as a transformation of a legacy n-block β_L, with the following properties.

1. If $\beta_L(i, j) \neq 0$, then $\beta(i, j)$ is defined and $M_\beta(i, j) = M_{\beta_L}(i, j)$.

2. If $\beta_L(i, j) = 0$, choose i', j', k' such that $\beta' = \beta_L[i', j' \mid p^{k'}]$ is the maximal legacy 0-block containing (i, j); then $\beta[i', j' \mid p^{k'}]$ is a 0-block.

3. The north border has a valid n-block north glue sequence.

4. The east border has a valid n-block east glue sequence.

The algorithm given in Fig. 6.12 gives an assembly sequence for an n-block of size p, given a pre-n-block of size p. Note that the labels on all tiles in KernelBlock are the same as those on a legacy n-block and that the east border has a valid east border glue sequence and the north border has a valid north border glue sequence.

We can now present the main lemma from which Theorem 38 follows, which inductively defines the construction of the infinite generalized Sierpinski carpet.

Definition 6. *Let M denote the $p \times p$ matrix of labels for $\alpha_L[0, 0 \mid p]$. Let $s, t < p$ and $k > 0$. An assembly α is* (s, t)-correct *at level p^{k+1} if the following conditions hold.*

1. *For all (i, j) with $i \leq s$ and $j < t$, $\alpha[ip^k, jp^k \mid p^k]$ is a $M(i, j)$-block.*

2. *For all (i, j) with $p > i > s$ and $j < t - 1$, $\alpha[ip^k, jp^k \mid p^k]$ is a $M(i, j)$-block.*

3. *For all (i, j) with $i \leq s$ and $j = t$, and for all (i, j) with $p > i > s$ and $j = t - 1$*

   (a) *if $M(i, j) \neq 0$, $\alpha[ip^k, jp^k \mid p^k]$ is a $M(i, j)$-block*

   (b) *if $M(i, j) = 0$, $\alpha[ip^k, jp^k \mid p^k]$ is a partial 0-block.*

4. *For all other $i, j < p$, the block $\alpha[ip^k, jp^k \mid p^k]$ is completely undefined.*

5. *For all $i, j \geq p^{k+1}$, $\alpha(i, j)$ is undefined.*

KernelBlock(n, α, i, j)
parameters: an assembly α such that $\alpha[i, j \mid p]$ is a pre-n-block of size p
returns: an assembly α' such that $\alpha \to \alpha'$ and $\alpha'[i, j \mid p]$ is an n-block of size p

Let $\beta = \alpha[i, j \mid p]$

Let M denote the matrix of labels for $\alpha_L[0, 0 \mid p]$

for $j = 1, \ldots, p-1$

  for $i = 1, \ldots, p-1$

    Let $x = M(i-1, j)$, $y = M(i, j-1)$, $z = M(i-1, j-1)$

    Let $w = ax + by + cz \pmod{p}$.

    Case 1: $x \neq 0, y \neq 0$ (implies $w \neq 0$).

      Place the appropriate legacy tile at $\beta(i, j)$.

    Case 2: $x \neq 0, y = 0$ (implies $w \neq 0$).

      Place a vertical ruler end (Fig. 6.11(h)) at $\beta(i, j)$.

    Case 3: $w = 0$ (implies $x \neq 0, y \neq 0$).

      Place a ruler trigger (Fig. 6.11(e)) at $\beta(i, j)$.

    Case 4: $x = 0$.

      Subcase 1: $y \neq 0$. Place a horizontal ruler end (Fig. 6.11(f)) at $\beta(i, j)$.

      Subcase 2: $y = 0$. Place a special ruler end (Fig. 6.11(j)) at $\beta(i, j)$.

Figure 6.12: Assembly sequence for an n-block of size p.

Lemma 39. *(a) For every $k \geq 0$, $s, t < p$, and $n \neq 0$, if α is an assembly and $\alpha[sp^k, tp^k]$ is a pre-n-block in α of size p^k, there is an assembly sequence $\alpha \to \alpha'$ such that $\alpha'[sp^k, tp^k \mid p^k]$ is an n-block, and such that no new tiles are placed outside of $\alpha[sp^k, tp^k \mid p^k]$*

(b) For $k > 1$, if α is an assembly for which $\alpha[0, 0 \mid p^{k-1}]$ is a 1-block of size p^{k-1}, $\alpha(i, j)$ is undefined for $i, j \geq p^{k-1}$, and for which the south and west borders consist only of legacy edge tiles (Fig. 6.11b–6.11c), then there is an assembly α' with $\alpha \to \alpha'$ such that $\alpha'[0, 0 \mid p^k]$ is a 1-block of size p^k, $\alpha(i, j)$ is undefined for $i, j \geq p^k$, and for which the south and west borders consist only of legacy edge tiles.

Proof. Let M denote the $p \times p$ matrix of labels for $\alpha_L[0, 0 \mid p]$.

If $k = 1$, then for (a) the required assembly sequence is given by the algorithm in Fig. 6.12. For (b), let α be the assembly with $|\mathrm{dom}\,(\alpha)| = 1$ and $\alpha(0, 0)$ equal to the legacy seed tile. Then there is an assembly sequence in which we attach the legacy bottom edge tiles up to $(p-1, 0)$ and the legacy left edge tiles up to $(0, p-1)$, and then apply the algorithm in Fig. 6.12.

Let $k \geq 1$ and assume for an induction that (a) and (b) hold for all $k' \leq k$. Let $n \neq 0$ and let α' be an assembly such that $\alpha'[sp^{k+1}, tp^{k+1}]$ is a pre-n-block of size p^{k+1}. Let α denote $\alpha'[sp^{k+1}, tp^{k+1}]$, the subassembly within which all our work will take place.

We inductively define a sequence of assemblies

$$\alpha \to \alpha_{0,0} \to \alpha_{1,0} \cdots \to \alpha_{p-1,0} \to \alpha_{0,1} \to \alpha_{1,1} \cdots \to \alpha_{p-1,1} \to \alpha_{0,2} \cdots \to \alpha_{p-1,p-1}$$

maintaining the invariant that $\alpha_{i,j}$ is (i,j)-correct at level p^k.

1. By Lemma 37, $\alpha[0,0]$ is itself a pre-n-block of size p^k. Let $\alpha_{0,0}$ be the result of applying the induction hypothesis to $\alpha[0,0]$; then $\alpha[0,0 \mid p^k]$ is a n-block, and $\alpha_{0,0}$ is $(0,0)$-correct at level p^k.

2. For $i = 1, \ldots, p-1$:

   Let $x = M(i-1,0)$, and $w = M(i,0)$. Note $x \neq 0$ and $w \neq 0$. Let $\beta = \alpha[ip^k, 0 \mid p^k]$. By Lemma 37, the south border of β has a valid w-block south border glue sequence. $\alpha[(i-1)p^k, 0 \mid p^k]$ is an x-block of size p^k, and therefore has a valid x-block east border glue sequence. Define the subassembly β' as follows: for the next $p^k - 1$ steps in the assembly sequence, attach the appropriate legacy tiles at $\beta(0,1), \beta(0,2), \ldots \beta(0, p^k - 1)$. Now β' has a valid w-block west border glue sequence, i.e., $\beta'[0,0]$ is a pre-w-block of size p^k. Let $\alpha_{i,0}$ be the result of applying the induction hypothesis to β'.

3. For $j = 1, \ldots, p-1$:

   (a) Let $y = M(0,j-1)$, and $w = M(0,j)$. Note $y \neq 0$ and $w \neq 0$. Let $\beta = \alpha[0, jp^k \mid p^k]$. By Lemma 37, the west border of β' has a valid w-block west border glue sequence. $\alpha[0, (j-1)p^k \mid p^k]$ is a y-block of size p^k and therefore has a valid y-block north border glue sequence. Define the subassembly β' as follows: for the next p^k steps in the assembly sequence, attach the appropriate legacy tiles at $\beta(1,0), \beta(2,0), \ldots \beta(p^k - 1, 0)$. Now β' has a valid w-block south border glue sequence, i.e., β' is a pre-w-block of size p^k. Let $\alpha_{0,j}$ be the result of applying the induction hypothesis to β'.

(b) For $i = 1, \ldots, p-1$:

Let $x = M(i-1, j)$, $y = M(i, j-1)$ $z = M(i-1, j-1)$, and $w = M(i, j)$. Let $\beta = \alpha[ip^k, jp^k \mid p^k]$.

Case 1: $x \neq 0, y \neq 0$ (implies $w \neq 0$). Then $\alpha[(i-1)p^k, jp^k \mid p^k]$ is an x-block of size p^k and $\alpha[ip^k, (j-1)p^k \mid p^k]$ is a y-block of size p^k. Define the subassembly β' by attaching the appropriate legacy tiles along the south and west border of β, and let $\alpha_{i,j}$ be the result of applying the induction hypothesis to β'.

Case 2: $x \neq 0, y = 0$ (implies $w \neq 0$). Then $\alpha[(i-1)p^k, jp^k \mid p^k]$ is an x-block of size p^k and $\alpha[ip^k, (j-1)p^k \mid p^k]$ is a partial 0-block of size p^k. Define the subassembly β' as follows: First attach a vertical reader trigger with parameters (x, z, w), as shown in Fig. 6.11(h)–(i), at $\beta(0, 0)$. Then apply Lemma 38(b) to $\alpha[ip^k, (j-1)p^k \mid p^k]$, which provides the south border of β'. Then attach appropriate legacy tiles as the west border; now β' is a pre-w block of size p^k. Let $\alpha_{i,j}$ be the result of applying the induction hypothesis to β'.

Case 3: $w = 0$ (implies $x \neq 0, y \neq 0$). Define the assembly β' as follows: First attach a ruler trigger with parameters (x, y, z), as shown in Fig. 6.11(e), at $\beta(0, 0)$, and then assemble a ruler ground sequence (Fig. 6.8) along the south border of β. Likewise assemble a vertical ruler ground sequence along the west border of β. Then β' is a pre-0-block of size p^k.

Case 4: $x = 0$.

Subcase 1: $y \neq 0$. Define the subassembly β' as follows: First, attach a horizontal reader trigger with south glue (y, z), as shown in Fig. 6.11(f)–(g), at $\beta(0, 0)$. Then apply Lemma 38(a) to the block $\alpha[(i-1)p^k, jp^k \mid p^k]$, which constructs the west border of β' with a valid w-block west border glue sequence. Attach appropriate legacy tiles along the south border of β'. Now β' is a pre-w-block of size p^k. Let $\alpha_{i,j}$ be the result of applying the induction hypothesis to β'.

Subcase 2: $y = 0$. Define the subassembly β' as follows: First attach an instance of the

special reader trigger of Fig. 6.11(j)–(k) at $\beta(0,0)$. Apply Lemma 38(a) to the block $\alpha[(i-1)p^k, jp^k \mid p^k]$, which constructs the west border of β'. Then apply Lemma 38(b) to the block $\alpha[ip^k, (j-1)p^k \mid p^k]$, which constructs the south border of β'. Now β' is a pre-w-block of size p^k. Let $\alpha_{i,j}$ be the result of applying the induction hypothesis to β'.

Since $\alpha_{p-1,p-1}$ is $(p-1, p-1)$-correct at level p^{k+1}, (a) is established. For (b), it is sufficient to extend the legacy edge tiles to length p^{k+1}, let the resulting assembly be denoted $\alpha_{0,0}$, and apply the argument above starting with step 2. □

To complete the proof of Theorem 38, note that Lemma 39(b) shows the existence of an assembly sequence for the in-place approximation. We omit the verification that the assembly sequence is locally deterministic and therefore directed. The dimension of the assembled structure is established using Theorem 37; in particular see the remarks following Corollary 8.

6.5 Conclusion

We have shown that for every generalized Sierpinski carpet F there exists an *in-place approximation* of F, that is, a set $X \supseteq F$ with the same fractal dimension as F that strictly self-assembles in such a way that those tiles corresponding to the set F are recognizably labeled. Moreover, there is an algorithm that produces the tileset for the approximation X from the four parameters a, b, c, and p specifying F. As part of the construction we introduced *rulers* and the corresponding *readers* as a new way to control the growth of an assembly. To analyze the dimension of the approximation X we introduced the concept of an *embedded fractal* and showed that whenever a set G is embedded in a self-similar fractal F, the dimension of the resulting set is always $\max(\mathrm{Dim}_\zeta(G), \mathrm{Dim}_\zeta(F))$.

It is unknown whether any self-similar fractal self-assembles in the aTAM, and in fact it *is* known that the discrete Sierpinski triangle cannot self-assemble [51]. Lutz and Shutters [53] showed in addition that there is a limitation on how closely the Sierpinski triangle can be approximated: if X is any set that strictly self-assembles, and S is the Sierpinski triangle, then $\mathrm{Dim}_\zeta(X \,\Delta\, S) \geq \mathrm{Dim}_\zeta(S)$, where Δ denotes the symmetric difference. In particular, any

in-place approximation X of S has the property that the set of "extra" points $X \setminus S$ used in the approximation has dimension at least as great as the dimension of S itself.

Theorem 37 suggests the following line of reasoning: if a discrete self-similar fractal F does not strictly self-assemble, it may be possible to construct an in-place approximation X as we have done for the generalized Sierpinski carpets. X must add some additional points to the empty squares in F. Theorem 37 shows that if only a single point is added to each of the empty squares of F, then $\text{Dim}_\zeta(X \setminus F) = \text{Dim}_\zeta(F)$.

Conjecture 3. *For every self-similar fractal $F \subset \mathbb{Z}^2$ either F strictly self-assembles, or, for every in-place approximation X of F, $\text{Dim}_\zeta(X \setminus F) \geq \text{Dim}_\zeta(F)$.*

CHAPTER 7. General Conclusion

This dissertation has addressed several problems at the interface between computation and biology. In particular we looked at the multi-state perfect phylogeny problem and the tree compatibility problem in computational phylogenetics, and we also looked at approximating fractal structures in the tile assembly model. In this chapter we give some concluding remarks with regards to these results along with some ideas for further study.

In Chapter 2 we looked at multi-state character compatibility, and gave an alternative characterization of the sets of pairwise compatible three-state characters that are incompatible. We also gave an $O(m^2n + p)$ time algorithm to enumerate all p minimal obstruction sets to the compatibility of a set of 3-state characters. One direction for future research is to design an efficient algorithm for counting or enumerating all perfect phylogenies for a compatible set of three-state characters. Gusfield and Wu [37] reduced the three-state perfect phylogeny problem to 2-SAT. Counting all solutions for an instance of 2-SAT is #P-complete [80, 49]. However, it is possible that the characterization of compatible sets of three-state characters given here can be used to design more efficient algorithms for counting and enumerating three-state perfect phylogenies.

Also in Chapter 2, we proved quadratic lower bounds on the maximum cardinality of a minimal obstruction set to the compatibility of a set of r-state characters. Finding any upper bound on the maximum cardinality of such a set remains a central open question in computational phylogenetics. However, since DNA sequences naturally give rise to four-state characters, the four-state character compatibility problem is of particular importance. Further analysis of the characterization of four-state character compatibility given by Kannan and Warnow [43] may lead to an upper bound for the four-state case.

In Chapter 3 we looked at the tree compatibility problem. In particular, we studied the

agreement supertree approach for combining rooted phylogenetic trees when the input trees do not fully agree on the relative positions of the taxa. We considered two approaches for dealing with such conflicting input trees: edge contraction and taxon removal. For both problems we gave FPT algorithms for the problem parameterized by the number of input trees and the number of edges contracted or taxa removed. It is known that taxon removal is fixed-parameter intractable if only the number of taxa removed is parameterized. However, no such analogous result is known for edge contraction. Another direction for future research would be to extend our results for agreement to the compatibility case where the supertree is not required to be homeomorphic to the input trees, i.e., it is allowed to be a refinement of each of the input trees. However, the exponential lower bound on the size of the obstructions for compatible supertrees mentioned in Chapter 3 makes it likely that entirely new techniques will be needed for developing FPT algorithms form compatible supertrees. One further direction would be to study the case for unrooted trees. The difficulty here would be in finding an analogue to the auxiliary graph described in Section 3.3.1 for unrooted input trees.

In Chapters 5 and 6 we studied the approximate self-assembly of the Sierpinski triangle and the Sierpinski carpet. Several interesting open questions about the self-assembly of fractal structures remain. The most central one being to exhibit a fractal that can strictly self-assemble, or prove that none exists. Another related line of research is to find limits on approximating the Sierpinski carpet analogous to what was given here for the Sierpinski triangle. Although studying the self-assembly of fractals is interesting in its own right, the study of self-similar fractals given here yielded many insights into more general self-assembly systems. In particular, the notion of conditional determinism for proving the correctness of our approximation of the Sierpinski triangle, and the rulers and readers used in the construction of the Sierpinski carpet. It is likely that further study of the self-assembly of fractal structures will give rise to more insights into general self-assembly systems.

BIBLIOGRAPHY

[1] Leonard Adleman, Qi Cheng, Ashish Goel, and Ming-Deh Huang. Running time and program size for self-assembled squares. In *Proceedings of the thirty-third annual ACM symposium on Theory of computing*, STOC '01, pages 740–748. ACM, 2001.

[2] Richa Agarwala and David Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1994.

[3] Alfred V. Aho, Yehoshua Sagiv, Thomas G. Szymanski, and Jeffrey D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal on Computing*, 10(3):405–421, 1981.

[4] Martin T. Barlow and S. James Taylor. Defining fractal subsets of $\mathbb{Z}^d$. In *Proceedings London Mathematical Society*, volume 64, pages 125–152, 1992.

[5] P. H. Barrett. A transcription of Darwin's first notebook on "Transmutation of species". *Bulletin of the Museum of Comparative Zoology*, 122(247–296), 1960.

[6] Vincent Berry and François Nicolas. Maximum agreement and compatible supertrees. *Journal of Discrete Algorithms*, 5(3):564–591, 2007.

[7] Olaf R. P. Bininda-Emonds, editor. *Phylogenetic supertrees: Combining information to reveal the tree of life*, volume 4 of *Computational Biology*. Springer, 2004.

[8] Hans Bodlaender, Mike Fellows, and Tandy Warnow. Two strikes against perfect phylogeny. In *Automata, Languages and Programming*, volume 623 of *Lecture Notes in Computer Science*, pages 273–283. Springer, 1992.

[9] Boris A. Bondarenko. *Generalized Pascal triangles and pyramids, their fractals, graphs and applications*. The Fibonacci Association, 1993.

[10] Nicolas Bousquet, Jean Daligault, and Stéphan Thomassé. Multicut is FPT. In *STOC 2011*, pages 459–468. ACM, 2011.

[11] David Bryant. Optimal agreement supertrees. In *Computational Biology*, volume 2066 of *Lecture Notes in Computer Science*, pages 24–31. Springer, 2001.

[12] Peter Buneman. The recovery of trees from measurements of dissimilarity. In *Mathematics in the Archeological and Historical Sciences*, pages 387–395. Edinburgh University Press, 1971.

[13] Peter Buneman. A characterization of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.

[14] Eugene Cahen. Sur la fonction $\zeta(s)$ de Riemann et sur des fonctions analogues. *Annales de l'École Normale supérieure*, 11(3):75–164, 1894.

[15] Alessandra Carbone and Nadrian C. Seeman. A route to fractal DNA-assembly. *Natural Computing*, 1:469–480, 2002.

[16] Alessandra Carbone and Nadrian C. Seeman. Coding and geometrical shapes in nanostructures: a fractal DNA-assembly. *Natural Computing*, 2:133–151, 2003.

[17] Q. Cheng, A. Goel, and P. Moisset. Optimal self-assembly of counters at temperature two. In *Proceedings of the 1st Foundations of Nanoscience: Self-Assembled Architectures and Devices, FNANO'04, Snowbird, UT*, April 2004.

[18] H. Colonius and H. H. Schulze. Tree structures for proximity data. *British Journal of Mathematical and Statistical Psychology*, 34(2):167–180, 1981.

[19] Dekker, M. C. H. Reconstruction methods for derivation trees. Master's thesis, Vrije Universiteit, Amsterdam, Netherlands, 1986.

[20] Max Dietrich, Catherine McCartin, and Charles Semple. Bounding the maximum size of a minimal definitive set of quartets. *Information Processing Letters*, 112(16):651–655, 2012.

[21] David Doty, Xiaoyang Gu, Jack H. Lutz, Elvira Mayordomo, and Philippe Moser. Zeta-dimension. In *Mathematical Foundations of Computer Science*, volume 3618 of *Lecture Notes in Computer Science*, pages 283–294. Springer, 2005.

[22] Rod G. Downey and Michael R. Fellows. *Parameterized complexity*, volume 3. Springer, New York, 1999.

[23] Andreas Dress and Michael Steel. Convex tree realizations of partitions. *Applied Mathematics Letters*, 5(3):3–6, 1992.

[24] George F. Estabrook, C. S. Johnson, JR., and F. R. McMorris. A mathematical foundation for the analysis of cladistic character compatibility. *Mathematical Biosciences*, 29(1-2):181–187, 1976.

[25] George F. Estabrook and F. R. McMorris. When are two qualitative taxonomic characters compatible? *Journal of Mathematical Biology*, 4:195–200, 1977.

[26] Leonhard Euler. Variae observationes circa series infinitas. *Commentarii Academiae Scientiarum Imperialis Petropolitanae*, 9:160–188, 1737.

[27] David Fernández-Baca. The perfect phylogeny problem. In *Steiner Trees in Industry*, pages 203–234. Kluwer, 2001.

[28] David Fernández-Baca, Sylvain Guillemot, Brad Shutters, and Sudheer Vakati. Fixed-parameter algorithms for finding agreement supertrees. In *Combinatorial Pattern Matching*, volume 7354 of *Lecture Notes in Computer Science*, pages 373–384, 2012.

[29] Walter M. Fitch. Toward finding the tree of maximum parsimony. In *Proceedings of the 8th International Conference on Numerical Taxonomy*, pages 189–230, 1975.

[30] Walter M. Fitch. On the problem of discovering the most parsimonious tree. *The American Naturalist*, 111(978):223–257, 1977.

[31] A. D. Gordon. Consensus supertrees: The synthesis of rooted trees containing overlapping sets of labelled leaves. *Journal of Classification*, 9:335–348, 1986.

[32] Ronald L. Graham, Donald E. Knuth, and Oren Patashnik. *Concrete Mathematics*. Addison-Wesley, 1994.

[33] S. Grünewald and K. T. Huber. Identifying and defining trees. In *Reconstructing Evolution: New Mathematical and Computational Advances*. Oxford University Press, 2007.

[34] Sylvain Guillemot and Vincent Berry. Fixed-parameter tractability of the maximum agreement supertree problem. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(2):342–353, 2010.

[35] Dan Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21(1):19–28, 1991.

[36] Dan Gusfield. The multi-state perfect phylogeny problem with missing and removable data: Solutions via integer-programming and chordal graph theory. *Journal of Computational Biology*, 17(3):383–399, 2010.

[37] Dan Gusfield and Yufeng Wu. The three-state perfect phylogeny problem reduces to 2-SAT. *Communications in Information & Systems*, 9(4):195–301, 2009.

[38] Rob Gysel, Fumei Lam, and Dan Gusfield. Constructing perfect phylogenies and proper triangulations for three-state characters. In *Algorithms in Bioinformatics*, volume 6833 of *Lecture Notes in Computer Science*, pages 104–115. Springer, 2011.

[39] Michel Habib and Thu-Hien To. On a conjecture of compatibility of multi-states characters. In *Algorithms in Bioinformatics*, volume 6833 of *Lecture Notes in Computer Science*, pages 116–127. Springer, 2011.

[40] Viet Tung Hoang and Wing-Kin Sung. Improved algorithms for maximum agreement and compatible supertrees. *Algorithmica*, 59(2):195–214, 2011.

[41] I. Hueter and Y. Peres. Self-affine carpets on the square lattice. *Combinatorics, Probability and Computing*, 6:197–204, 1997.

[42] Jesper Jansson, Joseph H.-K. Ng, Kunihiko Sadakane, and Wing-Kin Sung. Rooted maximum agreement supertrees. *Algorithmica*, 43(4):293–307, 2005.

[43] Sampath Kannan and Tandy Warnow. Inferring evolutionary history from DNA sequences. *SIAM Journal on Computing*, 23(4):713–737, 1994.

[44] Sampath Kannan and Tandy Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. *SIAM Journal on Computing*, 26(6):1749–1763, 1997.

[45] Ming-Yang Kao. *Encyclopedia of algorithms*. Springer, 2007.

[46] Steven M. Kautz and James I. Lathrop. Self-assembly of the Sierpinski carpet and related fractals. In *DNA Computing and Molecular Programming*, volume 5877 of *Lecture Notes in Computer Science*, pages 78–87. Springer, 2009.

[47] Steven M. Kautz and Brad Shutters. Supplementary materials for self-assembling rulers for approximating generalized Sierpinski carpets. www.cs.iastate.edu/˜shutters/saragsc, 2012.

[48] Steven M. Kautz and Brad Shutters. Self-assembling rulers for approximating generalized Sierpinski carpets. *Algorithmica*, to appear.

[49] Dexter C. Kozen. *The Design and Analysis of Algorithms*. Springer, Berlin, 1992.

[50] Fumei Lam, Dan Gusfield, and Srinath Sridhar. Generalizing the splits equivalence theorem and four gamete condition: perfect phylogeny on three-state characters. *SIAM Journal on Discrete Mathematics*, 25(3):1144–1175, 2011.

[51] James I. Lathrop, Jack H. Lutz, and Scott M. Summers. Strict self-assembly of discrete Sierpinski triangles. *Theoretical Computer Science*, 410:384–405, 2009.

[52] C. Randal Linder and Loren H. Rieseberg. Reconstructing patterns of reticulate evolution in plants. *American Journal of Botany*, 91(10):1700–1708, 2004.

[53] Jack H. Lutz and Brad Shutters. Approximate self-assembly of the Sierpinski triangle. *Theory of Computing Systems*, 51(3):372–400, 2012.

[54] Jack H. Lutz and Brad Shutters. Supplementary materials for approximate self-assembly of the Sierpinski triangle. http://www.cs.iastate.edu/˜shutters/asast, 2012.

[55] Wayne P. Maddison. Gene trees in species trees. *Systematic Biology*, 46(3):523–536, 1997.

[56] Dániel Marx and Igor Razgon. Fixed-parameter tractability of multicut parameterized by the size of the cutset. In *STOC 2011*, pages 469–478. ACM, 2011.

[57] Christopher A. Meacham. Theoretical and computational considerations of the compatibility of qualitative taxonomic characters. In *Numerical Taxonomy*, volume G1 of *Nato ASI series*. Springer, 1983.

[58] Meei Pyng Ng and Nicholas C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1–2):19–31, 1996.

[59] Rolph Niedermeier and Peter Rossmanith. An efficient fixed-parameter algorithm for 3-hitting set. *Journal of Discrete Algorithms*, 1(1):89–102, 2003.

[60] Lars Olsen. Distribution of digits in integers: fractal dimension and zeta functions. *Acta Arithmetica*, 105(3):253–277, 2002.

[61] Matthew J. Patitz. Simulation of self-assembly in the abstract tile assembly model with ISU TAS. *CoRR*, abs/1101.5151, 2011.

[62] Matthew J. Patitz and Scott M. Summers. Self-assembly of discrete self-similar fractals. *Natural Computing*, 9:135–172, 2010.

[63] M. Razpet. The lucas property of a number array. *Discrete Mathematics*, 248:157–168, 2002.

[64] Mihail C. Roco, Chad A. Mirkin, and Mark C. Hersam. Nanotechnology research directions for societal needs in 2020, NSF/WTEC Report, Springer, 2011.

[65] Paul W. K. Rothemund. *Theory and experiments in algorithmic self-assembly*. PhD thesis, University of Southern California, Los Angeles, California, 2001.

[66] Paul W. K Rothemund, Nick Papadakis, and Erik Winfree. Algorithmic self-assembly of DNA Sierpinski triangles. *PLoS Biology*, 2(12), 2004.

[67] Paul W. K. Rothemund and Erik Winfree. The program-size complexity of self-assembled squares. In *STOC*, pages 459–468. ACM, 2000.

[68] Celine Scornavacca. *Supertree methods for phylogenomics*. PhD thesis, Univ. of Montpellier II, Montpellier, France, 2009.

[69] Nadrian C. Seeman. Nucleic-acid junctions and lattices. *Journal of Theoretical Biology*, 99:237–247, 1982.

[70] Nadrian C. Seeman. DNA in a material world. *Nature*, 421:427–431, 2003.

[71] Charles Semple and Mike Steel. *Phylogenetics*. Oxford Lecture Series in Mathematics and its Applications. Oxford University Press, 2003.

[72] Brad Shutters and David Fernández-Baca. A simple characterization of the minimal obstruction sets for three-state perfect phylogenies. *Applied Mathematics Letters*, 25(9):1226–1229, 2012.

[73] Brad Shutters, Sudheer Vakati, and David Fernández-Baca. Improved lower bounds on the compatibility of quartets, triplets, and multi-state characters. In *Algorithmis in Bioinformatics*, volume 7534 of *Lecture Notes in Bioinformatic*, pages 190–200. Springer, 2012.

[74] Brad Shutters, Sudheer Vakati, and David Fernández-Baca. Incompatible sets of quartets, triplets, and characters. *Algorithms for Molecular Biology*, to appear.

[75] Warclaw Sierpiński. Sur une courbe dont tout point est un point de ramification. *Compte Rendus hebdomadaires des séance de l'Académie des Science de Paris*, 160:302–305, 1915.

[76] David Soloveichik and Erik Winfree. Complexity of self-assembled shapes. *SIAM Journal on Computing*, 36:1544–1569, 2007.

[77] Michael Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9(1):91–116, 1992.

[78] Mike Steel. Personal communications, 2012.

[79] Ian Stewart. Four encounters with Sierpinski's gasket. *The Mathematical Intelligencer*, 17(1):52–64, 1995.

[80] Leslie G. Valiant. The complexity of enumeration and reliability problems. *SIAM Journal on Computing*, 8(3):410–421, 1979.

[81] Hao Wang. Proving theorems by pattern recognition – ii. *The Bell System Technical Journal*, 40:1–41, 1961.

[82] Hao Wang. Dominoes and the AEA case of the decision problem. In *Mathematical Theory of Automata*, 1962.

[83] Stephen J. Willson. The equality of fractional dimensions for certain cellular automata. *Physica D*, 24:179–189, 1987.

[84] Erik Winfree. *Algorithmic self-assembly of DNA*. PhD thesis, California Institute of Technology, Pasadena, California, 1998.