UNIVERSITY OF CAMERINO

School of Advanced Studies

Doctor of Philosophy in Sciences and Technology

Computer Science - XXXVI Cycle

Tools for Computational Biology

Brauer Monoid and Phylogenetic Networks

Supervisors

Prof. Emanuela Merelli

PhD Candidate

Daniele Marchei

Academic Year 2023-2024

Declaration

I herewith declare that I have produced this paper under the supervision of Prof. Emanuela Merelli (University of Camerino), without the prohibited assistance of third parties and without making use of aids, other than those specified. Notions taken over directly or indirectly from other sources have been identified as such. This paper has not previously been presented in an identical or similar form to any other Italian or foreign examination board.

Daniele Marchei
2024

* This dissertation is presented in partial fulfillment of the requirements for **Ph.D. degree** in the School of Advanced Studies of University of Camerino.

Acknowledgements

First of all, I would like to thank my supervisior Emanuela Merelli, for her patience and invaluable mentorship she has given me during my time at Unicam. I would also like to thank Andrew Francis for welcoming me at the Western Sydney University during my PhD abroad period, showing me how contagious the passion for research can be.

A big thanks also goes to James East, Matthias Fresacher, Alfilgen Sebandal and Azeef Parayil Ajmal for their helpful suggestions and discussions, including Matteo Belenchia for his invaluable proof-read of this thesis. I would also like to thank James Mitchell for the clarification of the time complexity for Algorithm 13 of [East et al., 2019].

Lastly, I would like to thank Attila Egri-Nagy and Natasha Jonoska for their revision of this manuscript and for the many quality improvement suggestions.

Abstract

In this thesis, we explore the Brauer monoid and phylogenetic networks. The former is an algebraic structure of interest in semigroup theory which is closely related to perfect matchings and has been adapted as a tool for studying RNA secondary structures, while the latter is a model for evolutionary relationships between a given set of species. We begin by exploring the factorization problem for the Brauer monoid, i.e. how its elements could be decomposed into simpler ones. In particular, we present two methods: a naive one that uses heuristics, which was useful for bootstrapping the research for the paper [Marchei and Merelli, 2022], but did not always return the optimal solution, and another method that solves this problem optimally in $\mathcal{O}(N^4)$ and, according to our knowledge, is the first one to run in polynomial time. We then use these methods to extend a work, introduced in 1993 by Kauffman and Magarshak, and show how some RNA secondary structures are reflected in the Brauer monoid factorization. We then move on to study phylogenetic networks and describe how they can be encoded using the language of set covers (a generalization of perfect matchings). This turns out to be a useful tool not only for describing well-known classes of networks by just imposing new axioms to the covers, but also useful for studying new classes that have non-trivial intersections with existing ones. We explore one of such classes and name it *spinal networks*.

Contents

List of Figures

List of Tables

List of Algorithms

List Of Publications

- Marchei, D. and Merelli, E. (2022). RNA secondary structure factorization in prime tangles. *BMC bioinformatics*, 23(6):1–18.
- Francis, A., Marchei, D. and Steel, M. (2024). Phylogenetic network classes through the lens of expanding covers. *Journal of Mathematical Biology*, 88(5), 58.
- Marchei D., Merelli E. and Francis A. (2024). Factorizing the Brauer Monoid in polynomial time. (submitted to *Journal of Symbolic Computation*)

Introduction

Introducing new tools can open up new ways of looking at a problem. In this thesis we introduce two: one for studying RNA secondary structures, and one for encoding phylogenetic networks.

About 30 years ago, [Kauffman and Magarshak, 1995] proposed a mapping between RNA secondary structures and elements in the Brauer monoid $\mathcal{B}_N$, called tangles. We decided to further explore this idea by looking at how these tangles can be factorized. To do that, a (possibly fast) factorization algorithm is necessary. The literature has some solutions, but they do not cover all possible cases, or their time complexity is impractical for large inputs because they require a more than factorial time complexity, in fact

$$|\mathcal{B}_N| = (2N-1)(2N-3)\cdots 3 \cdot 1$$

We first devised a heuristic-based algorithm, which was fast for finding non-optimal solutions, but slow at refining them. This, however, was still a good approximation for doing research in the field, and prove how certain properties of the RNA secondary structures can be recognized by the structure of their corresponding tangle factorization [Marchei and Merelli, 2022]. We later found a better algorithm that runs in $\mathcal{O}(N^4)$ which, to the best of our knowledge, is the first polynomial time solution in N (and not in the size of $\mathcal{B}_N$) for this problem. The key insight was the idea of mapping every element in the Brauer monoid to one of its subsemigroups (the symmetric group). This lead to the discovery of a length function (a function that determines the minimal length of each tangle's factorization), which we prove it can be used for constructing a polynomial time factorization algorithm.

The second tool we describe is a new encoding for phylogenetic networks. Usually, the definitions for specific classes of networks are suitably crafted for their specific domain of interest, typically by imposing constraints on their graph structure. However, as described in [Francis and Steel, 2023], there exists a large class of networks, called *labellable networks*, that includes many of the networks studied in the literature, and it turns out that they can be described using set covers. Using this correspondence as starting point, we found out that several famous subclasses can be expressed using this language as well, along with a new class (*spinal networks*) which we hope could be of interest. This result also equips the researcher with a new dictionary, helping them translate between the language of graphs and the language of covers, providing furthermore a new way of classifying networks by adjusting, as needed, the properties these covers have to satisfy.

In Chapter 1 we give the mathematical prerequisites for reading this thesis; in Chapter 2 we present the two methods for factorizing any tangle in the Brauer monoid, which find application in Chapter 3, where we study RNA secondary structures using their corresponding factorization in the Brauer monoid. In Chapter 4 we then introduce the new language of covers for phylogenetic networks.

Chapter 1

Preliminaries

In this chapter we are going to present the basic mathematical framework we will use throughout this thesis. We will firstly give a primer on semigroups and monoids, with their applications in Computer Science and Biology, and then proceed to introduce phylogenetic networks and how they can be described using the set-theoretic concept of covers.

1.1 Semigroups and Monoids

In this section, we will introduce some basic algebraic concept we will use in Chapter 2 and Chapter 3. We will start with the definition for a semigroup.

Definition 1. A semigroup S is a non-empty set with a composition operator $\circ : S \times S \to S$ such that for all $x, y, z \in S$, then $(x \circ y) \circ z = x \circ (y \circ z)$, also known as the associative property.

In this thesis, we will focus on a particular type of semigroup: the monoid.

Definition 2. A monoid is a semigroup S with an identity element e. In other words, for all $x \in S$, $x \circ e = e \circ x = x$.[1]

[1] Furthermore, if we assume that every element x has an inverse x^{-1} such that $x \circ x^{-1} = x^{-1} \circ x = e$, we obtain the famous algebraic structure called *group*.

Classical examples of semigroups which are not monoids are:

- the set of natural numbers $\mathbb{N}$ under the min operator;
- the class of sets under intersection;
- the set of even integers $\mathbb{Z}$ under multiplication.

Examples of monoids are:

- the natural numbers $\mathbb{N}$ under addition and 0 as identity;
- the set of finite strings over a set of symbols Σ under string concatenation and the empty string as identity;
- for any finite set A of N elements, the set of functions $f : A \to A$ under the composition operator and the function $id_A(a) = a$ as identity. This is called the full transformation monoid $\mathcal{T}_N$.

It is often the case that these structures are defined in terms of *generators* and *axioms* over them.

Definition 3. Given a semigroup (monoid) S, then a set $G \subseteq S$ is a generating set for S if for all $x \in S$, then $x = g_1 \circ g_2 \circ \cdots \circ g_n$ for some $g_1, g_2, \cdots, g_n \in G$.

For example, the monoid of natural numbers greater or equal to two under multiplication can be generated by the set of prime numbers, this fact is famously known as the Fundamental Theorem of Arithmetic.

Definition 4. Given a semigroup (monoid) S and a generating set G for S, the axiom set A for G is a set of equations defining equalities for words on G. Equations in A are in the form

$$g_1 \circ \cdots \circ g_n = g'_1 \circ \cdots \circ g'_k$$

for some integers n and k and some $g_1, \cdots g_n, g'_1, \cdots g'_k \in G$.

To continue the above example:

$$(p \times q) \times r = p \times (q \times r) \tag{1.1a}$$

$$p \times q = q \times p \tag{1.1b}$$

where p, q, r are prime numbers. Equation 1.1a refers to the axiom of associativity, required by the semigroup structure, while Equation 1.1b describes the axiom of commutativity, which is not required by the semigroup structure, but it is necessary to fully describe the multiplication operator. Since 1 is not considered a prime number, it is not part of the generating set and therefore there are no axioms for it. Usually, only the axioms specific to the operator under study are listed, while the axiom related to the semigroup (or monoid) structure are omitted if it is clear from the context. We will do the same throughout this work.

The last important definition to introduce regards *subsemigroups* and *submonoids*.

Definition 5. Given a semigroup (monoid) S, a subsemigroup (submonoid) Z is a semigroup (monoid) such that $Z \subseteq S$.

Some examples are:

- S is the monoid of natural numbers under addition and Z is the submonoid of even natural numbers under addition;
- if $\Sigma = \{a, b\}$ then Σ^* is the monoid of all finite strings under concatenation with a and b as symbols and the empty string as identity, then $Z = \{a\}^+$ is the subsemigroup of strings on just the element a under concatenation (the empty string is not included);
- the set of bijective functions[2] in the full transformation monoid $\mathcal{T}_N$ forms a submonoid (in particular, it forms a subgroup called the symmetric group S_N).

We will now illustrate some applications of semigroups and monoids in Computer Science and Biology.

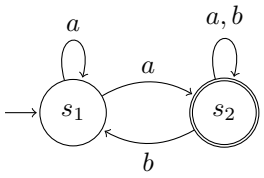
1.1.1 Applications in Computer Science

Semigroups and monoids have some interesting intersections with Computer Science, as for instance in [Pin, 1995], which extends Kleene's theorem by

[2]A function f is bijective if there exists a function g such that $f \circ g = g \circ f = id_A$ (assuming $f, g : A \to A$). In other words, $f = g^{-1}$.

saying that a language $L \subset A^+$ is regular if and only if it is "recognized" by a finite semigroup S, meaning that there exists a function $\varphi : A^+ \to S$ if $\varphi(x \cdot y) = \varphi(x) \circ \varphi(y)$, where x and y are strings, $\cdot$ is the concatenation operator and $\circ$ is the operator for S, and if $x \in L$ and $\varphi(x) = \varphi(y)$, it implies that $y \in L$.

In their Example 4.1, the finite state automata shown on the side is recognized by the semigroup $S = \{\left(\begin{smallmatrix}0&0\\1&1\end{smallmatrix}\right), \left(\begin{smallmatrix}1&1\\0&1\end{smallmatrix}\right), \left(\begin{smallmatrix}1&1\\1&1\end{smallmatrix}\right)\}$ under matrix multiplication because $\varphi(a) = \left(\begin{smallmatrix}1&1\\0&1\end{smallmatrix}\right)$, $\varphi(b) = \left(\begin{smallmatrix}0&0\\1&1\end{smallmatrix}\right)$, and $\varphi(ab) = \left(\begin{smallmatrix}1&1\\1&1\end{smallmatrix}\right)$, where 0 and 1 are to be interpreted as boolean values.



Furthermore, the Krohn–Rhodes theorem is closely related to finite automata, stating that every semigroup can be decomposed as a wreath product of "prime" groups and "unit" semigroups [Krohn and Rhodes, 1965]. This is equivalent to say that every finite automaton is the cascade product of permutation-reset machines [Ginzburg and Yoeli, 1965].

In [Meseguer and Montanari, 1990] the authors describe how a Petri Net can be seen as two monoids:

- the monoid of places: if S is the set of places, for example $S = \{a, b, c\}$, then $S^{\oplus}$ is the monoid containing elements in the form $xa \oplus yb \oplus zc$ where $x, y, z \in \mathbb{N}$ and $xa \oplus yb \oplus zc \oplus x'a \oplus y'b \oplus z'c = (x+x')a \oplus (y+y')b \oplus (z+z')c$. The operator $\oplus$ can also be extended to parallel firing of transitions;
- the monoid of transition composition, in which firing two transitions t_1 and t_2 sequentially corresponds to the firing of another transition t defined as the composition of the two.

There are also some results in the context of Complexity Theory. For example, Corollary 10 of [Beaudry, 1988] proves that deciding if a function $f : A \to A$, for some finite set A, can be expressed as composition of generators of a commutative transformation semigroup is NP-complete. There are also some undecidable problems regarding semigroups. Among others, [Markov, 1951] states that there is no general algorithm for deciding if a set of axioms describes a semigroup. We refer to [Nyberg-Brodda, 2022] for an English translation.

1.1.2 Applications in Biology

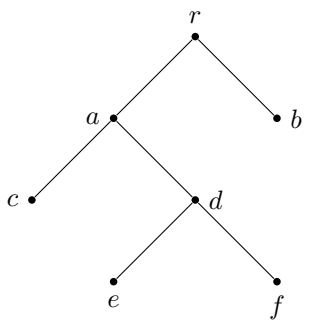


Figure 1.1 Example of a phylogenetic tree. This induces a semigroup whose elements are the nodes and the operator is the least common ancestor. For example $c \circ f = a$, $d \circ b = r$.

Semigroups and monoids have also been used in the context of biology. For example, every phylogenetic tree (also known as binary tree by computer scientists) induces a semigroup in which the elements are its nodes and the operator is the *least common ancestor*, also known as the *join operator* on semilattices[3] (see the figure on the left).

In [Quadrini et al., 2019a], the authors introduced a grammar that uniquely identifies surfaces associated to RNA secondary structures. These surfaces are modelled as "fatgraphs", which are closed under the associative operators of concatenation and nesting, which they define in the paper, thus forming a semigroup.

In [Egri-Nagy and Nehaniv, 2008], the authors use the Krohn-Rhodes theorem to decompose the transformation semigroup associated to a biological network (in their example: the E.coli's *lactose operon*). They illustrate how this decomposition provides an easy description of the metabolic cycle.

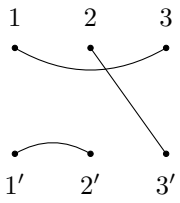
More akin to this thesis, the Brauer monoid (which we will introduce in the following section) has been used for proposing a way of combining binary phylogenetic trees in a genetic algorithm [Diaconis and Holmes, 1998] and then extended to non-binary phylogenetic trees in [Francis and Jarvis, 2022]. The Brauer monoid has also been used to model RNA secondary structures, in which a mapping between the two was first proposed in [Kauffman and Magarshak, 1995] and further analyzed in [Marchei and Merelli, 2022], whose results will be discussed in Chapter 3.

A particular submonoid of the Brauer monoid, the Jones monoid, has been utilized in [Garrett et al., 2019, 2021] as a model for DNA origami [Rothemund, 2006]. In particular, in [Garrett et al., 2019], the authors describe the "Origami

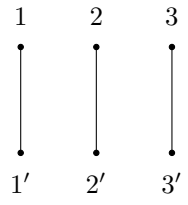
[3]A semilattice is a semigroup satisfying the commutativity law ($x \circ y = y \circ x$) and the idempotency law ($x \circ x = x$). If we assume $\circ$ to be the *least common ancestor* operator, then it can be easily seen how these laws are satisfied in a phylogenetic tree.

monoid", in which elements are depicted as two oriented and overlapping Jones monoids, modelling the *scaffold* and *staples* of the DNA. These results are then extended in [Garrett et al., 2021] by introducing "virtual" scaffolds and staples, which give rise to more interesting monoidal structures depending on the chosen generators.

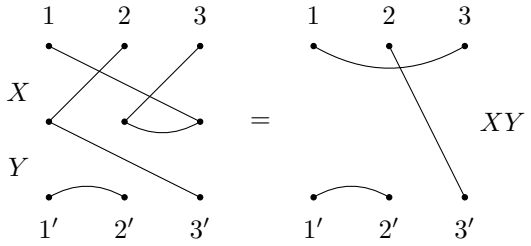
1.1.3 The Brauer monoid



Given $N \geq 0$, arrange $2N$ nodes in two rows of N nodes each. Nodes placed in the upper row are labelled with $[N] = \{1, 2, \cdots, N\}$ while nodes in the bottom row are labelled with $[N'] = \{1', 2', \cdots, N'\}$. A tangle is a set of N edges connecting any two distinct nodes in $[N] \cup [N']$ such that each node is in exactly one edge. For each N, there are exactly $(2N-1)!!$ tangles[4]. We will represent edges as $e = (x, y)$ in a canonical form in which $x < y$ if both x and y are nodes in the same row, while in the case that e connects nodes from in different rows, $x \in [N]$ and $y \in [N']$. For example, the tangle on the left is notated as $(1,3)(2,3')(1',2')$.



Given two tangles X and Y we define their composition $X \circ Y$ by stacking X on top of Y (matching the bottom row of X with the top row of Y) and then tracing the path of each edge (we will ignore internal loops in our setup). The set of all tangles on $2N$ nodes under composition is called the Brauer monoid $\mathcal{B}_N$ [Brauer, 1937] and the identity tangle, shown above, is $I_N = (1,1')(2,2') \cdots (i,i') \cdots (N,N')$. As it is often custom, we will write XY instead of $X \circ Y$ to indicate the composition of two tangles X and Y. Here is an example:

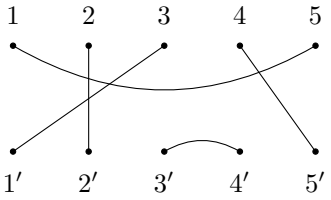


[4] $(2N-1)!!$ is the "odd double factorial", defined as $(2N-1)!! = (2N-1) \cdot (2N-3) \cdots 3 \cdot 1$.

For our purposes, it will be useful to classify edges by where they are connected. Tangles have two types of edges [Dolinka and East, 2018]:

- *hooks* are edges in which both nodes are on the top or on the bottom row. The former ones are called *upper hooks* and the latter *lower hooks*;
- *transversals* are edges in which one node is on the top row while the other one is on the bottom. We further classify transversal edges as:
  - *positive transversal* are in the form $x > y'$
  - *zero transversal* are in the form $x = y'$
  - *negative transversal* are in the form $x < y'$

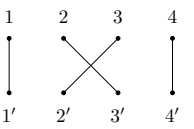
  where x is the upper node and y' is the lower node.



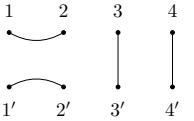
Given a tangle X, we say that two edges *cross* if they intersect each other in the diagrammatic representation of X (assuming the edges are drawn in a way that minimizes the number of crossings). The *size* of an edge $e = (x, y)$ is defined as $|e| = |x - y|$ (x and y are arbitrary nodes). For example, the tangle on the side has one upper hook $(1, 5)$ with size $|1 - 5| = 4$ and one lower hook $(3', 4')$ with size $|3 - 4| = 1$. The remaining edges are $(2, 2')$ (zero transversal), $(3, 1')$ (positive transversal of size two) and $(4, 5')$ (negative transversal of size one).

The Brauer monoid can also be defined as the set of tangles generated by the composition of *prime* tangles, which we call T-primes and U-primes, defined as:

- $T_i = (1, 1') \cdots (i, i' + 1)(i + 1, i') \cdots (N, N')$; $T_2 =$



- $U_i = (1, 1') \cdots (i, i + 1)(i', i' + 1) \cdots (N, N')$. $U_1 =$



where i is called the index of that prime tangle.

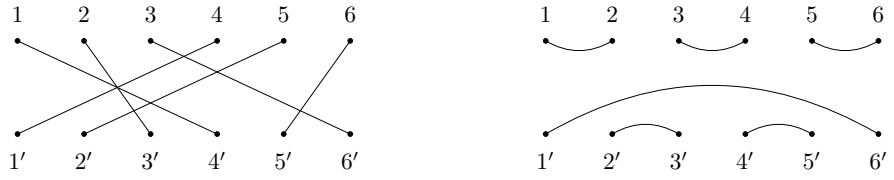
delete					braid			
1.	T_iT_i	$=$	I_N		11.	$T_iT_jT_i$	$=$	$T_jT_iT_j$ if $\lvert i-j\rvert = 1$
2.	U_iU_i	$=$	U_i		12.	$T_iU_jT_i$	$=$	$T_jU_iT_j$ if $\lvert i-j\rvert = 1$
3.	T_iU_i	$=$	U_i					
4.	U_iT_i	$=$	U_i					
5.	$U_iU_jU_i$	$=$	U_i	if $\lvert i-j\rvert = 1$				
6.	$U_iT_jU_i$	$=$	U_i	if $\lvert i-j\rvert = 1$	swap			
7.	$T_iU_jU_i$	$=$	T_jU_i	if $\lvert i-j\rvert = 1$	13.	T_iT_j	$=$	T_jT_i if $\lvert i-j\rvert > 1$
8.	$U_iU_jT_i$	$=$	U_iT_j	if $\lvert i-j\rvert = 1$	14.	T_iU_j	$=$	U_jT_i if $\lvert i-j\rvert > 1$
9.	$U_iT_jT_i$	$=$	U_iU_j	if $\lvert i-j\rvert = 1$	15.	U_iT_j	$=$	T_jU_i if $\lvert i-j\rvert > 1$
10.	$T_iT_jU_i$	$=$	U_jU_i	if $\lvert i-j\rvert = 1$	16.	U_iU_j	$=$	U_jU_i if $\lvert i-j\rvert > 1$

Table 1.1 Axioms for the Brauer monoid. Axioms from 1 to 10 are called *delete*, 11 and 12 are *braid* and 13-16 are *swap*. We will call them "rules" if we are rewriting from left to right of the equality symbol.

These primes also satisfy the axioms in Table 1.1 [Kudryavtseva and Mazorchuk, 2006]. We classify these axioms into three types:

- *delete axioms*: because one side has fewer prime tangles than the other;
- *braid axioms*: because they resemble the braid relation in the symmetric group;
- *swap axioms*: because they allow to commute primes without changing the resulting tangle.

Throughout this thesis, we are going to use the words "axioms" when we want to refer to the equality between two words, and "rules" when we want to rewrite the word from the left-hand side to the right-hand side. Lastly, the Brauer monoid has two submonoids: the set of all tangles only generated by T-primes, called the symmetric group S_N, whose tangles have only transversal edges; and the Jones monoid $\mathcal{J}_N$ [Jones, 1983; Temperley and Lieb, 1971], corresponding to the set of tangles generated by U-primes. All tangles in $\mathcal{J}_N$ are without crossings.



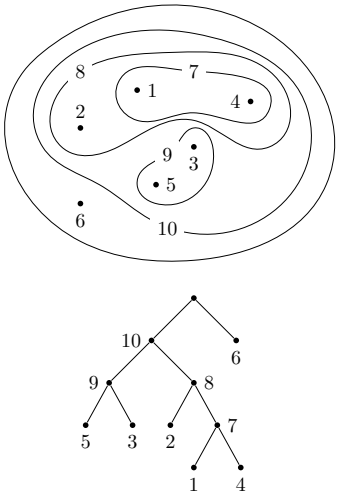
(a) A tangle in S_6. (b) A tangle in $\mathcal{J}_6$.

Figure 1.2 Example of tangles in S_6 and $\mathcal{J}_6$.

1.2 Phylogenetic networks

1.2.1 From Tangles to Phylogenetic networks

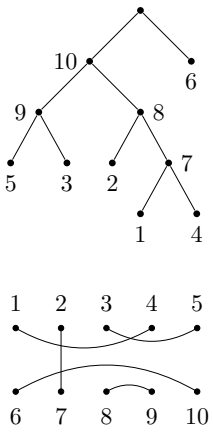
Recall that the size of $\mathcal{B}_N$ is $|\mathcal{B}_N| = (2N-1)!!$. The double odd factorial has some close connections with the branch of combinatorics, and with perfect matchings in particular. Given the set $A = [2N]$, a perfect matching for A is a partition[5] $\mathcal{C}$ such that every subset of $\mathcal{C}$ is of size two. For example, a perfect matching for $A = [6]$ is $\{\{1,5\},\{2,6\},\{4,3\}\}$. It can be proven that the number of perfect matchings for $A = [2N]$ is exactly $(2N-1)!!$. Therefore, the Brauer monoid can be also thought as the set of all perfect matchings of $2N$ elements.



Let us now move our attention to a different but related problem. In [Schröder, 1870], the author describes and solves the following combinatorial problem: given N points, pair two of them and treat this new pair as a new point. Now there are $N-1$ unpaired points. Repeat this process until there are only two points left. How many ways there are to carry on this process? This problem is known as the Schröder's Third Problem, and its solution is $(2N-3)!!$. The same problem can be thought as counting the number of binary trees with N labelled leaves: given N nodes without a parent, pick two of them and set them as the children of a new node. There are now $N-1$ nodes without a parent. Repeat this process until there is only one node remaining (the root). It can be easily seen that these two problems are equivalent, and therefore the set of binary trees with N labelled leaves has cardinality $(2N-3)!!$ [Erdős and Székely, 1989].

It is now clear that the set of perfect matchings of $2N$ elements has the same size of the set of binary trees with $N+1$ labelled leaves. In fact, given a perfect matching $\mathcal{C}$ for a set A, then if $\{x,y\} \in \mathcal{C}$ it implies that there exists a binary tree in which x and y are siblings, i.e. they share the same parent.

[5] A *partition* of a finite set A is a set of non-empty, pairwise disjoint subsets of A whose union is A.



For example, in the figure on the left, the tangle $X \in \mathcal{B}_5$ is constructed such that an edge $(x, y) \in X$ if and only if x and y are siblings in the binary tree[6].

In the context of evolutionary biology, a binary tree is called a phylogenetic tree, and they are used to modeling ancestral relations between species. It is of particular interest the problem of finding the "optimal" tree for a given set of species, and different metrics have been proposed to define distance functions between trees. In particular, this bijection has been used by [Diaconis and Holmes, 1998] to explore possible new distance functions on phylogenetic tree by looking at distances between perfect matchings.

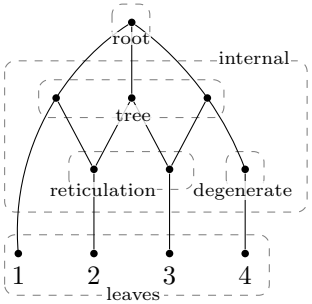
In this thesis, we would like to explore a generalization of phylogenetic trees, called *phylogenetic networks*. We will show how they can be described by a generalization of perfect matchings, called *covers*, and how we can move between different known classes of phylogenetic networks by just assuming different properties their cover must satisfy.

Phylogenetic networks can provide more complete representations of evolutionary relationships among species than possible with a simple phylogenetic tree [Bapteste et al., 2013; Huson et al., 2010]. Although a single tree can accurately show ancestral speciation events (splitting of lineages), it cannot display reticulate evolution (where the flow of genomic information follows the merging of ancestral lineages). Well-known reticulate processes in biology include hybridization, horizontal gene transfer, recombination, and endosymbiosis, in both the recent and distant past. By contrast, rooted phylogenetic networks can explicitly and simultaneously display both speciation and reticulate evolution. As a result, the mathematical and algorithmic investigation of phylogenetic networks has become a highly active field over the last $\sim$15 years, and numerous classes of networks have been defined and studied [Kong et al., 2022]. In Chapter 4, we will put aside the Brauer monoid and show how

[6]The labelling of the tangle's nodes is arbitrary, but depending on how they are labelled, multiple tangles could correspond to the tree. Therefore, when applying this mapping, there should be an agreement on how the tangles' nodes are labelled. In this particular example, the top row is labelled with integers from 1 to 5 and the bottom row with integers from 6 to 10.

a recently introduced correspondence for a large class of phylogenetic networks (the *labellable* networks [Francis and Steel, 2023]) can be used to characterize a number of widely used other classes of network. Classes of network have been introduced for a variety of reasons, but usually in order to capture some feature that seems biologically important, or because they are mathematically convenient. Their definitions typically involve constraints on their structures as graphs. For instance, tree-child networks are those for which no vertex has only reticulations as its children, whereas tree-based networks are those that can be constructed from a base tree by adding additional edges between the tree edges. The class of labellable networks contains many commonly studied classes, and they have been shown to correspond to a set of covers of finite sets that satisfy a property called "expanding". We explore features of covers arising from networks, and characterize many of the familiar classes in terms of properties of their associated covers.

1.2.2 Phylogenetic networks

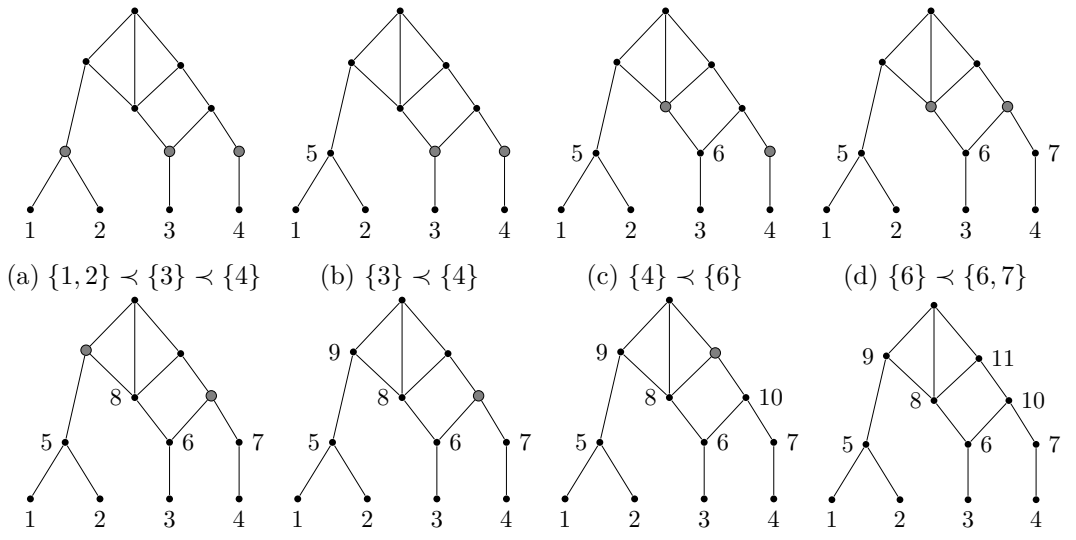


A *phylogenetic network* on n leaves is a directed acyclic graph with a single vertex of in-degree zero, called the root, and n vertices of in-degree 1 and out-degree zero (the leaves), labelled by $[n]$. Nodes which are non-root and non-leaf are called *internal*. Vertices that have in-degree and out-degree both equal to 1, or both strictly greater than 1 are called *degenerate*. If the network has any degenerate vertices, it is said to be a *degenerate* network; otherwise, it is *non-degenerate*. We will draw all networks with the root at the top and leaves at the bottom, and all edges are directed with orientation down the page. If every vertex has in-degree and out-degree at most 2, then the network is said to be *binary*. If the network is non-degenerate and binary, then all vertices other than the leaves and root have total degree 3. Vertices in a network that have in-degree 1 are called *tree vertices*, and those with in-degree greater than 1 are called *reticulate vertices*, or *reticulations*. We will typically use k to denote the number of reticulations in a network, and m to denote the number of non-root vertices in total. In the figure above, $n = 4$, $k = 2$ and $m = 10$.

A *labellable* phylogenetic network is one whose vertices can be deterministically labelled according to Algorithm 1, presented [Francis and Steel, 2023], and that generalises one for trees due to [Erdős and Székely, 1989], for an example see Figure 1.3. Such networks are characterized topologically by the property that the map from non-leaf vertices to their sets of children is one-to-one [Francis and Steel, 2023, Thm.3.3]. In Algorithm 1, the relation $\prec$ refers to the lexicographic order on sets, given by $A \prec B$ if $A \subset B$ or $\min(A \setminus B) < \min(B \setminus A)$.

Algorithm 1 Labelling algorithm for phylogenetic networks.

procedure LABELNETWORK(N)
  $m \leftarrow 0$
  $I \leftarrow$ set of unlabelled internal vertices in N plus the root
  while $|I| > 1$ **do**
    $A \leftarrow$ set of nodes in I with all labelled children
    for $v \in A$ **do**
      if the set of children of v is minimal under $\prec$ **then**
        label v as $n + m + 1$
        $m \leftarrow m + 1$
        $I \leftarrow I \setminus \{v\}$



(a) $\{1,2\} \prec \{3\} \prec \{4\}$ (b) $\{3\} \prec \{4\}$ (c) $\{4\} \prec \{6\}$ (d) $\{6\} \prec \{6,7\}$

(e) $\{5,8\} \prec \{6,7\}$ (f) $\{6,7\}$ (g) $\{8,10\}$ (h) No internal vertex left to be labelled.

Figure 1.3 Algorithm 1 selects all nodes that have all their children labelled (marked with big gray dots) and labels the node whose set of children is minimal under $\prec$.

1.2.3 Covers as a language for phylogenetic networks

Now take again the example in Figure 1.3. Notice that if, at each step, we keep track of the minimal set under $\prec$ and add the set of children of the root at the end, then we will obtain the following set of sets $\mathcal{C}$:

$$\mathcal{C} = \{\{1,2\},\{3\},\{4\},\{6\},\{5,8\},\{6,7\},\{8,10\},\{8,9,11\}\} \tag{1.2}$$

This is called a *cover*, and as we have mentioned, will be the subject of study of Chapter 4, as they have the capability of describing the whole network and can encode different topological properties in a set-theoretic manner.

Definition 6. A *cover* of a finite set A is a set of non-empty subsets of A whose union is A.

In a cover $\mathcal{C}$ for A, its sets may or may not overlap, the only requirement is that their union is exactly A. For visual aid, we will drop the curly braces and write all sets of $\mathcal{C}$ divided by a | as follows:

$$\mathcal{C} = 1,2 \mid 3 \mid 4 \mid 6 \mid 5,8 \mid 6,7 \mid 8,10 \mid 8,9,11$$

The cardinality $|\mathcal{C}|$ of a cover $\mathcal{C}$ is the number of sets it contains. We use $||\mathcal{C}||$ to denote the number of distinct elements in the sets in $\mathcal{C}$, that is, $||\mathcal{C}|| := |\bigcup_{C_i \in \mathcal{C}} C_i|$.

Recall the definition from [Francis and Steel, 2023]:

Definition 7. A cover $\mathcal{C}$ of $[m]$ is *expanding* if, for $n = m - |\mathcal{C}| + 1$, it satisfies:

1. No element of $[n]$ appears more than once, and
2. For $i = 1, \ldots, |\mathcal{C}|$, the cover contains at least i subsets of $[n + i - 1]$.

Theorem 1. *[Francis and Steel, 2023, Thm. 4.4] The class of labellable phylogenetic networks is in bijection with the collection of expanding covers of finite sets.*

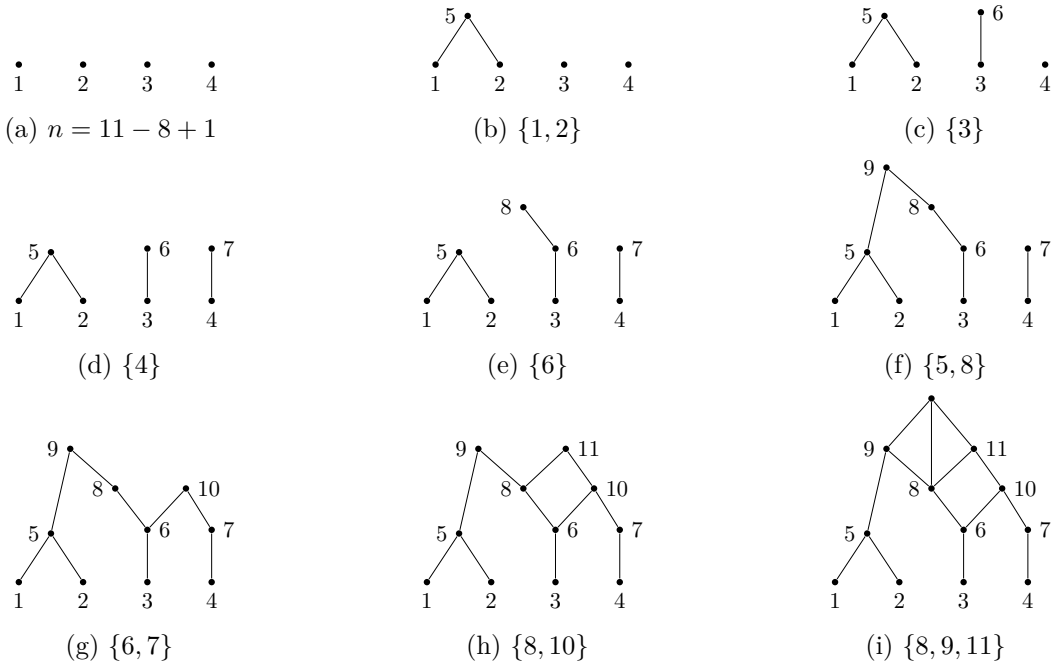
The map from a labellable phylogenetic network to its expanding cover takes each non-leaf vertex to the set of labels of its children. That is, sets

in the cover are sets of labels of sibling vertices sharing a parent. The map from an expanding cover $\mathcal{C}$ to a labellable network is a constructive map that first establishes the number of leaves in the network via the following formula [Francis and Steel, 2023, Lemma 4.1]:

$$n = ||\mathcal{C}|| - |\mathcal{C}| + 1.$$

The construction of the network then begins with n isolated leaf vertices, and adds parent vertices to sets of vertices present in the growing network, and lexicographically minimal of those in $\mathcal{C}$. The expanding conditions ensure that there is always such a set, and that the map is well-defined. The following figure illustrates the construction of the network in Figure 1.3 from the cover that we have already seen:

$$\mathcal{C} = \{\{1,2\},\{3\},\{4\},\{6\},\{5,8\},\{6,7\},\{8,10\},\{8,9,11\}\}$$



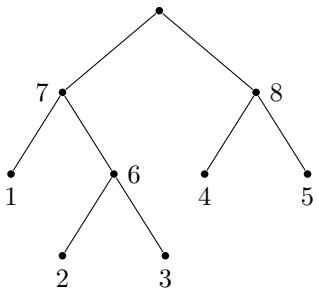
While the condition for a cover to be expanding may seem artificial, and it certainly restricts from the collection of all covers of a set, it can be seen as a natural extension of the notion of partitions. In particular, it turns out that all partitions are expanding covers.

Lemma 1. *Every set partition is an expanding cover.*

Proof. Let π be a partition of $[m]$ with $\ell = |\pi|$ blocks, and set $n = m-\ell+1$. Two conditions define an expanding cover. The first is that elements of $\{1, \ldots, n\}$ are not repeated in π, which is satisfied by virtue of π being a partition. The second is that for each $i = 1, \ldots, \ell$, π contains at least i subsets of $[n + i - 1]$, and we prove this by induction on i. First, consider the base case $i = 1$. We need to show that there is at least one set in π that is a subset of $[n]$. There are $\ell = m - n + 1$ pairwise disjoint subsets of $[m]$ in π, and there are $m - n$ integers in $[m]$ that are not in $[n]$. Therefore, there must be at least one set in π that does not contain an element of $\{n + 1, \ldots, m\}$ and is thus is a subset of $[n]$, as required.

Suppose that for $i = k$, π contains at least k subsets of $[n+k-1]$. We would like to show that π contains at least $k + 1$ subsets of $[n + (k + 1) - 1] = [n + k]$. The proof proceeds in the same manner as the case of $i = 1$.

First remove k subsets of $[n + k - 1]$ from π, so that π has $\ell - k$ sets remaining. We need to show at least one remaining set is entirely contained within $[n + k]$. There are $m - (n + k)$ integers in π that are *not* in $[n + k]$, and $\ell - k = (m - n + 1) - k = m - (n + k) + 1$ sets are available. Therefore, at least one must not contain any element outside $[n + k]$, as required. □



Since all set partitions are expanding covers, we can ask what sort of networks have partitions as their covers. A partition has a single occurrence of each integer, which means that each vertex of the network (each label) has a single set of siblings. In other words, the network has no reticulations, and thus is a tree. For example, the tree on the side has cover $2, 3 \mid 1, 6 \mid 4, 5 \mid 7, 8$, which is in fact a partition. This correspondence of trees with partitions allows trees with degenerate vertices (i.e., vertices with in-degree and out-degree 1). In this way, the correspondence for partitions is closer to the result of Erdős and Székely [Erdős and Székely, 1989] than the non-degenerate framework that has partitions in bijection with phylogenetic forests in [Francis and Jarvis, 2022].

The lexicographic order on sets that helps determine the labelling sequence is not always the ordering of sets used to label the internal vertices of the network; that sequence is given by the *labelling order*, which is defined as follows [Francis and Steel, 2023, Section 4]:

Definition 8. The *labelling order* for an expanding cover $\mathcal{C}$ is determined by the following process.

1. For $i = 1, \ldots, |\mathcal{C}|$,
   (a) Set C_i to be the minimal set in $(\mathcal{C}, \prec)$ contained in $[n + i - 1]$; and
   (b) Redefine $\mathcal{C} = \mathcal{C} \setminus \{C_i\}$.
2. Output the sequence $C_1, \ldots, C_{|\mathcal{C}|}$.

The cover given in Equation 1.2 is already in labelling order, to show that this is different from the plain lexicographic order here is that cover but lexicographically ordered:

$$1, 2 \mid 3 \mid 4 \mid 5, 8 \mid 6 \mid 6, 7 \mid 8, 9, 11 \mid 8, 10$$

We can write a cover in labelling order more explicitly by labelling every subset in position $1 \leq i < |\mathcal{C}|$ by $i + n$, whereas the last subset is labelled ρ for the *root*. In this way, the label for each subset corresponds to the label of its parent in the corresponding labellable network. In our running example (recall that $n = 4$):

$$\mathcal{C} = \{1, 2\}_5, \{3\}_6, \{4\}_7, \{6\}_8, \{5, 8\}_9, \{6, 7\}_{10}, \{8, 10\}_{11}, \{8, 9, 11\}_\rho$$

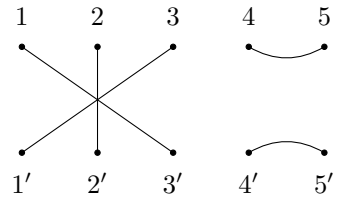
The labelling order is necessary to establish conditions on a cover that give non-degenerate networks, for instance, and we will use it later in Chapter 4 to describe *normal networks* (in Section 4.4) and *orchard networks* (Section 4.6).

Chapter 2

Factorizing the Brauer monoid

Finding a minimal factorization for a generic semigroup can be done by using the Froidure-Pin Algorithm, which is not feasible for semigroups of large sizes. On the other hand, if we restrict our attention to just a particular semigroup, we could leverage its structure to obtain a much faster algorithm. In particular, $\mathcal{O}(N^2)$ algorithms are known for factorizing the symmetric group S_N and the Jones monoid $\mathcal{J}_N$, but none for their superset the Brauer monoid $\mathcal{B}_N$. Hence, in this chapter we will focus on the problem of finding a minimal factorization for any tangle in $\mathcal{B}_N$. The work presented in this chapter is based on two papers: the content of Section 2.3 has been published in the paper [Marchei and Merelli, 2022], while the content of Section 2.4 is based on the paper "Factorizating the Brauer monoid in polynomial time" by Daniele Marchei, Emanuela Merelli and Andrew Francis, which was submitted to the Journal of Symbolic Computation.

2.1 The factorization problem for $\mathcal{B}_N$



We call a word $F \in \{T_i, U_i\}^*$ a *factorization* (the empty word coincides with the identity I_N). If this factorization is reduced, i.e. there is no other equivalent word of shorter length, then it is called *minimal*. For example, one non-minimal factorization for the tangle on the left is $T_1T_1T_2T_1U_4T_2U_4$, while $T_1T_2T_1U_4$ is a minimal one.

Define $\otimes : \mathcal{B}_N \times \mathcal{B}_M \to \mathcal{B}_{N+M}$ to be the tensor product such that $Z = X \otimes Y$ is the tangle in which Y is placed on the right of X. We call X and Y the components of Z [Ram, 1995]. If we assume F_X and F_Y to be minimal factorizations for X and Y, then we have that $F_Z = F_XF_Y$ is a minimal factorization for Z, therefore we can factorize each component separately. The tangle above, for example, is in $\mathcal{B}_5$ and is the tensor product of two tangles,  $\in \mathcal{B}_3$ and  $\in \mathcal{B}_2$, in fact a minimal factorization for it can be expressed as $(T_1T_2T_1)(U_4)$. For the rest of the thesis we will assume that every tangle has only one component.

There are algorithms in the literature for factorizing elements in an arbitrary semigroup, we will give a brief recap of them.

The Froidure-Pin Algorithm can find a minimal factorization for any element in a finite semigroup S by performing $|S| + |A| - |G| - 1$ operations, where A is the set of axioms of S, and G the set of generators [Froidure and Pin, 1997]. For the Brauer monoid the resulting time complexity lower-bound is therefore $\Omega(|\mathcal{B}_N|) = \Omega((2N-1)!!)$.

Algorithm 13 of "Computing with semigroups" [East et al., 2019] is capable of finding a factorization, but it is not guaranteed to be minimal. It computes two sets $\mathfrak{R}$ (the $\mathcal{R}$-classes of $\mathcal{B}_N$) and $(\mathcal{B}_N)\lambda$. The size of $\mathfrak{R}$ for $\mathcal{B}_N$ can be calculated by the following recurrence relation:

$$\begin{aligned} a(0) &= 1 \\ a(1) &= 1 \\ a(N) &= a(N-1) + (N-1)a(N-2) \end{aligned}$$

which is the number of ways to partition a set of size N into subsets of size one or two [Dolinka et al., 2017]. This recurrent relation is clearly bounded below by $N!$. The authors also say that for regular semigroups (as in the case for $\mathcal{B}_N$), $|\mathfrak{R}| = |(\mathcal{B}_N)\lambda|$, and that the calculation for $\mathfrak{R}$ is redundant. This does not reduce the time complexity because $(\mathcal{B}_N)\lambda$ still needs to be computed.

Lastly, two submonoids of $\mathcal{B}_N$ can be factorized in quadratic time. The symmetric group can be factorized by using the BubbleSort algorithm, and the Jones monoid can be factorized by using the algorithm proposed by Ernst et al. [Ernst et al., 2016]. We will discuss these two algorithms in the following section.

2.2 Factorization of S_N and $\mathcal{J}_N$

2.2.1 Factorization of S_N

Every element in S_N can be uniquely described as a permutation of the string $1, 2, \cdots, N$. A simple isomorphism from a string permutation $s_1, s_2, \cdots, s_N$ and a tangle in S_N is to connect the upper node i to the lower node s'_i. The identity permutation string is therefore the one in which every element is in ascending order. In this context, factorizing a tangle in S_N is the same as finding the shortest sequence of adjacent transpositions (i.e. $s_i, s_{i+1} \to s_{i+1}, s_i$) such that the original string is reduced to the identity string. In other words, we have to sort the string. This is a well-known problem in Computer Science and there are numerous fast algorithms for solving it. However, due to the constraint of using only adjacent transpositions, we are bound to a quadratic time complexity since the longest minimal factorization in S_N has size $\frac{N(N-1)}{2}$. One such algorithm is the BubbleSort [Friend, 1956], which specifically sorts strings using adjacent transpositions, yielding the shortest possible sequence. It works by scanning the sequence from left to right and if two adjacent elements are in reverse order, swaps them. Then the procedure repeats until every element is in order. Algorithm 2 follows the same procedure but stores the prime tangle T_i when the elements in position i and $i + 1$ are swapped. The output is a minimal factorization for the input tangle $X \in S_N$.

Algorithm 2 Factorization algorithm for any tangle in S_N.

Require: $X \in S_N$

function FACTORIZE $S_N(X)$

$s \leftarrow$ string permutation for X

$F \leftarrow$ empty list

for $j \in [N \dots 1]$ **do**

for $i \in [1 \dots j-1]$ **do**

if $s_i > s_{i+1}$ **then**

Swap s_i and s_{i+1}

Append T_i to F

return F

2.2.2 Factorization of $\mathcal{J}_N$

The Jones monoid $\mathcal{J}_N$ [Jones, 1983; Temperley and Lieb, 1971] is a submonoid of $\mathcal{B}_N$ in which only U-primes are taken as generators. Informally speaking, $\mathcal{J}_N$ contains all tangles with no crossings. A $\Omega(N^2)$ factorization algorithm was first proposed by Ernst et al. [Ernst et al., 2016] and in this section we will give a surface-level explanation of how it works (see Figure 2.1).

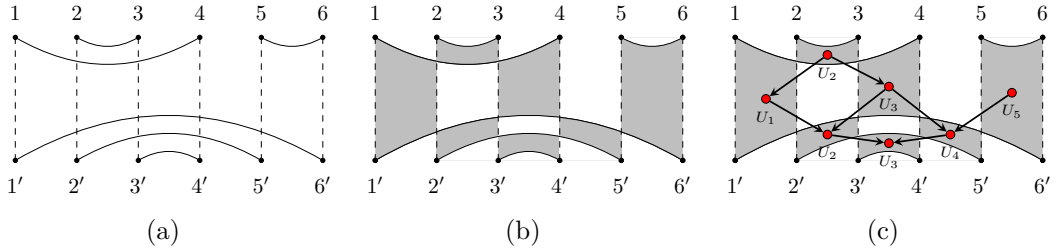


Figure 2.1 (a) A tangle in $\mathcal{J}_6$. Drawing six imaginary edges (i, i') we see that the tangle is now divided into five columns and each column is divided into regions, delimited by edges. (b) Select the regions with odd depth. (c) If two regions are diagonally adjacent, connect the top one with the bottom one, thus obtaining a Directed Acyclic Graph. Label each node as U_i, where i is the index of the columns it is in. If we read this DAG from top to bottom and from left to right we obtain the minimal factorization $U_2U_5U_1U_3U_2U_4U_3$.

Given a tangle X in $\mathcal{J}_N$, divide it into $N-1$ columns by adding a vertical line for each pair (i, i'). Each column now will be further divided into different regions delimited by the edges it contains. Each region has a depth value indicated by how many other regions there are above it. We will call regions with

even depth 0-regions and regions with odd depth 1-regions. Two regions, R_1 and R_2, in the same column are vertically adjacent if $|depth(R_1) - depth(R_2)| = 1$. Given two regions R_1 and R_2 in adjacent columns and two points $p_1 \in R_1$ and $p_2 \in R_2$, if we can draw a straight line between them without crossing an edge in X, then we say that R_1 and R_2 are horizontally adjacent. If there is a region R' vertically adjacent to R_1 and horizontally adjacent to R_2, then R_1 and R_2 are diagonally adjacent. We will write $R_1 \to R_2$ in the special case where R' is below R_1.

Let $\mathfrak{R}$ be the set of all 1-regions. From here we construct a Directed Acyclic Graph (DAG) G such that every region $R \in \mathfrak{R}$ is a vertex of G and given two regions R_1 and R_2, if $R_1 \to R_2$, then (R_1, R_2) is an edge of G. If a vertex does not have incoming edges, then we call it a root of G. Finally, to obtain the factorization for X we traverse each vertex from *top to bottom* and from *left to right*, i.e. we list all roots $r_i, r_j, \cdots, r_k$ of G, store $U_iU_j \cdots U_k$ and delete the roots, now other nodes will not have incoming edges, list them as the new roots $r'_i, r'_j, \cdots, r'_k$, and so on until G is empty (Algorithm 3).

Algorithm 3 Factorizes tangle in $\mathcal{J}_N$

```
procedure FACTORIZE J_N(X)
    Divide X in columns
    R ← regions of X
    ℜ ← 1-regions of R
    G ← DAG from ℜ s.t. if R_1 → R_2, then (R_1, R_2) ∈ G
    F ← empty list
    while G ≠ ∅ do
        S ← roots of G
        for s ∈ S do
            i ← column where s lies in
            Append U_i to F
        Remove nodes in S from G
    return F
```

The time complexity of this algorithm is bounded above by $\mathcal{O}(N^2)$ because the number of 1-regions is the same as the number of nodes in G, which is equal to the length of the factorization for the input tangle which, as we will prove in Corollary 3, we know is quadratic.

2.3 Factorization of $\mathcal{B}_N$ using heuristics

As for S_N and $\mathcal{J}_N$, factorizable in polynomial time because they have properties that can be exploited (all transversal edges and diagonal adjacent regions respectively), we wanted to find more types of tangles that have other properties that could be used to devise a factorization algorithm specific for them. Only for the purpose of this section, we will introduce two more subsets of $\mathcal{B}_N$:

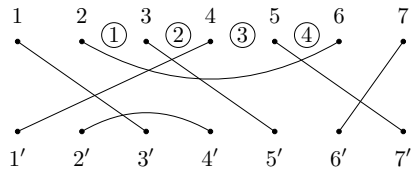
- $\mathcal{H}$-tangles: a tangle $X \notin \mathcal{J}_N$ containing at least one upper hook h of size $|h| > 1$;
- $\mathcal{U}$-tangles: a tangle $X \notin \mathcal{J}_N$ in the form $X = U_i X'$. In other words, a tangle with at least one crossing and an upper hook of size one.

2.3.1 Factorizing $\mathcal{H}$-tangles

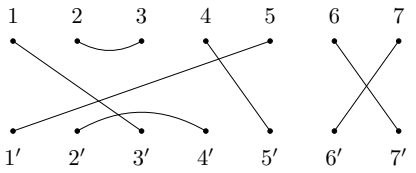
We will extract factors from a $\mathcal{H}$-tangle X by transforming it into a $\mathcal{U}$-tangle. The idea is to take one of the upper hooks h with size $|h| > 1$ and *shrink* it until it becomes of size one. To do this we pre-compose X with T-primes until this condition is met. During the shrinking process, other edges will inevitably change size. In order to decide *where* we should shrink h, we use a heuristic that chooses a location where the size of the other edges increases the least. We apply this heuristic to the smallest lower hook of X, in this way there will be no smaller lower hook inside it.

Heuristic 1. *Given a $\mathcal{H}$-tangle X, let $h = (i, i+k)$ be the smallest upper hook of X of size $k > 1$. Let j be the index of the shrinkage location where the size of the other edges increases the least. Shrink h into location j by pre-composing X with $L = T_{i+j-1}T_{i+j-2}\cdots T_i$ and $R = T_{i+k-1}T_{i+k-2}\cdots T_{i+j}$. This procedure yields a $\mathcal{U}$-tangle X' such that $X = LRX'$.*

For example take the following $\mathcal{H}$-tangle on the left. It has a single upper hook h of size four and therefore it has four possible shrinkage locations, indicated with circled numbers. The table on the right lists, for each edge inside h, how much its size would change if we shrunk h in a particular location.



	$(3,5')$	$(4,1')$	$(5,7')$	Sum	L	R
①	-1	$+1$	-1	-1		$T_3T_4T_5$
②	$+1$	$+1$	-1	$+1$	T_2	T_4T_5
③	$+1$	-1	-1	-1	T_3T_2	T_5
④	$+1$	-1	$+1$	$+1$	$T_4T_3T_2$	

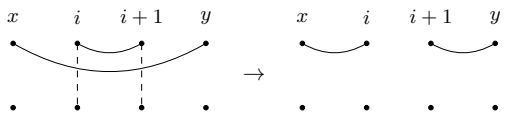


Since ① and ③ have the smallest sum, we pick the first one and pre-compose X with $T_3T_4T_5$, which results in the tangle on the left. This new tangle is the tensor product of two tangles: one in $\mathcal{B}_5$ and one in $\mathcal{B}_2$, therefore they can be factorized independently. This heuristic is not guaranteed to yield a minimal factorization, but it can be computed in linear time.

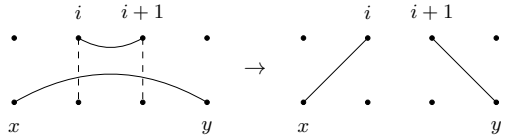
2.3.2 Factorizing $\mathcal{U}$-tangles

Recall that a $\mathcal{U}$-tangle is a tangle in the form $X = X'U_i$, we would like to find X' by removing U_i from X. To do this, we define the *merge operation* between a hook h of size one with another edge e in the tangle.

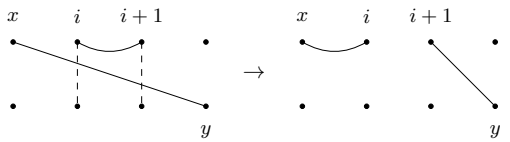
Definition 9 (Merge operation)**.** Given a tangle X with an upper hook of size one $h = (i, i+1)$ (resp. lower hook $h = (i', i'+1)$) and another distinct edge $e = (x, y)$ such that e intersects the imaginary edges (i, i') and $(i+1, i'+1)$, we say that we *merge h with e* by removing them from X and connecting their respective nodes such that the newly added edges e_1 and e_2 do not cross. For example, for upper hook $h = (i, i+1)$:



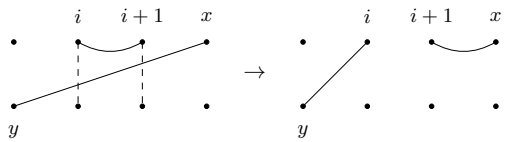
(a) if e is an upper hook such that $x < i$ and $i+1 < y$, then $e_1 = (x, i)$ and $e_2 = (i+1, y)$



(b) if e is a lower hook such that $x \leq i$ and $i+1 \leq y$, then $e_1 = (i, x)$ and $e_2 = (i+1, y)$



(c) if e is a negative transversal such that $x < i$ and $i+1 \leq y$, then $e_1 = (x, i)$ and $e_2 = (i+1, y)$



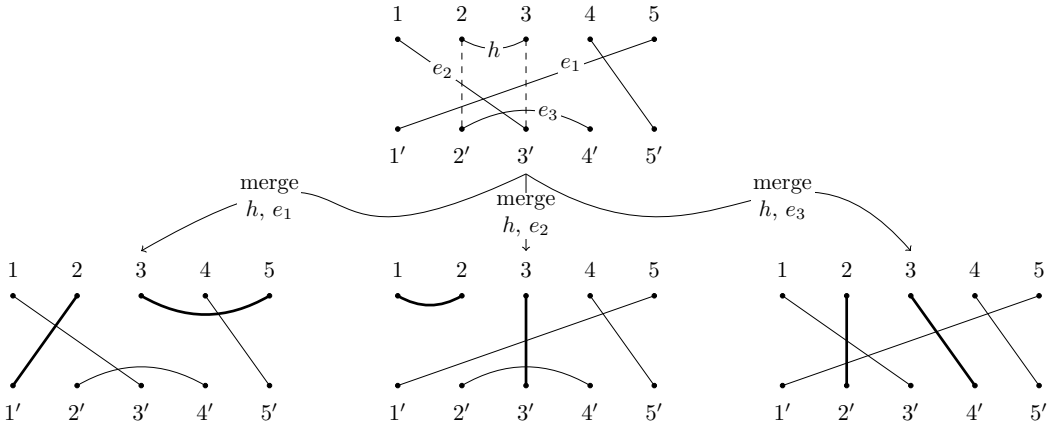
(d) if e is a positive transversal such that $x > i+1$ and $i \geq y$, then $e_1 = (i, y)$ and $e_2 = (i+1, x)$

Assuming that the number of crossings in a tangle corresponds to the number of T-primes in its factorization, we would like this merging process to maintain the crossing number constant, in this way we are sure we are not including more T-primes in the non-minimal factorization we are calculating.

Heuristic 2. *Let $X = U_i X'$ be a $\mathcal{U}$-tangle with c crossings and with an upper hook $h = (i, i+1)$. Let $I = \{(i, i'), (i+1, i'+1)\}$. For all edges $e \neq h$ calculate $inter(e)$ to be the number of intersections e has with edges in I. Let $S = \{e : e \in X, inter(e) = 2\}$ be the set of edges that intersect both edges in I, for each $e \in S$ calculate the number of crossings the tangle X' would have if we merged h with e and pick the tangle whose number of crossings is equal to c. If more than one edge satisfies this last condition, among them, pick the edge that has the least amount of intersections in X.*

Note that, for the case of edges in I, it will happen that some edges in X will share a node with edges in I. We count them as intersecting too.

Take the resulting tangle from the previous example. We have an upper hook of size one $h = (2, 3)$ and three edges that intersect $(2, 2')$ and $(3, 3')$ (labelled as e_1, e_2 and e_3) therefore we have to merge three edges with h. The new edges are highlighted in bold.



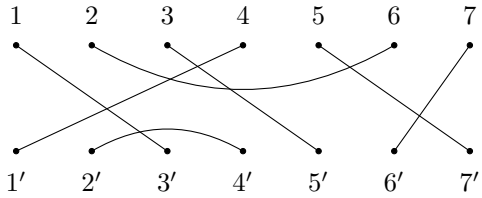
Among these three new tangles, only the first two have the same number of crossings as the original one. We pick the first one.

Merging two edges takes constant time, but calculating the number of crossings takes $\mathcal{O}(N^2)$ [Vernizzi et al., 2016], and since we have to merge h with N edges in the worst case, the time complexity for this heuristic is $\mathcal{O}(N^3)$.

2.3.3 Minimal Factorization

The heuristics mentioned above do not always yield a minimal factorization, therefore a reduction step is required. We can use the axioms presented in Table 1.1 to reduce a non-minimal factorization by implementing them as a Term Rewriting System (TRS) in a rewriting logic tool (we chose the *Maude System* [Clavel et al., 2022]). The TRS accepts in input a factorization and iteratively tries to apply as many delete rules as it can. Then, it tries to apply braid and swap rules non-deterministically until a new delete rule is applicable, and at this point the TRS starts again with the newly shortened factorization. This reduction procedure stops when no delete rule is applicable after all braid and swap rules have been applied.

Using our running example, if we continued to apply our heuristics then we would obtain the following factorization:



$T_5T_4T_3U_2T_4U_3T_1T_2T_6$

The TRS reduces it by performing the following rewrites:

$$
\begin{array}{llllll}
T_5T_4T_3\underline{U_2T_4}U_3T_1T_2T_6 & \text{rule 15} & U_iT_j & \to & T_jU_i & \iff |i-j|>1 \\
T_5\underline{T_4T_3T_4}U_2U_3T_1T_2T_6 & \text{rule 11} & T_iT_jT_i & \to & T_jT_iT_j & \iff |i-j|=1 \\
T_5T_3T_4\underline{T_3U_2U_3}T_1T_2T_6 & \text{rule 7} & T_iU_jU_i & \to & T_jU_i & \iff |i-j|=1 \\
T_5T_3T_4T_2U_3T_1T_2T_6 & \text{STOP} & - & & &
\end{array}
$$

In the last step there are no delete rules applicable and no braid and swap rules that eventually lead to a delete. Therefore, this factorization is minimal.

However, the overall time complexity is difficult to calculate and likely to be huge. This is because, for the symmetric group, it is known [Stanley, 1984] that the maximal number of minimal factorizations is

$$\frac{\binom{N}{2}!}{1^{N-1}3^{N-2}\cdots(2N-3)^1}$$

and the TRS will have to check all of them before deciding that there are no more reductions possible.

2.4 Polynomial time factorization

We have seen so far how the factorization problem for the Brauer monoid could be solved by first finding a non-minimal factorization and then reducing it using a Term Rewriting System. This approach was used in [Marchei and Merelli, 2022], but eventually we found a better algorithm capable of finding a minimal factorization in $\mathcal{O}(N^4)$, which we will describe in this section. To understand how Algorithm 4 works, let us now introduce the concept of a "length function" for $\mathcal{B}_N$.

Definition 10 (Length function for $\mathcal{B}_N$)**.** A length function $\ell(X) : \mathcal{B}_N \to \mathbb{N}$ is a function that returns the minimal number of prime factors required to compose the tangle X, in other words, it returns the length of a minimal factorization for X (define $\ell(I_N) = 0$).

Algorithm 4 Finds the minimal factorization for any tangle in $\mathcal{B}_N$.

Require: $X \in \mathcal{B}_N$
  function FACTORIZE $\mathcal{B}_N(X)$
    $l \leftarrow \ell(X)$
    $F \leftarrow$ empty list
    while $l \neq 0$ **do**
      if $(i, i+1) \in X$ **then**
        $h \leftarrow (i, i+1)$
        for $e \in X, e \neq h$ **do**
          $X' \leftarrow$ merge h with e in X (if defined)
          if $\ell(X') = l - 1$ **then**
            $X \leftarrow X'$
            Append U_i to F
            break
      else
        for $i \in [1 \dots N-1]$ **do**
          $X' \leftarrow T_i X$
          if $\ell(X') = l - 1$ **then**
            $X \leftarrow X'$
            Append T_i to F
            break
      $l \leftarrow l - 1$
    return F

Algorithm 4 works as follow:

1. calculate $\ell(X)$
2. if X has an upper hook h of size one, find another edge e such that, when merged with h, gives a tangle X' with $\ell(X') = \ell(X) - 1$. Record U_i.
3. otherwise, enumerate all possible $X' = T_i X$ until, for some i, $\ell(X') = \ell(X) - 1$. Record T_i.
4. set $X \leftarrow X'$ and repeat until the identity is reached

A key property worth mentioning is that if $\ell(X)$ can be computed in polynomial time, say $t(N)$, then Algorithm 4 returns a minimal factorization in $\mathcal{O}(N^3 t(N))$. This will be proved in Corollary 4.

Lastly, we list two assumptions we believe are true but were not able to prove.

Assumption 1. *Every factorization in the Brauer monoid can be reduced to a minimal one by a sequence of delete, braid and swap rules. In other words, we do not need to increase the factorization length in order to find a shorter one.*

Assumption 2. *If a tangle X has k crossings, then there exists a minimal factorization F with exactly k T-primes and no other factorization with fewer T-primes exists.*

We empirically tested their validity, for the methodology we refer to Appendix A. These two assumptions will be useful for some proofs in the following sections, which are dedicated to presenting the various theorems needed to find such length function, which will be described in Section 2.4.4. We will prove that ℓ can be computed in quadratic time, therefore Algorithm 4 runs in $\mathcal{O}(N^5)$, but some optimizations will bring down the time complexity to $\mathcal{O}(N^4)$.

2.4.1 Mapping $\mathcal{B}_N$ to S_N

Tangles in the symmetric groups S_N are generated only by T-primes. This implies that, given a tangle X in S_N, calculating $\ell(X)$ will amount to just

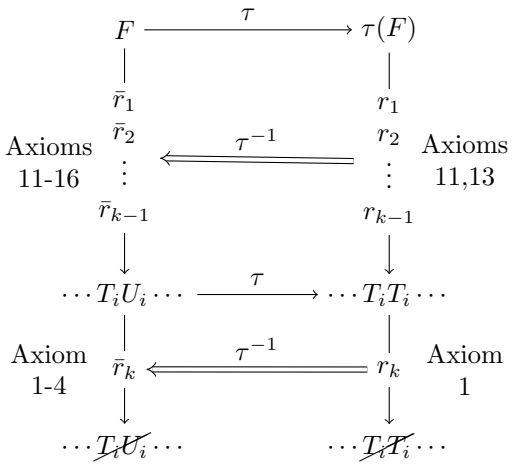
counting its number of crossings, which can be done in $\mathcal{O}(N^2)$ time[1]. With this in mind, it would be useful to have a function $\tau : \mathcal{B}_N \to S_N$ that maps tangles $X \in \mathcal{B}_N$ to tangles in S_N such that $\ell(X) = \ell(\tau(X))$. In this way, we could define $\ell(X)$ by just computing $\tau(X)$ and counting its number of crossings. Therefore, if τ can be computed in polynomial time, then ℓ can be computed in polynomial time too. We now proceed to define τ.

Definition 11. Given a factorization F for a tangle $X \in \mathcal{B}_N$, we define τ as the function that maps each prime factor p_i of F to T_i. In other words, $T_i \mapsto T_i$ and $U_i \mapsto T_i$ for each T and U prime in F.

Theorem 2. *If F is a minimal factorization, then $\tau(F)$ is minimal too.*

Proof. We prove this by contradiction.

Let F be a minimal factorization and assume $\tau(F)$ is not minimal. Therefore, $\tau(F)$ can be rewritten to contain the subword T_iT_i by using axioms 11 and 13 [Björner and Brenti, 2005, Theorem 3.3.1].



Let $s = r_1 r_2 \cdots r_{k-1}$ be a sequence of rewritings for $\tau(F)$ using axioms 11 or 13 and r_k be a rewriting step using axiom 1. For each $r_i \in s$, there exists at least one rewriting $\bar{r}_i$ in the preimage $\tau^{-1}(r_i)$, meaning that r_i applied to $\tau(F)$ maps to a rewriting step $\bar{r}_i \in \tau^{-1}(r_i)$, applied to F, that uses axioms 11 to 16.

This implies that there exists a sequence of rewritings $\bar{s} = \bar{r}_1 \bar{r}_2 \cdots \bar{r}_{k-1}$ for F such that, after $\bar{r}_{k-1}$, F will contain one of the following subwords: T_iT_i, U_iU_i, T_iU_i or U_iT_i. Therefore, $\bar{r}_k$ will use one of the axioms from 1 to 4, thus reducing F to a shorter factorization.

This implies that F was not minimal, which is a contradiction. □

[1]There is a faster approach that brings down the time complexity to $\mathcal{O}(N \log N)$ [Kleinberg and Tardos, 2006], but as we will see in Section 2.4.3, for our purposes it will be much more useful to have the actual factorization for X, which cannot be done faster than $\mathcal{O}(N^2)$.

Corollary 1. *Let F be an arbitrary factorization. If $\tau(F)$ is not minimal, then F is not minimal too.*

Proof. This is the contrapositive of Theorem 2. □

Corollary 2. *Assuming F to be minimal, then $|F| = |\tau(F)|$.*

Corollary 2 allows us to determine the maximal length a minimal factorization in $\mathcal{B}_N$ can have.

Corollary 3. *The longest minimal factorization in $\mathcal{B}_N$ has length $\frac{N(N-1)}{2}$.*

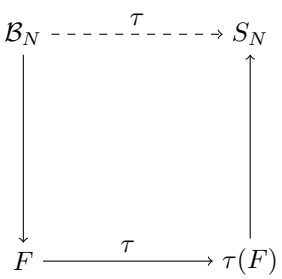
Proof. By Theorem 2 we know that every factorization F has another factorization $\tau(F)$ with the same length composed only by T-primes. Therefore, we only need to check the longest minimal factorization in S_N, which is well-known to be $\frac{N(N-1)}{2}$. □

Since we now know the length of the longest minimal factorization, we can now find the time complexity for Algorithm 4.

Corollary 4. *Algorithm 4 has time complexity $\mathcal{O}(N^3 t(N))$, where $t(N)$ is the time complexity for the length function ℓ.*

Proof. In the case in which there exists edge $h = (i, i+1)$ in $X \in \mathcal{B}_N$, Algorithm 4 iterates through all edges, merges them with h and then calculates ℓ for each of the resulting tangle. The overall complexity for this case is therefore $\mathcal{O}(Nt(N))$. By Corollary 3 we know that the longest factorization possible is quadratic and Algorithm 4 finds each factor one at a time. Therefore, the time complexity for Algorithm 4 is $\mathcal{O}(N^3 t(N))$. □

We can now use the above theorems to find a (not very useful) length function. Given a tangle $X \in \mathcal{B}_N$, assume to have a minimal factorization F for X. Now compute $\tau(F)$, which corresponds to another tangle $\tau(X) \in S_N$. By Corollary 2, we know that the number of crossings of $\tau(X) = |F|$, so we just count them and we obtain the number of factors for X. We can represent this mapping as the diagram on the side. Admittedly, this is not a very useful mapping



because we are assuming to have F in the first place (which is what we are ultimately looking for). What we would like to find now is a way to lift τ to arbitrary tangles, not just factorizations (the dashed arrow above). In this way, if we find a polynomial time algorithm for finding a tangle $\tau(X) \in S_N$ such that its number of crossings is equal to $\ell(X)$, without the need for computing a minimal factorization for X, we would then have a polynomial time length function. We will present such an algorithm in Section 2.4.3.

2.4.2 Passing through and coming back

This section is entirely dedicated to proving that if the number of T-primes an edge e "passes through" is greater or equal to $|e|$, then it does not pass through any U-prime. This is a key property we will leverage in Section 2.4.3, where at the end of it, we will have all the necessary components for computing $\tau(X)$ in polynomial time.

Definition 12 (Passing through)**.** Given a factorization F for a tangle X, a prime tangle $p_i \in F$ and an edge $e \in X$, we say that e "passes through" p_i if one or both of the two edges of p_i:

- $(i, i' + 1)$ and $(i + 1, i')$ if $p_i = T_i$;
- $(i, i + 1)$ and $(i', i' + 1)$ if $p_i = U_i$;

are part of e. We indicate with $\#T(e)$ and $\#U(e)$ the number of T-primes and U-primes respectively e passes through in F.

We will now define what "coming back" means. The goal for these definitions is to prove that if $\#U(e) > 0$ then e cannot come back. This will imply that $\#U(e) > 0 \implies \#T(e) < |e|$ and therefore, by contraposition, $\#T(e) \geq |e| \implies \#U(e) = 0$, which is what we want to prove.

Definition 13 (Coming back at p)**.** Given a factorization F for a tangle X, a prime tangle $p \in F$ and an edge $e \in X$, we say that e "comes back" at p if $|e|$ decreases when passing through p.

Definition 14 (Coming back in F)**.** Given a factorization F for a tangle X and an edge $e \in X$, we say that e "comes back" in F if there exists a prime p at which e comes back.

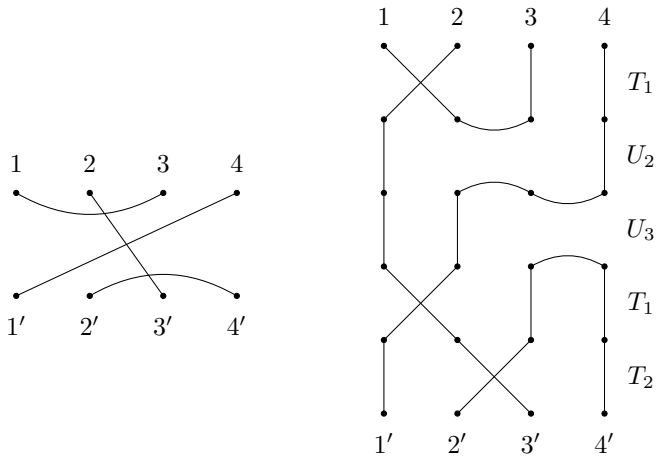


Figure 2.4 A tangle along with one of its minimal factorizations $F = T_1U_2U_3T_1T_2$. The edge $(1, 3)$ passes through T_1 and U_2, edge $(2, 3')$ passes through two T_1s and one T_2, edge $(4, 1')$ passes through U_3, U_2 and T_1 and edge $(2', 4')$ passes through T_2 and U_3. Edge $(2, 3')$ comes back at the bottom-most T_1 because it decreases in size, this also means that it comes back in F. Note also that $(2, 3')$ is the only edge that passes through only T-primes and satisfies the property $\#T(e) \geq |e|$.

See Figure 2.4 for an example.

Theorem 3. *Let $X \in \mathcal{B}_N$, $e \in X$ and F any minimal factorization for X. Then e comes back in F if and only if $\#T(e) + \#U(e) > |e|$.*

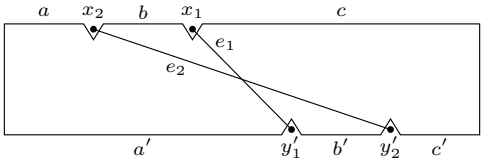
Proof. Forward direction.

Assume e comes back in F, then there exists a prime p_i at which e comes back, but if it comes back at p_i it means e passed through another prime p_i' with the same index i, and since every edge passes through at least $|e|$ distinct primes, it must be that $\#T(e) + \#U(e) > |e|$.

The backward direction uses the same argument. □

We will now use Theorem 3 to prove that a tangle in S_N has a pair of crossing edges e_1 and e_2 (both positive/negative transversals) if and only if e_1 comes back. This theorem will allow us to ignore these tangles in the following theorems, because later on we will assume that a given edge does not come back.

Theorem 4. *Given $X \in S_N$ with a negative transversal $e_1 = (x_1, y_1')$, if there exists another edge $e_2 = (x_2, y_2')$ such that $x_2 < x_1$ and $y_2' > y_1'$ then e_1 comes back in any factorization for X.*



Proof. Let's set some variables:

$$\begin{aligned} a &= x_2 - 1 \\ b &= x_1 - x_2 - 1 \\ c &= N - x_1 \\ a' &= y_1' - 1 \\ b' &= y_2' - y_1' - 1 \\ c' &= N - y_2' \end{aligned}$$

where a, b and c are the number of nodes with indices less than x_2, in between x_2 and x_1, and greater than x_1 respectively (same for a', b' and c'). Therefore, we have that

$$a + b + c = a' + b' + c'$$

implying that

$$a' - a - b = c - b' - c'$$

which counts the least amount of edges that have to cross e_1 from right to left. Notice now that

$$\begin{aligned} x_1 &= b + x_2 + 1 = b + a + 2 \\ y_1' &= a' + 1 \end{aligned}$$

and therefore

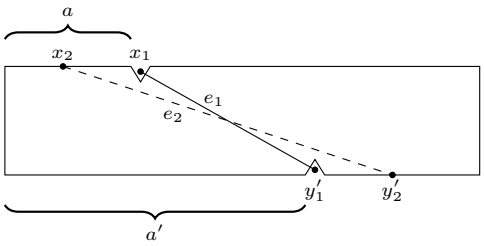
$$|e_1| = y_1' - x_1 = a' + 1 - (b + a + 2) = a' - a - b - 1$$

By substitution, we can now obtain

$$\begin{aligned} a' - a - b &= c - b' - c' \\ |e_1| + 1 &= c - b' - c' \\ |e_1| &< c - b' - c' \end{aligned}$$

which shows that the least amount of edges that have to cross e_1 is bigger than $|e|$, which implies that $\#T(e_1) > |e_1|$ and therefore that e_1 comes back in F (Theorem 3). Note that this proof is not dependent on the factorization chosen, therefore e_1 comes back in all minimal factorization for X. □

Theorem 5. *Given $X \in S_N$ with a negative transversal $e_1 = (x_1, y'_1)$ and minimal factorization F for X, if e_1 comes back in F, then there exists another edge $e_2 = (x_2, y'_2)$ such that $x_2 < x_1$ and $y'_2 > y'_1$.*



Proof. Since e_1 comes back in F, then $\#T(e_1) > |e_1|$ (Theorem 3). Therefore, $\#T(e_1) = |e_1| + c$ where $c \geq 1$.

Let's set some variables:

$$\begin{aligned} a &= x_1 - 1 \\ a' &= y'_1 - 1 = a + |e_1| \end{aligned}$$

Where a is the number of nodes before x_1 and a' is the number of nodes before y'_1. Suppose now that there is no edge $e_2 = (x_2, y'_2)$ such that $x_2 < x_1$ and $y'_2 > y'_1$, therefore we are assuming that all $|e_1| + c$ edges that cross e_1 connect nodes with indices greater than x_1 to nodes with indices less than y'_1.

The number of nodes with indices less than y'_1 connected to edges that do not cross e is therefore:

$$\begin{aligned} a' &- (|e_1| + c) &&= \\ a + |e_1| &- |e_1| - c &&= \\ & & a - c & \end{aligned}$$

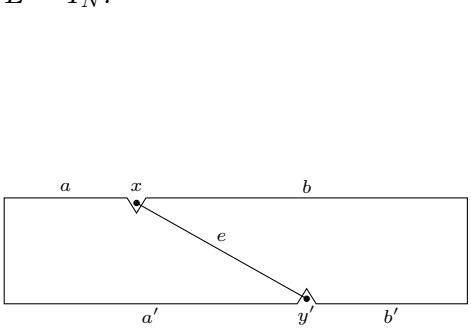
Since we know that $c \geq 1$, we have that $a - c < a$, which implies that not all nodes with indices less than x_1 can connect to lower nodes with indices less than y'_1. Therefore, there must be another edge $e_2 = (x_2, y'_2)$ that crosses e_1 with $x_2 < x_1$ and $y'_2 > y'_1$. □

Corollary 5. *Given $X \in S_N$ with a negative transversal $e_1 = (x_1, y'_1)$, then e_1 comes back in every minimal factorization for X if and only if there exists another edge $e_2 = (x_2, y'_2)$ such that $x_2 < x_1$ and $y'_2 > y'_1$.*

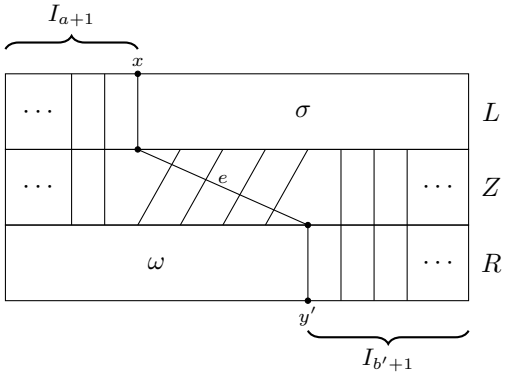
In light of Corollary 5, from now on we will just say that e does not come back in X, since it is independent of the factorization. Assuming that we now have a tangle $X \in S_N$ with an edge e that does not come back, we can always

decompose it as $X = LZR$. This decomposition will come in handy when we will have to prove that a particular factorization is always not minimal, and it will turn out that the two tangles L and R can be safely ignored, simplifying the proof.

Theorem 6 (LZR Decomposition)**.** *Let $a, b \geq 0$ be integers. Let $a', b' \geq 0$ be integers such that $a' > a$ and $a + b = a' + b'$. Let $X \in S_{a+b+1}$ be a tangle containing the edge $e = (a + 1, a' + 1)$ that does not come back. Then X can be decomposed as a composition of three tangles $L, Z, R \in S_{a+b+1}$, where $L = I_{a+1} \otimes \sigma$ with $\sigma \in S_b$, $Z = T_{a+1}T_{a+2}\cdots T_{a+|e|}$ (containing the edge e) and $R = \omega \otimes I_{b'+1}$ with $\omega \in S_{a'}$. This decomposition satisfies the property $\ell(X) = \ell(L) + \ell(Z) + \ell(R)$. In the special case in which $a' + 1 = a + b + 1$ then $L = I_N$.*



(a) A tangle with an edge e that does not come back.



(b) Illustration of the LZR decomposition.

Proof. Suppose a minimal factorization F for X is given. Since we know that e does not come back, then we also know that $\#T(e) = |e|$. This implies that F contains the sub-word $Z = T_{a+1}T_{a+2}\cdots T_{a+|e|}$. Now use the axioms to rewrite F so that $F = L'ZR'$, where L' and R' contain the rest of the factorization. Finally, use axiom 13 to move every prime p_i in L' with $i < a$ into R' and move every prime p_i in R' with $i > a' + 1$ into L'. Thus obtaining $F = LZR$. This obviously satisfies $\ell(X) = \ell(L) + \ell(Z) + \ell(R)$.

In the case in which $N = y' = a + b + 1$, we can construct R by removing edge e from X, relabelling all upper nodes d such that $a + 2 \leq d \leq N$ as $d - 1$ and adding the edge (N, N'). By construction, we have then that $X = ZR$, and therefore we can set $L = I_N$. □

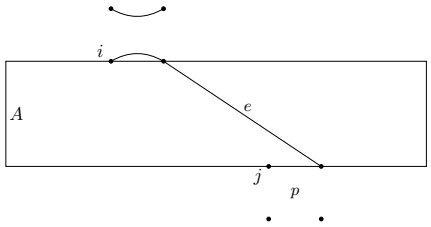


Figure 2.6 The general case in which an edge e "comes back". e passes through a U-prime U_i, then, after a series of T-primes contained in a tangle $A \in \mathcal{B}_N$, it passes through a prime tangle p_j that reduces its size. We would like to prove that the factorization $U_i F_A p_j$ (where F_A is a minimal factorization for A) is always non-minimal.

Finally, we can now prove that if an edge e passes through at least one U-prime then it cannot come back (see Figure 2.6). We assume U_i to be the last U-prime edge e passes through before passing through prime tangle p. This is because in the case in which e passed through another U-prime, say U_j, in a different factorization, we could pick U_j as the focus of the proof. This implies that, after U_i, e passes through only T-primes, in a generic tangle A, before p. We will also assume that it is the first time that e comes back. This assumption also allows us to say that, since e cannot come back the first time, it can never come back.

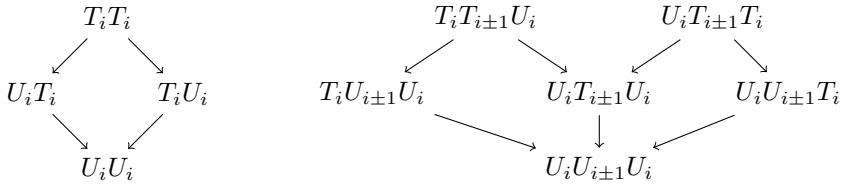
Lastly, we prove only the case in which e is a negative transversal in A and U_i is on top of A. This is because we can reflect $U_i A p_j$ vertically or horizontally to cover any case, and since reflecting a tangle does not change the length of its factorization, then if one of them is not minimal, then all reflections are not minimal too. This implies that an edge cannot come back if at *any time* it passes through a U-prime.

The rest of this section is dedicated to proving that the configuration seen in Figure 2.6 always results in a non-minimal factorization. To do this, we will need to apply $\tau(F)$ to just a subword of F, which we will now prove that it will still result in a minimal factorization assuming F is minimal.

Definition 15. Let $F = p_1 p_2 \cdots p_k$ be a factorization for a tangle $X \in \mathcal{B}_N$, and let $1 \leq i \leq j \leq k$. We define τ^* to be the function that applies τ to only the subword $p_i p_{i+1} \cdots p_j$ of F.

Theorem 7. *If F is a minimal factorization, then $\tau^*(F)$ is minimal too.*

Proof. The proof is similar to the one for Theorem 2, but it requires Assumption 1 because $\tau^*(F)$ can be any factorization in $\mathcal{B}_N$, and every braid/swap r applied to $\tau^*(F)$ has to correspond to a braid/swap in F that is in the preimage of $\tau^*(r)$. There are also more ways in which $\tau^*(F)$ can be non-minimal. See the following Hasse diagrams:



The arrows represent the preimage of τ^*, each element has also a self-loop that was not drawn. For the proof of Theorem 2 only the left Hasse diagram was applicable, but now also the right one has to be taken into consideration. Since we assumed that $\tau^*(F)$ was not minimal, then by Assumption 1 it can be rewritten by a sequence of swaps and braids to contain any element in the above Hasse diagram, but for all such elements there exists a preimage that must be in F and, since all preimages are not minimal, it implies that F was not minimal in the first place, hence we have a contradiction and $\tau^*(F)$ must have been minimal too. □

Corollary 6. *Let F be an arbitrary factorization. If $\tau^*(F)$ is not minimal, then F is not minimal too.*

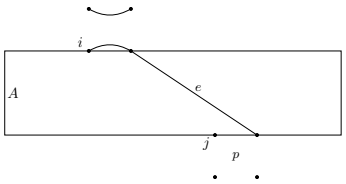
Proof. This is the contrapositive of Theorem 7. □

By using Theorem 6 and Theorem 7 we can now reduce the general case of Figure 2.6 to a simpler one without loss of generality, in which we prove the non-minimality of the factorization $U_iZZ'T_j$, where Z and Z' are the results of two LZR decompositions and thus have the form $Z = T_{i+1}T_{i+2}\cdots T_j$ and $Z' = T_kT_{k+1}\cdots T_{j-1}$ for some $k < j$.

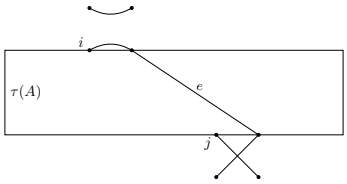
Theorem 8. *Let $X \in \mathcal{B}_N$ be a tangle such that $X = U_i A p_j$, for some $i \leq j$, $A \in \mathcal{B}_N$ and prime tangle p_j. Then, for all minimal factorizations F_A of A, the factorization $U_i F_A p_j$ is always non-minimal.*

Proof. The proof is divided in two parts: we first transform $U_i F_A p_j$ in a different factorization $U_i Z Z' T_j$, and then prove that $U_i Z Z' T_j$ is always non-minimal, implying that $U_i F_A p_j$ was non-minimal too.

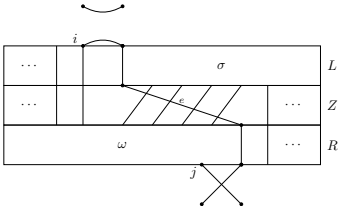
The first part can be described pictorially.



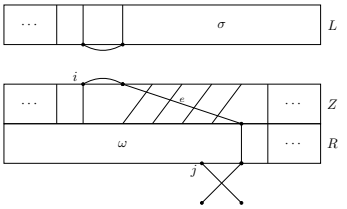
(a) $X = U_i A p_j$



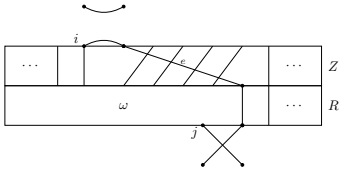
(b) Apply τ on A and p_j as Theorem 7 allows. Remember that e passes through only T-primes, therefore it is still present in $\tau(A)$.



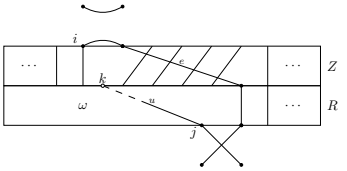
(c) Perform a LZR decomposition on $\tau(A)$ in which $a = i$, $b = N - i - 1$, $a' = j$ and $b' = N - j - 1$.



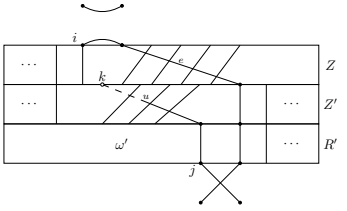
(d) $L = I_{i+1} \otimes \sigma$ with $\sigma \in S_b$ (by Theorem 6). This implies that we can move U_i to be under L.



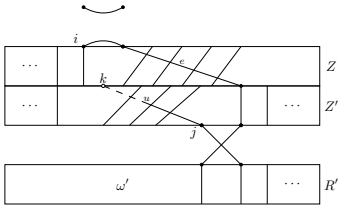
(e) Since L no longer affects edge e, we can delete it.



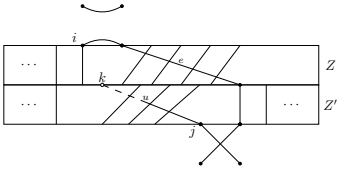
(f) Since $R = \omega \otimes I_{N-j}$ with $\omega \in S_j$, then there exists an edge $u = (k, j')$ for some $k < j$. The white circle and the dashed line indicate that we do not know where k lies on the top row.



(g) Perform a LZR decomposition on R, but since we have that u is connected to j, we then have that L' is the identity tangle and can be ignored (special case for Theorem 6).



(h) As before, we can move T_j above R'.



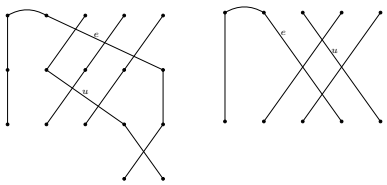
(i) We can now delete R', and this is the simplified case we will prove will always imply a not minimal factorization.

Once we are in the case of $U_i Z Z' T_j$, we will have a factorization in this form:

$$U_i \underbrace{T_{i+1}T_{i+2}\cdots T_j}_{Z} \underbrace{T_k T_{k+1}\cdots T_{j-1}}_{Z'} T_j$$

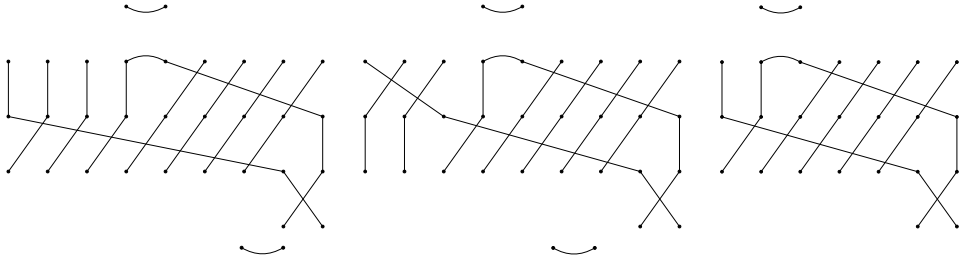
We will go through all cases for k and prove that this factorization is always non-minimal:

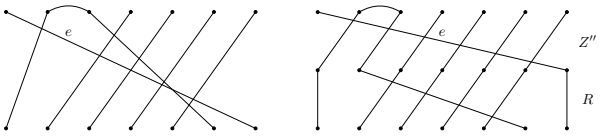
- if $k > i$, then there are too many crossings:



  - the factorization $ZZ'T_j$ has $\ell(Z) + \ell(Z') + 1$ crossings, but edge e and edge u can be redrawn so that they do not cross, thus obtaining the same tangle in S_N but with fewer crossings. This implies that this factorization is not minimal;

- if $k \leq i$, then the factorization contains $U_i T_{i+1} T_i$:





  - every factor in Z' with index $k < i-1$ can be ignored because

$$U_i \underbrace{T_{i+1}T_{i+2}\cdots T_j}_{Z} \underbrace{\overbrace{T_k T_{k+1}\cdots T_{i-2}}^{k<i-1} T_{i-1}T_iT_{i+1}\cdots T_{j-1}}_{Z'} T_j \quad =$$

$$\overbrace{T_k T_{k+1}\cdots T_{i-2}}^{k<i-1} U_i T_{i+1}T_{i+2}\cdots T_j T_{i-1}T_iT_{i+1}\cdots T_{j-1}T_j$$

  therefore reducing the factorization to

$$U_i \underbrace{T_{i+1}T_{i+2}\cdots T_j}_{Z} \underbrace{T_{i-1}T_iT_{i+1}\cdots T_{j-1}}_{Z'} T_j$$

By construction, we have that the tangle $ZZ'T_j$ will contain the edge $e = (i-1, j'+1)$, which falls into the special case of the LZR decomposition. This implies that we can rewrite the above factorization as

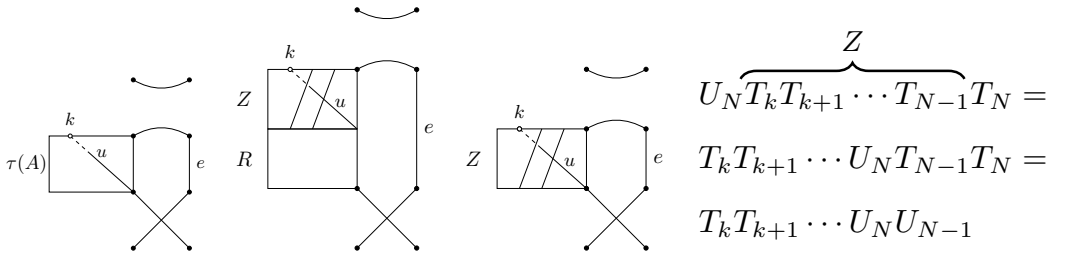
$$U_i Z'' R$$

where $Z'' = T_{i-1}T_i \cdots T_j$ and $R = \omega \otimes I_{N-j}$, where $\omega \in S_j$. This is clearly not minimal because

$$\begin{aligned} U_i Z'' &= \\ U_i T_{i-1} T_i \cdots &= \\ U_i U_{i-1} \cdots & \end{aligned}$$

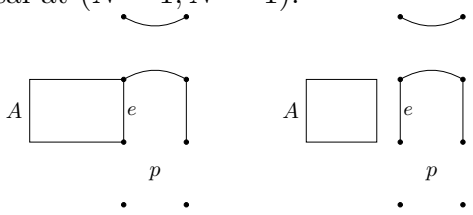
in both cases, we have shown that the factorization $U_i ZZ'T_j$ was not minimal, and therefore $U_i \tau(A)\tau(p_j)$ was not minimal too, but by Corollary 6 this implies that $U_i A p_j$ is also not minimal as required.

There are two more special cases to address. The first one is when edge e is a zero transversal. In this case, the procedure is the same as before, but we perform the LZR decomposition only once.



(a) (b) (c) (d)

The last special case is when not only e is a zero transversal, but A has another zero transversal at $(N-1, N'-1)$.



(a) (b)

In this case, no simplification step is required because the factorization is trivially non-minimal for any prime tangle p. □

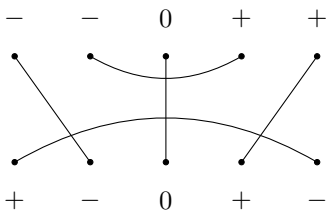
Corollary 7. *Let X be a tangle in $\mathcal{B}_N$ and let $e \in X$. If $\#T(e) \geq |e|$, then $\#U(e) = 0$.*

Proof. If it was the case that $\#U(e) > 0$ but $\#T(e) \geq |e|$, then it would imply that there are more T-primes than $|e|$ and therefore, at some point, e must come back, which we just proved in Theorem 8 is not possible given that $\#U(e) > 0$ and therefore $\#U(e) > 0 \implies \#T(e) < |e|$, which is the contrapositive of $\#T(e) \geq |e| \implies \#U(e) = 0$. □

2.4.3 Node polarity

In this section we will prove that $\tau(X)$ is unique and computable in polynomial time. To do this, we will introduce the concept of "node polarity", a property preserved by τ.

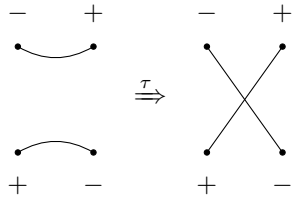
Given a node x, we define the *polarity* of x as follows:



- if x is connected to a transversal edge e, then x is positive (+) if e is positive transversal, negative (-) if it is negative transversal and zero (0) if it is a zero transversal
- if x is connected to an upper hook h, then x is negative (-) if it is the left node of h and positive (+) if it is its right node. The polarity is reversed for lower hooks.

Theorem 9. *τ preserves node polarity.*

Proof. For all edges e such that $\#T(e) \geq |e|$ this is trivially true because they are present in both in X and $\tau(X)$, this includes all edges with node polarity 0.



Therefore we need to prove that τ preserves node polarity for all nodes that are connected to edges e such that $\#U(e) > 0$. As we can see in the diagram on the side, after τ every node of every factor in the factorization will be connected to another one with the same polarity. Since we are assuming that $\#U(e) > 0$, then it implies that e does not come back, and therefore node polarity is preserved. □

Polarity preservation is the first property that $\tau(X)$ must satisfy, but it is not enough because there are many tangles in S_N with the same polarity as X. Lemma 2 and Theorem 10 will state that edges with the same polarity in $\tau(X)$ do not cross, which will imply the uniqueness of $\tau(X)$.

Lemma 2. *Let σ be a permutation containing an inversion ($\sigma(i)$,$\sigma(j)$), with $i < j$. After swapping i with j, the new permutation will have fewer inversions than σ.*

Proof. For readability's sake, we will define $y = \sigma(i)$ and $x = \sigma(j)$. We also define the notation σ_I for the set of elements in σ having indices in the set I.

Let L, C and R be three disjoint sets of indices satisfying $i < L, C, R < j$. Let's also assume that

$$\sigma_L > y > \sigma_C > x > \sigma_R$$

meaning that the elements between y and x can take any value. We do not need to check the elements outside the rage $[i, j]$ because their number of inversions will stay fixed.

We can now observe what happens after swapping y with x. Moving y will add $|\sigma_L|$ inversions because y is smaller than every element in σ_L, while it will remove $|\sigma_C| + |\sigma_R|$ inversions because y is bigger than the elements contained in σ_C and σ_R.

On the other hand, moving x will remove $|\sigma_L| + |\sigma_C|$ inversions because they contain bigger elements, and it will add $|\sigma_R|$ because it contains smaller elements.

Finally, swapping y and x will remove one inversion because we assumed $y > x$.

By summing everything together we obtain that the new permutation will have a different amount of inversions, i.e.:

$$|\sigma_L| - |\sigma_C| - |\sigma_R| - |\sigma_L| - |\sigma_C| + |\sigma_R| - 1 = -2|\sigma_C| - 1$$

We can see that the difference in the number of inversions is always negative, and therefore the new permutation will have fewer inversions. □

Theorem 10. *Let X be a tangle with minimal factorization F. If the nodes of two edges $e_1, e_2 \in \tau(X)$ have the same polarity and e_1 and e_2 are not edges of X too, then they do not cross.*

Proof. We know that τ preserves node polarity and also that since $|\tau(F)|$ is minimal, then it will have the minimal amount of T-primes and therefore the minimal amount of crossings in $\tau(X)$. By Lemma 2 we know that two edges that do not cross add fewer T-primes compared to edges that do cross. Therefore, e_1 and e_2 do not cross in $\tau(X)$. □

We can now finally prove that there exists only one $\tau(X)$.

Theorem 11. *Given a tangle X, there exists only one $\tau(X)$ that preserves node polarity and minimizes the number of crossings.*

Proof. For the sake of brevity, we will assume all mentioned nodes are not connected to edges in both X and $\tau(X)$. We will also just focus on nodes with negative polarity, because for positive polarity the argument is basically the same.

Suppose two upper nodes $x_1 < x_2$ have negative polarity. Assume also that they are the last two upper nodes with negative polarity. Assume now the same for two lower nodes $y'_1 < y'_2$.

If we connect x_1 with y'_2 then it must be the case that x_2 has to be connected to y'_1, thus introducing a crossing, which is forbidden by Theorem 10. Therefore, we have that x_2 must be connected to y'_2 in $\tau(X)$. We are now in a situation in which x_1 and y'_1 are the last nodes that are not connected, and by the previous argument, they must form an edge in $\tau(X)$.

We can repeat this argument until there are no more edges to connect. Since at each step there was only one possible connection to make, it implies that $\tau(X)$ is unique. □

Corollary 8. *For all minimal factorizations F and F' for X, the tangles corresponding to $\tau(F)$ and $\tau(F')$ are equal.*

The proof for Theorem 11 gives also an idea of how τ could be computed. We first start with calculating the node polarity for each node in X from left

to right, but every time we find a node with a certain polarity, say "+", we will label that node with $+j$, where j is a counter that keeps track of how many nodes with polarity "+" we have encountered so far. We can then compute $\tau(X)$ by adding every edge e in X such that $\#T(e) \geq |e|$, and then connecting nodes from top to bottom in $\tau(X)$ if and only if they have the same node polarity label in X (Algorithm 5). See Figure 2.10 for an example. Algorithm 5 has therefore quadratic time complexity because we have to calculate $\#T(e)$ for each edge.

Algorithm 5 Compute $\tau(X)$

Require: $X \in \mathcal{B}_N$

function $\tau(X)$

Let $\rho(i) \in \{+, -\}$ be the polarity for node i in X

$S \leftarrow$ all nodes from 1 to N connected to edge e s.t. $\#T(e) < |e|$

$S' \leftarrow$ all nodes from $1'$ to N' connected to edge e s.t. $\#T(e) < |e|$

For $i \in S$, label with "$\rho(i)j$" the jth node with polarity $\rho(i)$

For $i' \in S'$, label with "$\rho(i')j$" the jth node with polarity $\rho(i')$

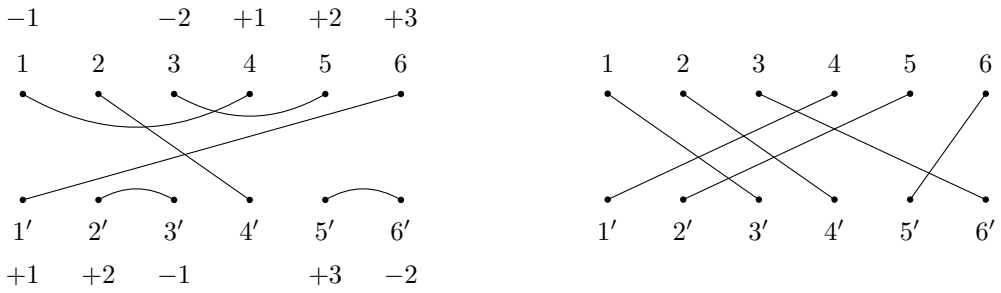
$Z \leftarrow$ empty tangle in $\mathcal{B}_N$

Add to Z all edges in X s.t. $\#T(e) \geq |e|$

for nodes x, y' s.t. x and y' have the same node polarity label in X **do**

Add to Z the edge (x, y')

return Z



(a) $T_3U_4U_5T_2U_1T_3U_2$ (b) $T_3T_4T_5T_2T_1T_3T_2$

Figure 2.10 (a) A tangle X for which we have computed the node polarities along with one of its minimal factorizations. Note that the edge $(2, 4')$ is not considered because $\#T(2, 4') \geq |(2, 4')|$, therefore it passes through only T-primes and it is not affected by τ, which means that it is present in both X and $\tau(X)$. (b) The tangle corresponding to $\tau(X)$ and one of it minimal factorizations. To compute it, add the edge $(2, 4')$ and connect from top to bottom the nodes that in X have the same node polarity label. Note how the indices for both factorizations coincide.

Now that we know that $\tau(X)$ is unique, it is not difficult to see that not only we could count the number of crossings of $\tau(X)$ to find the number of prime tangles for X, but we can also factorize $\tau(X)$ to find the indices of those primes.

Theorem 12. *Given a tangle X with minimal factorization F_X, then there exists a minimal factorization F_Y for $Y = \tau(X)$ such that F_Y has T-primes with the same indices in the same order as in F_X.*

Proof. Since τ does not change the indices, then this is a direct implication of the fact that $\tau(X)$ is the tangle we obtain by composing the factors in $\tau(F_X)$. This factorization can be found by using Algorithm 2. □

2.4.4 Length function

We are now ready to define two length functions for the Brauer monoid. We will use different subscripts to differentiate between them. The first one, ℓ_τ, is trivially $\ell_\tau(X) = \ell(\tau(X))$, where $\ell(\tau(X))$ is just the number of crossings for $\tau(X)$. This function has quadratic complexity because it has to calculate the node polarity first. But even if node polarity could be computed faster, Theorem 12 tells us that we can obtain the indices for a factorization of X by factorizing $\tau(X)$ (which is very useful information, as we will see in Section 2.4.5), and factorizing a tangle in S_N cannot be done faster than $\mathcal{O}(N^2)$ (see Section 2.2.1).

The second length function is a direct corollary of the fact that $\#T(e) \geq |e| \implies \#U(e) = 0$ that we proved in Corollary 7, because it implies that $\#U(e) > 0 \implies \#T(e) < |e|$ and therefore $\#U(e) > 0 \implies \#T(e) + \#U(e) = |e|$. Let's define the function $\#P(e)$ to be the number of prime tangles the edge e passes through in a factorization F.

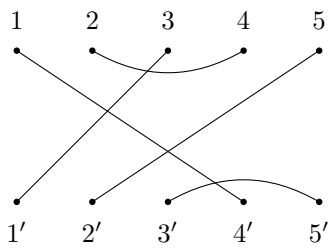
Since if $\#T(e) \geq |e|$ then e passes through only T-primes, and otherwise we have that $\#T(e) + \#U(e) = |e|$, we can define $\#P(e)$ to be:

$$\#P(e) = \begin{cases} \#T(e) & : \ \#T(e) \geq |e| \\ |e| & : \ otherwise \end{cases} = max(\#T(e), |e|)$$

Remember that $\#T(e)$ is just the number of crossings edge e has (Assumption 2). Since now both $\#T(e)$ and $|e|$ are values that are independent of the factorization considered, so is $\#P(e)$. This allows us to use $\#P(e)$ to find a length function for the Brauer monoid by just summing crossings and edge sizes. It is defined as follows:

$$\ell_P(X) = \frac{1}{2} \sum_{e \in X} \#P(e)$$

Since $\#P(e)$ counts the number of primes e passes through, we have that the sum will count every prime twice. We then have to divide by two to obtain the minimal number of primes for the tangle X.



$T_2U_3U_4T_1T_2T_3$

edge	$\#T(e)$	$\|e\|$	max
$(1, 4')$	3	3	3
$(2, 4)$	1	2	2
$(3, 1')$	2	2	2
$(5, 2')$	1	3	3
$(3', 5')$	1	2	2
			12

$$\implies \ell_P = \frac{12}{2} = 6$$

This function still has a quadratic time complexity, but if we assume we already have calculated $\#T(e)$ for all $e \in X$, then it can be computed in linear time. We will use this trick in the following section to bring down the time complexity from $\mathcal{O}(N^5)$ to $\mathcal{O}(N^4)$.

2.4.5 Factorization algorithm

As we have seen in the previous section, every length function has to be computed in quadratic time, therefore by Corollary 4 we have that Algorithm 4 runs in $\mathcal{O}(N^5)$. It turns out however that we can do better. Instead of calculating $\#T(e)$ every time we have to compute ℓ_P, we can store the number of crossings for each edge at the beginning, and then we can just update them every time we merge a lower hook or compose the tangle with a T-prime. In this way, updating will have linear time complexity and therefore ℓ_P will be linear, which brings the overall time complexity to $\mathcal{O}(N^4)$.

We update $\#T(e)$ in different ways depending on whether we composed with a T-prime or merged two edges. In the first case, composing with T_i will modify just two edges, i.e. the ones connected to i and $i+1$. In this case, we just decrease $\#T$ by one for each of them.

In the second case, let's say we merged the upper hook h of size one with the edge e. Since we are going to remove them from X, we need to decrease $\#T$ for all edges that cross with e (no edge will cross with h). Then, after merging h with e, two new edges will be in X, let's call them e_1 and e_2. At this point we iterate again through all the edges of X, and if another edge d crosses with e_1, then we increase $\#T(d)$ and $\#T(e_1)$, we then do the same for e_2.

In both cases updating $\#T$ takes at most linear time, therefore ℓ_P is computable in linear time and we have a $\mathcal{O}(N^4)$ factorization algorithm, we just have to compute $\#T$ once at the beginning. See Figure 2.11 for an example. The final algorithm[2], presented in Algorithm 6, works as follows:

1. Precompute $\#T(e)$ for all $e \in X$
2. Factorize $\tau(X)$ and compute its factorization indices
3. For all factorization indices i:
   (a) If X has an upper hook $h = (i, i+1)$
      i. For all possible mergable edges e in X such that $\#T(e) < |e|$
         A. merge h with e obtaining tangle X'
         B. update $\#T(d)$ for all edges $d \in X'$
         C. if $\ell_P(X') = \ell_P(X) - 1$, record U_i as a factor of X and set $X \leftarrow X'$
   (b) Otherwise,
      i. set $X \leftarrow T_i X$
      ii. Record T_i as a factor of X
      iii. Update $\#T$ for the two edges connected to i and $i+1$
4. Return the recorded factors

Algorithm 6 could be altered to output a factorization that minimizes the number of T-primes without affecting the time complexity by just keeping

[2]A Python implementation can be found at https://github.com/DanieleMarchei/BrauerMonoidFactorization.

track of the tangle with the least amount of crossings when merging e and h. However, the nested for-loops that search for the edge to merge h with are still the bottleneck of the algorithm. One way to bring down the complexity to $\mathcal{O}(N^3)$ could be finding a constant time decision algorithm that determines if a particular edge passes through U_i. In this way, finding the edges that can be merged would take linear time and hence reach $\mathcal{O}(N^3)$. We were not able to find such an algorithm, so we leave it as a future research direction.

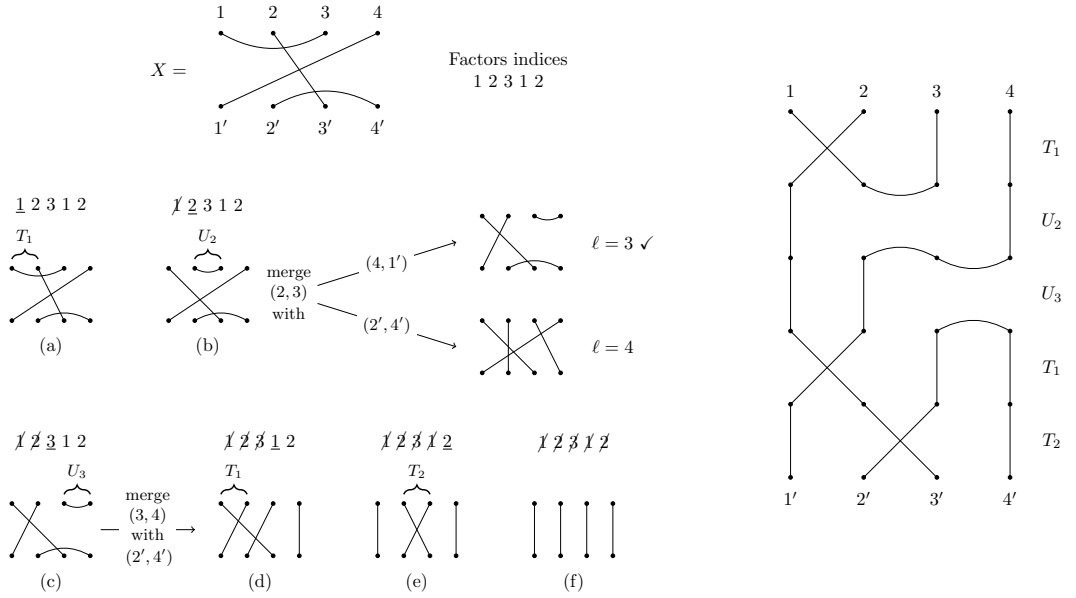


Figure 2.11 High level illustration for how Algorithm 6 works. On the left, we have a tangle X from which we have extracted its factorization indices. (a) The first factor index is 1, therefore we look at upper nodes 1 and 2 of our tangle and see that they are not connected. We record T_1 and compute T_1X for the second step. (b) The second index is 2 and the upper nodes 2 and 3 are connected, therefore we record U_2. For the next step, we have to decide which edge we have to merge $(2, 3)$ with. We have two options: $(4, 1')$ and $(2', 4')$ (edge $(1, 3')$ satisfies $\#T(e) \geq |e|$, so it does not pass through a U-prime). If we merge it with $(4, 1')$ we obtain a tangle with only three factors, while the other will have four, therefore we select the first one for the next step. (c) We are in the same situation of step (b) but now we can only merge $(3, 4)$ with $(2', 4')$ and we record U_3. (d) Same situation of step (a), we record T_1. (e) The factor index is 2 and the upper nodes 2 and 3 are not connected, therefore we record T_2. (f) We have reached the identity tangle, so we stop the algorithm and output the factorization $T_1U_2U_3T_1T_2$, which is minimal. On the right we have drawn the factors we have found. It is easy to check that indeed they compose the original tangle X.

Algorithm 6 $\mathcal{O}(N^4)$ factorization

Require: $X \in \mathcal{B}_N$

function FACTORIZE $\mathcal{B}_N(X)$
  Calculate $\#T(e)$ for all $e \in X$
  $I \leftarrow$ factorization indices of $\tau(X)$ ▷ Algorithm 5 and Algorithm 2
  $F \leftarrow$ empty list
  for $i \in I$ **do**
    $h \leftarrow (i, i+1)$
    if $h \in X$ **then**
      $l \leftarrow \ell_P(X)$
      for $e \in X : e \neq h, \#T(e) < |e|$ **do**
        $X' \leftarrow X$
        for $d \in X' : d \neq h, e$ **do**
          if d crosses e **then**
            $\#T(d) \leftarrow \#T(d) - 1$
        Merge e and h in X', creating edges e_1 and e_2
        for $d \in X' : d \neq e_1$ **do**
          if d crosses e_1 **then**
            $\#T(e_1) \leftarrow \#T(e_1) + 1$
            $\#T(d) \leftarrow \#T(d) + 1$
        for $d \in X' : d \neq e_2$ **do**
          if d crosses e_2 **then**
            $\#T(e_2) \leftarrow \#T(e_2) + 1$
            $\#T(d) \leftarrow \#T(d) + 1$
        if $\ell_P(X') = l - 1$ **then**
          Append U_i to F
          $X \leftarrow X'$
          break
    else
      $X \leftarrow T_i X$
      Append T_i to F
      Decrease the number of crossings for the two edges connected at i and $i+1$
  return F

2.5 Discussion

The Brauer monoid can be factorized in polynomial time, specifically, with a time complexity of $\mathcal{O}(N^4)$. To the best of our knowledge, this is the first polynomial time algorithm proposed to solve this problem. We are not sure if it has an optimal running time, since there might be some room for improvements. We leave this as further research directions. In parallel, we also found two length functions that can be computed in quadratic time, one of which can be computed in linear time assuming the crossing numbers are known.

Some proofs for this chapter rely on two assumptions we were not able to prove nor find in the literature, but their validity has been empirically checked (see Appendix A). We leave their proof as another future research problem.

The generating set for $\mathcal{B}_N$ we used in this work is not the only one, in fact, as it is pointed out in [Kudryavtseva and Mazorchuk, 2006], $\mathcal{B}_N$ can be generated by $\{T_i\}_{1\leq i<N}$ and a single arbitrary U_j. Furthermore, the set $\{T_i\}_{1\leq i<N}$ is not even required, as any generating set for S_N plus U_j would do. We have not explored if and how our algorithm could be adapted to a generating set different to the one used, we leave this as a future research direction.

Typically, if we want to find if two factorizations F_1 and F_2 represent the same tangle, then we could just compute their corresponding tangles and check if they are equal. If instead we would like to have a factorization, rather than a tangle, as a canonical representation, then we could use (for example) the one proposed in [Aicardi et al., 2023], saying that each tangle $X \in \mathcal{B}_N$ can be factorized as follows:

$$X = s_1 U_1 U_3 \cdots U_{2k-2} s_2$$

where $s_1, s_2 \in S_N$ and $k \leq \frac{N}{2}$. Using Algorithm 6, we could define its output as another canonical form for any tangle (or factorization) in $\mathcal{B}_N$. This is because Algorithm 6 is deterministic, and therefore will always output the same factorization for a particular tangle.

The factorization problem could also lead to some interesting combinatorial problems. For example, how many tangles have length k in $\mathcal{B}_N$? Using the results presented, we enumerated all tangles up to $\mathcal{B}_{10}$ and obtained the results shown in Table 2.1, let's call it $T(N, k)$. Some clear patterns emerge, for

example $T(N,1) = 2(N-1)$ (as expected), or $T(N,1) = T(N-2, \frac{N(N-1)}{2} - 1)$ for $N \geq 5$, but we were unable to find a general formula.

k \ N	1	2	3	4	5	6	7	8	9	10
0	1	1	1	1	1	1	1	1	1	1
1		2	4	6	8	10	12	14	16	18
2			8	20	36	56	80	108	140	176
3			2	36	102	208	362	572	846	1192
4				30	196	562	1224	2294	3900	6186
5				10	228	1110	3192	7266	14380	25870
6				2	212	1650	6620	18746	43764	90034
7					106	1966	11090	40166	112250	266462
8					42	1914	15890	73278	247494	682770
9					12	1440	19442	116996	477830	1538840
10					2	830	20910	166400	825422	3100160
11						414	18798	212250	1291638	5667090
12						162	15402	244730	1853554	9514646
13						56	10174	255188	2448214	14804426
14						14	6154	240828	3003652	21502064
15						2	3282	207968	3411904	29298972
16							1530	161844	3627806	37604566
17							648	113490	3585522	45596280
18							234	73978	3325568	52372154
19							72	44336	2856302	57069858
20							16	24354	2325126	59057576
21							2	12462	1741684	58153920
22								5848	1238988	54397782
23								2502	830378	48420890
24								972	523782	41150508
25								324	312886	33243338
26								90	176806	25585214
27								18	94362	18883774
28								2	47280	13337554
29									22294	9028454
30									9756	5856940
31									3908	3653772
32									1406	2186074
33									434	1253770
34									110	688446
35									20	361372
36									2	180488
37										85298
38										37930
39										15636
40										5880
41										1972
42										566
43										132
44										22
45										2

Table 2.1 The number of tangles in $\mathcal{B}_N$ with length k.

Another question could be: what is the maximum amount of edges we can merge for a tangle X with a hook h of size one, such that $\ell(X) = \ell(X') + 1$, where X' is the tangle obtained after the merge? This is important to ask because, as we already discussed, finding the right edge to be merged is the

N	max amount of merges	n. tangles $B(N)$
2	1	1
3	1	6
4	2	2
5	2	48
6	3	18
7	3	936
8	4	360
9	4	32400
10	5	12600

Table 2.2 The maximum amount of possible merges in $\mathcal{B}_N$ and the number of tangles X with a hook h of size one that can be merged with other edges such that $\ell(X) = \ell(X') + 1$, where X' is the tangle obtained after the merge.

bottleneck of Algorithm 6. By means of enumeration, we obtained the results shown in Table 2.2, and it seems the case that the above questions is answered by $\lfloor \frac{N}{2} \rfloor$, while the number of tangles that have that number of merges is much more difficult to count. For example, the even entries match with the A132911[3] sequence of the OEIS [OEIS Foundation Inc., 2024], here we call it $C(k)$:

$$C(k) = (k+1)\frac{(2k)!}{2^k}$$

where k starts from zero. To obtain an exact match with the even entries of Table 2.2, we will call them $B(2k)$, we have to modify[4] it as follows:

$$B(2k) = C(k-1) = k!|\mathcal{B}_{k-1}|$$

where k starts from one. We don't have a proof for any of the above statements, nor we have a candidate formula for the odd entries. We leave these questions open as a further research direction.

[3] https://oeis.org/A132911

[4] By using the identity $|\mathcal{B}_k| = \frac{(2k)!}{2^k k!}$.

Chapter 3

Brauer monoid and RNA

Due to its key role in various biological processes, RNA secondary structures have always been the focus of in-depth analyses, with great efforts from mathematicians and biologists, to find a suitable abstract representation for modelling its functional and structural properties. One contribution is due to Kauffman and Magarshak, who modelled RNA secondary structures as tangles of the Brauer monoid. In this chapter, we extend the tangle-based model with its minimal prime factorization, useful to analyze patterns that characterize the RNA secondary structure. This chapter reports the results published in [Marchei and Merelli, 2022]. Some parts have been adapted using the results from Section 2.4. All RNA figures were drawn using the FORNA tool[1].

3.1 Background on RNA

In biological cells, the RNA (ribonucleic acid) is a molecule that regulates a huge variety of functions. It consists of a long chain of smaller molecules, called nucleotides, bonded sequentially (Adenine (A), Guanine (G), Cytosine (C) and Uracil (U)), known as *primary structure*; the first nucleotide of the chain is usually referred as 5' and the last one as 3'. A *secondary structure* appears when the RNA folds onto itself creating additional *weaker* bonds, called Watson-Crick pairs (A-U, C-G) and Wobble pairs (G-U). Figure 3.1

[1] http://rna.tbi.univie.ac.at/forna/forna.html

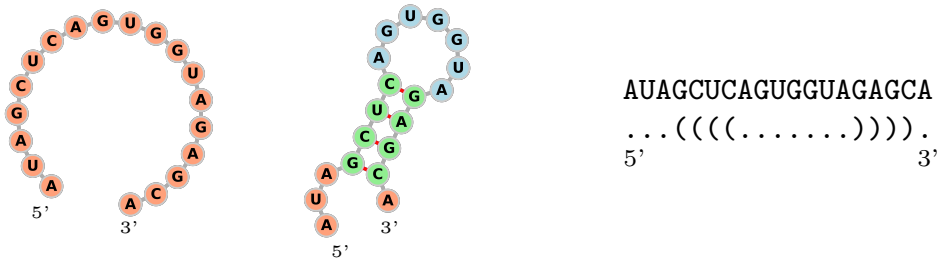


Figure 3.1 RNA found in *Mus musculus* (house mouse). It is primary structure is on the left and the secondary structure is on the right, along with its dot-bracket representation and flattened diagram.

shows a primary and secondary structure of the RNA found in *Mus musculus* (house mouse)[2] along with its *dot-bracket notation*, a string in which a pair of matching brackets correspond to a weak bond in the secondary structure and dots unpaired nucleotides. The dot-bracket string can also be represented by a *flattened diagram*, that is a set of points displayed horizontally (representing the nucleotides) joined by an arc in the upper half part of the diagram (representing the pairs). Since every arc has to connect two dots, every flattened diagram has N arc and $2N$ paired dots.

Depending on the bonds present in the secondary structure, different types of brackets may be needed to avoid ambiguity. The folding process gives rise to some interesting structural features that can be categorized as *hairpins*, *bulges*, *stems*, *interior loops* (see Figure 3.2), and *multiloops* (see Figure 3.3).

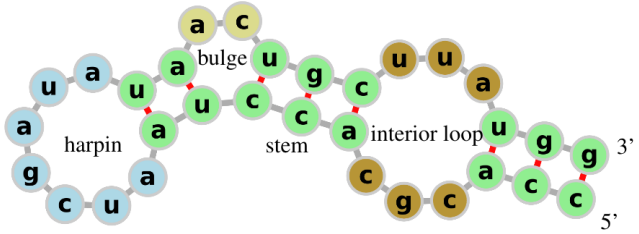


Figure 3.2 Example of various patterns that can emerge from a secondary structure. Blue nucleotides are part of a hairpin, green ones are part of stems, yellow nucleotides are part of a bulge, brown ones are part of an interior loop and orange nucleotides are unpaired.

It is often the case that RNA secondary structures form a *pseudoknot*, where a nucleotide in a hairpin loop is bonded with another nucleotide in a

[2] https://rnacentral.org/rna/URS000029FE6F/10090

different loop of the RNA (Figure 3.3). Predicting the optimal structure with pseudoknots during folding, also known as the *RNA folding problem*, often requires a prohibitive amount of time. Although great efforts were put to solve this problem, both from an algebraic perspective [Bon et al., 2008; Quadrini et al., 2019b; Reeder and Giegerich, 2004; Reidys et al., 2011], and from a machine learning one [Zhao et al., 2021], there is still room for improvements.

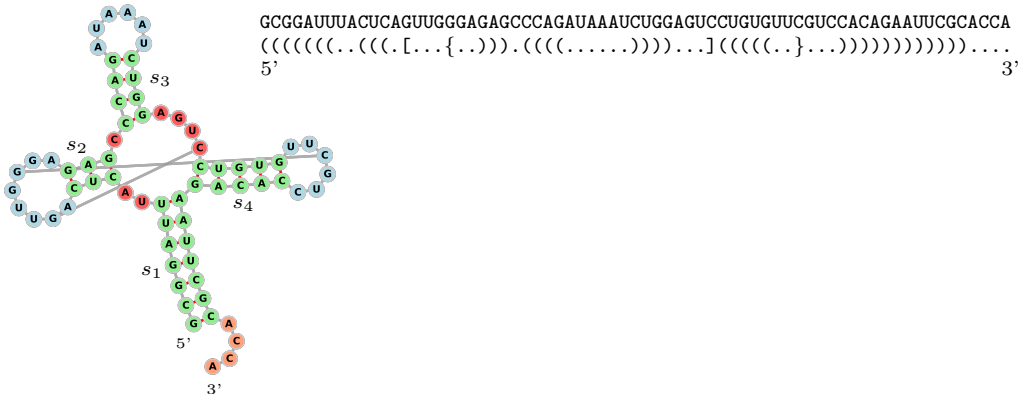


Figure 3.3 Secondary structure of the *yeast phenylalanine tRNA* along with its dot-bracket representation[3]. The folding forms a pseudoknot because of the G-C pair at position 14-42 (labelled with square brackets) and the G-C pair at positions 18-50 (labelled with curly braces). There are three multiloops (colored in red) at the base of the three stems with hairpins. The stems have been labelled from s_1 to s_4 because we are going to refer to them in the following sections.

Due to its pivotal role in biological processes, the study of RNA secondary structures is of great importance. The process of protein production is the result of the interaction of three types of RNA: *transfer RNA* (tRNA), *ribosomal RNA* (rRNA) and *messenger RNA* (mRNA). Viruses have evolved to inject their genome (in the form of RNA) into the host cells in order to replicate themselves. Moreover, it is still in the debate that the self-replicating capabilities of RNA may have given the basis for early life on Earth even before DNA appeared (*RNA World Hypothesis* [Gilbert, 1986; Maestri and Merelli, 2019]).

[3] https://www.rcsb.org/structure/1EHZ

3.2 From RNA to Tangles

The first attempt to draw a connection between RNA secondary structures and tangles in the Brauer monoid was due to Kauffman and Magarshak [Kauffman and Magarshak, 1995]. Their intuition was that the number of parenthesis in RNA dot-bracket representation and the number of dots in a tangle is always even, and each open parenthesis must correspond to a closed parenthesis somewhere in the string, corresponding with the existence of an edge in a tangle. Therefore, they provided the following procedure for converting an RNA secondary structure to a tangle:

1. flatten the secondary structure in a single long chain (equivalent to the dot-bracket notation);
2. discard the unpaired nucleotides, there are now $2N$ nucleotides and N pairs;
3. abbreviate stacked arcs to a single arc. We will call this reduced diagram *shape* [Giegerich et al., 2004; Reidys and Wang, 2010];
4. rotate the second half of the shape diagram above the first;
5. enumerate the nucleotides in the top row with numbers in $[N]$ and nucleotides in the bottom row with numbers in $[N']$.

As Giegerich et al. pointed out, the study of the shape of an RNA secondary structure lifts the user from the burden of paying attention to changes that do not affect the overall desired structure, which means that we do not lose information because we are doing a static analysis [Giegerich et al., 2004]. In this context, the procedure described above gives us the opportunity to study the shape of RNA secondary structures in terms of tangles and generators for these tangles.

We will now provide an example of the mapping procedure for deriving, from a RNA secondary structure, a tangle with its prime factors.

We will start from the yeast phenylalanine tRNA in Figure 3.3, and apply Kauffman and Magarshak's mapping to obtain the flattened diagram in Figure 3.4a.

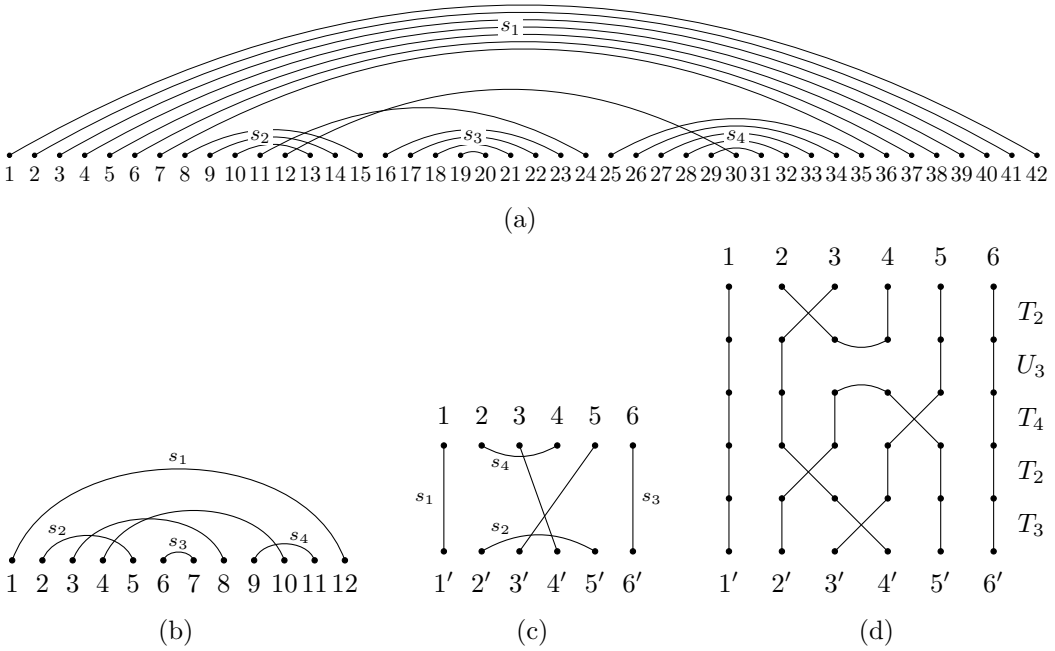


Figure 3.4 (a) The flattened diagram for the yeast phenylalanine tRNA (Figure 3.3). (b) The shape diagram is computed by merging together all parallel edges in the flattened diagram. (c) The corresponding tangle $X \in \mathcal{B}_6$. (d) A minimal factorization for X. The reader can keep track of the stems of Figure 3.3 by following the labels from s_1 to s_4.

This diagram is reduced to obtain a shape diagram (Figure 3.4b) that can be folded to get the corresponding tangle (Figure 3.4c). We can now factorize it by using the methods discussed in the previous chapter, thus obtaining the factorization $T_2U_3T_4T_2T_3$.

An online demo for computing the tangle factorization given an RNA secondary structure is available at https://share.streamlit.io/danielemarchei/rnatotangle/main.

RNA without pseudoknots Figure 3.5a is an example of an RNA molecule that does not have any pseudoknots, therefore its corresponding tangle should not have any crossings. To find it, we apply Kauffman and Magarshak's mapping. We take its secondary structure (represented as a flattened diagram in Figure 3.5b) and reduce it to a shape diagram (Figure 3.5c). The shape diagram can now be folded in half to obtain the tangle in $\mathcal{J}_6$ shown in Figure 3.5d.

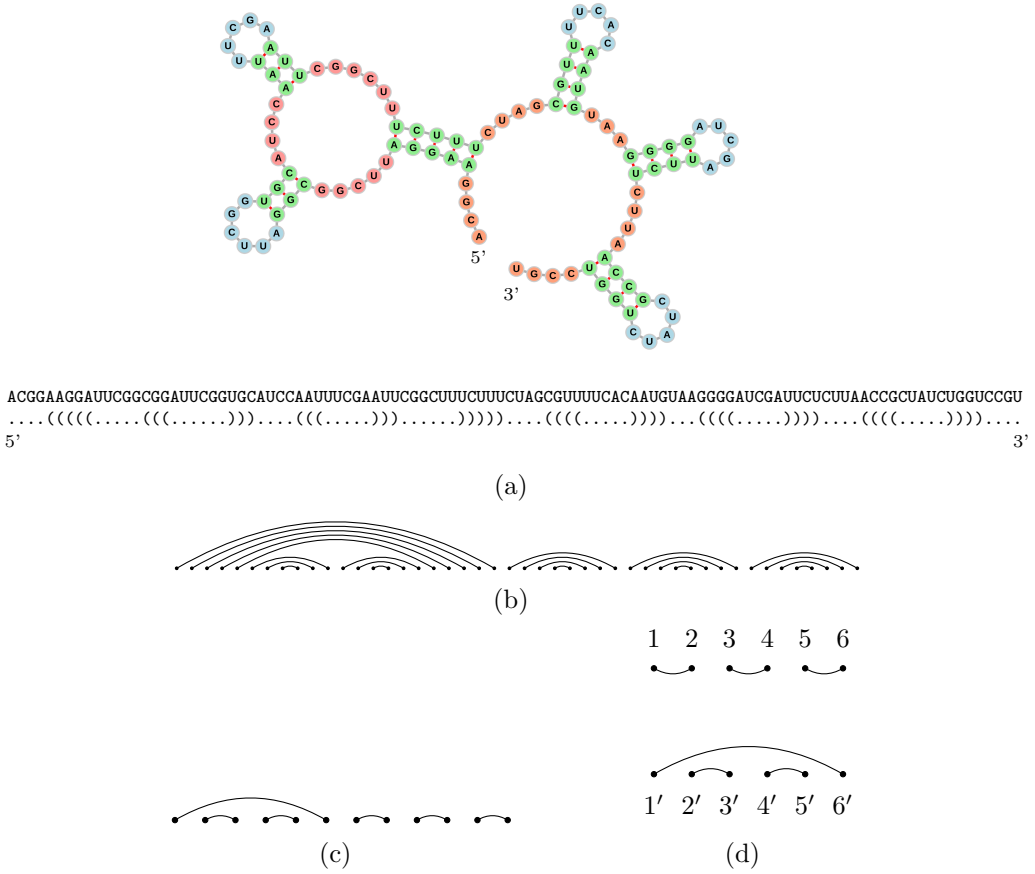


Figure 3.5 (a) A pseudoknot-free RNA secondary structure along with its primary structure and dot-bracket representation. (b) The flattened diagram. (c) The shape diagram obtained by collapsing parallel edges onto a single one. (d) The corresponding tangle.

Using Algorithm 3, we obtain the minimal factorization $U_1U_3U_5U_2U_4$ which, as predicted, has no T-primes.

RNA with pseudoknots We have already seen in Figure 3.3 an example of an RNA secondary structure with pseudoknots and how its corresponding tangle has four crossing. By Assumption 2 then there must be a minimal factorization with exactly four T-primes, and in fact, as shown in Figure 3.4, $T_2U_3T_4T_2T_3$ is one of them.

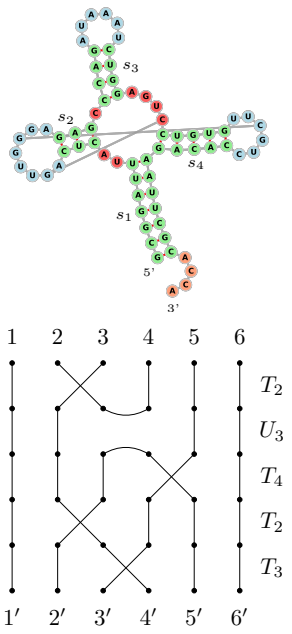
3.3 Some RNA structures reflected in the factorization

After applying the Kauffman and Magarshak's mapping, the resulting tangle is invariant to *synonymous* mutations, which are mutations that do not change the secondary structure. This is due to the fact that we discard unpaired nucleotides and abbreviate stacked arcs, allowing multiple secondary structures to map to the same factorization. This also allows researchers to move their attention to patterns in the factorizations of their desired shapes. A less obvious result (already observed by Kauffman and Magarshak) is that every secondary structure without pseudoknots maps to a tangle in $\mathcal{J}_N$. The intuition behind this result is that the number of valid ways we can arrange $2N$ open and closed parenthesis of a single type is the Catalan number

$$C_N = \frac{1}{N+1}\binom{2N}{N} \tag{3.1}$$

which is exactly the number of tangles in $\mathcal{J}_N$ [Chlouveraki and Pouchin, 2017]. This also implies that every pseudoknotted secondary structure corresponds to a tangle with at least one crossing, and thus at least one T-prime as a factor by Assumption 2.

Let us show some other properties using the RNA shown on the side that we used as example in Figures 3.3 and 3.4.

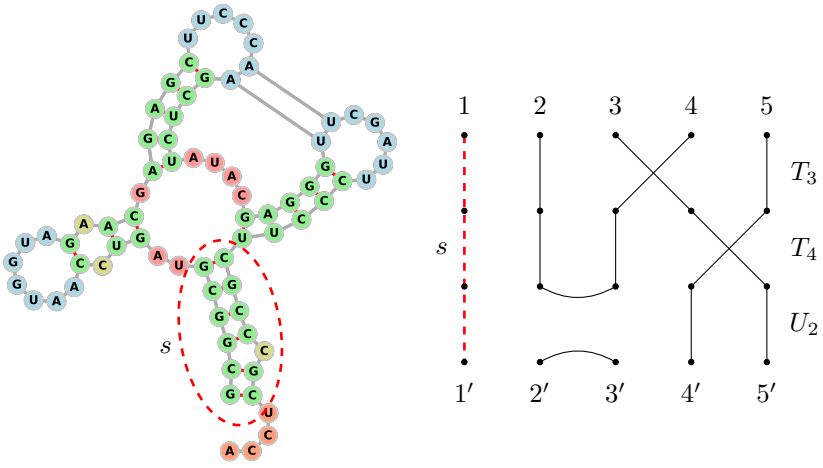


In the corresponding tangle, only stems and pseudoknots are visible and they are encoded in the factorization. Starting from stem s_1, seven pairs are identified with the vertical edge $(1, 1')$, which does not have corresponding factors. Its presence, however, causes the indexes of the prime tangles to be shifted by one (Proposition 1). The three pairs of the stem s_2 correspond to the edge $(2', 5')$, which passes through the factors T_2, U_3 and T_4 (note that T_2 and T_4 can commute, see Section 3.4). The stem s_3, corresponding to the edge $(6, 6')$, which does not

pass through any factor because in the shape diagram the edge $(6, 7)$ is exactly in the middle and, for the same reason of s_1, this causes the factorization to be unable to contain a factor with index five. The stem s_4, identified with the edge $(2, 4)$, passes through T_2 and U_3. Lastly, one pseudoknot is identified by the edge $(3, 4')$, and it passes through the two T_2s and one T_3, while the last pseudoknot, $(5, 3')$, passes through T_4 and T_3. Note that if two edges cross in the corresponding tangle, then they will pass through the same T-prime (Proposition 3), for example $(2, 4)$ and $(3, 4')$ or $(2', 5')$ and $(5, 3')$.

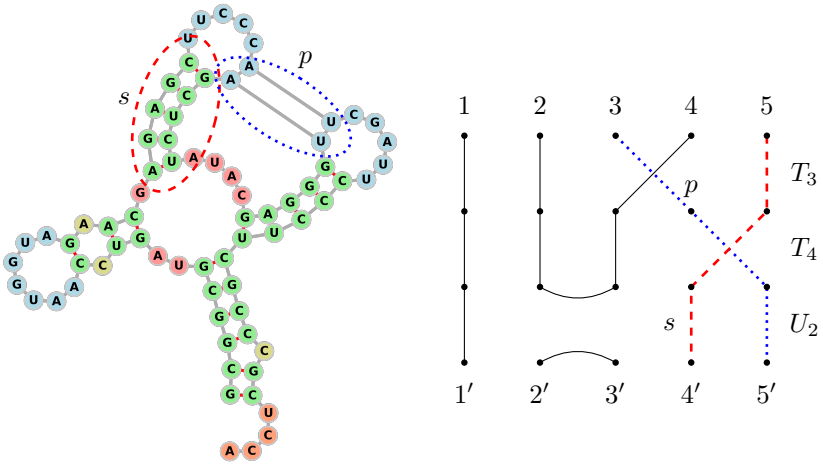
We will give a mathematical foundation for these empirical results. Given a section s of an RNA secondary structure, stem or pseudoknot, we write $edge(s) = (i, j)$ to denote its corresponding edge in the RNA shape (or tangle) beginning in position i and ending in position j (with $i < j$). Given a tangle X and an edge $e \in X$, we write $pt(e)$ to indicate the set of factors in which e passes through.

Proposition 1. *If an RNA secondary structure has a stem s with $edge(s) = (1, 2N)$, then the index of every factor of the corresponding tangle $X \in \mathcal{B}_N$ will always be greater than or equal to two. The converse is also true.*



Proof. Assume that an RNA shape has an edge $e = (1, 2N)$. Let X be the corresponding tangle, then $(1, 1') \in X$ and therefore there is no prime T_1 or U_1 in the factorization of X. The backward argument is also valid. □

Proposition 2. *Let s be a stem of an RNA secondary structure and let p be a pseudoknot starting inside the hairpin of s and ending outside of it. Then $edge(s)$ and $edge(p)$ will cross.*



Proof. We can abstract $edge(s)$ to be a 2-dimensional closed curve $\mathcal{S} \subset \mathbb{R}^2$ by closing its two ends with a horizontal line. We then have that $edge(p)$ starts inside of $\mathcal{S}$ and ends outside of it. By the Jordan Curve Theorem on $\mathbb{R}^2$, we know that $edge(p)$ must cross $\mathcal{S}$, and since we assume that in the shape diagram all edges are situated in the upper portion of the diagram, we know that $edge(s)$ must cross with $edge(p)$. □

Proposition 3. *Let X be a tangle with $e_1, e_2 \in X$ and let $G = pt(e_1) \cap pt(e_2)$. If e_1 and e_2 cross, then there exists $T_i \in G$ for some i.*

Proof. Since e_1 and e_2 cross, they must pass through the same T-prime T_i. This implies that $T_i \in pt(e_1)$ and $T_i \in pt(e_2)$, and therefore $T_i \in G = pt(e_1) \cap pt(e_2)$. □

3.4 Discussion

The existence of equivalent factorizations leads us to ask the following open question:

Open Problem. *What is the biological interpretation of commutative factors and, in general, of equivalent factorizations?*

We hypothesize two separate research directions, regarding:

- equivalent factorizations up to commutativity (swap axioms)
- equivalent factorizations up to braid axioms 11 and 12

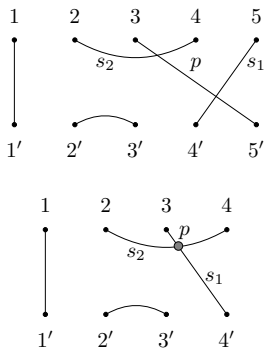
The reason for this distinction is that the swap axioms for $\mathcal{B}_N$ (Table 1.1) do not really impose a challenge during factorization, recall that they are defined as:

$$P_i P_j = P_j P_i \iff |i-j| > 1$$

where P_i and P_j are arbitrary primes tangles. Note how the number of prime factors P_i and P_j remains unchanged on both sided of the equation, whereas in the braid axioms:

$$\begin{array}{lrcll} 11. & T_i T_j T_i & = & T_j T_i T_j & \iff |i-j| = 1 \\ 12. & T_i U_j T_i & = & T_j U_i T_j & \iff |i-j| = 1 \end{array}$$

The number of T_i is two on the left side and one on the right for axiom 11, and for axiom 12, the left side and the right side do not even share a common factor. Since the factorization yielded by axioms 11 and 12 is fundamentally different, we think that they have a different biological interpretation than the swap axioms.



We can also discuss another research direction by analyzing different mappings from RNA secondary structures to tangles. For example, in the mapping we discussed in this chapter, if there is a pseudoknot p connecting stems s_1 and s_2 then in the corresponding tangle there will be three edges, one for each of them. In this framework, the interaction between two stems is represented by an edge intersecting their corresponding edges. We could, instead, think of another mapping in which stems connected by a pseudoknot will have their corresponding edges that cross each other. For example on the side there are two different mappings in which pseudoknots are encoded differently. s_1 and s_2 are two stems and p is a pseudoknot connecting them. The mapping on the top is the one proposed by Kauffman and Magarshak, the one on the bottom is another mapping in which the pseudoknot corresponds to the intersection (grey dot) between s_1 and s_2.

We did not explore this alternative mapping, so we leave it as a future research direction.

Let's discuss now some practical applications our methodology could be used for.

The "representation as factorization" we have discussed in this chapter can be useful as an additional classification criterion for RNA secondary structures databases, in which a user could query RNAs that are generated only by a particular set of prime tangles, without the need of specifying the exact shape of the RNA they are interested in. This could also lead to interesting applications in the context of sequence alignment, in which two sequences are compared not by the alignment of their nucleotides, but by their prime factors.

As we discussed in Section 3.1, the folding problem is the focus of a huge number of researches. In recent years, machine learning techniques have been widely used in this context, in which a model is trained to predict the optimal secondary structure from a sequence of nucleotides [Zhao et al., 2021]. We imagine that a machine learning model could be trained to predict the full factorization of the optimal secondary structure so that its shape would be easily computable or, alternatively, a model capable of predicting just a subset of this factorization, greatly reducing the search space of the optimal structure. We have not investigated this path, so we leave it as a future research direction.

Chapter 4

Phylogenetic networks

In this chapter, we will move away from the Brauer monoid and will aim to demonstrate how the encoding of labellable networks into covers may be of broad use in the classification of network classes. Different classes of networks are defined in different ways, and it can be difficult to present a clear hierarchy (there have been several visual attempts, for instance [Kong et al., 2022, Fig.12] and [Francis and Steel, 2023, Fig.6]). Being able to characterize different network classes by the properties of their covers gives a unified framework for defining networks, in the sense that one may add or remove axioms depending on the class of networks one wants to describe. In that sense, moving from one class to another may be just a matter of changing the axioms, providing a potentially useful lens for visualizing the relationships among classes. This chapter contains the results presented in [Francis et al., 2024].

4.1 Features of vertices in networks and their covers' properties

Many features of vertices in networks have direct translations into the language of covers, and we present some of them in Table 4.1. The first two lines of the table are clear: non-root vertices on a network are labelled by the labelling algorithm and those labels appear as integers in $[m]$, and the leaves are labelled by integers in $[n]$. The other lines of the table can be justified as follows.

A tree vertex in a network is a vertex with in-degree 1, which means it has only one parent and, therefore, is in only one set of sibling vertices. This set of sibling vertices could have any size greater than or equal to one, but it is only a single set. A reticulation vertex, on the other hand, has strictly more than one parent, and thus has two or more sets of siblings. No two vertices in a labellable network have the same set of children [Francis and Steel, 2023, Thm 3.3], so the label of a reticulation vertex will appear in at least two sets in the cover. The other translations in Table 4.1 follow immediately.

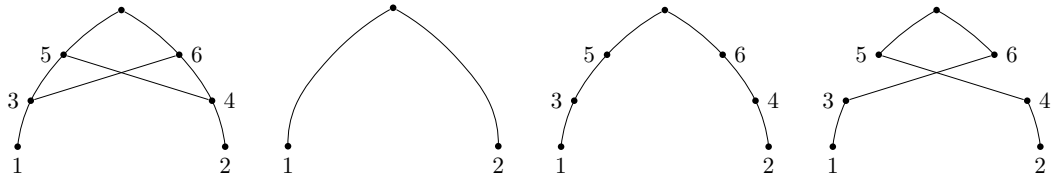
Throughout this chapter, we will add additional translations to the table, with a summary table given in the Discussion.

Network	**Cover**
Non-root vertex	An integer in $[m]$
Leaf	An integer in $[n]$
Tree vertex	An integer contained in just one subset
Reticulation vertex	An integer contained in more than one subset
In-degree of x	The number of subsets that contain x
Out-degree of x	Size of the subset with label x in the labelling order
Parents of x	All the subsets that contain x
Siblings of x	All the other integers contained in the subsets that contain x
Children of x	The subset with label x in the labelling order

Table 4.1 A translation of features of vertices in a labellable network with n leaves and m non-root vertices into features of the corresponding expanding cover.

4.2 Tree-based networks

A phylogenetic network is *tree-based* if it has a spanning tree whose leaves are those of the network [Francis and Steel, 2015]. Such a spanning tree is called a *base tree* for the network. Typically, a tree-based network can have many base trees. A similar notion that we will discuss is that of a *support tree* for a network. A support tree is a base tree but with additional degree 2 vertices where additional arcs are joined to complete the network. That is, the set of vertices in the support tree and the network are identical. See Figure 4.1 for an example.



(a) A non-labellable tree based network N

(b) A base-tree T for N

(c) A support-tree for N. Nodes from 3 to 6 are the additional degree 2 vertices added to T

(d) A spanning-tree for N

Figure 4.1 A classical example of a tree-based phylogenetic network that is not labellable.

Unlike the other classes that we consider in the coming sections, not all tree-based networks are labellable (Figure 4.1a), but neither are all labellable networks tree-based [Francis and Steel, 2023]. There is thus a non-trivial intersection of the two classes, and this intersection contains many other classes, including orchard, tree-child, and normal networks [Francis and Steel, 2023]. In the binary case, the tree-based networks that are labellable can be characterized in terms of their structural properties, as those for which no two reticulate vertices have the same sets of parents [Francis and Steel, 2023, Thm. 6.3]. In this section, we provide a new characterization of the tree-based labellable networks in terms of their covers, and the existence of an "embedded" partition, in Theorem 13.

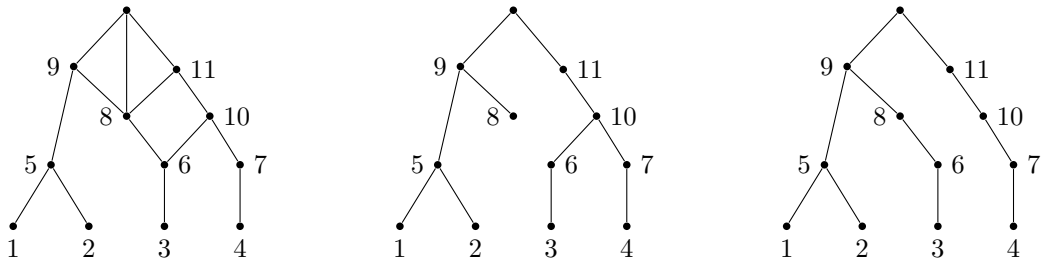
Definition 16. A partition π *embeds* in $\mathcal{C}$ if there is a one-to-one map from π to $\mathcal{C}$ that maps each set A in π to a set A' in $\mathcal{C}$ so that $A \subseteq A'$. A partition π *fully embeds* in a cover $\mathcal{C}$ if π embeds in $\mathcal{C}$ and $|\pi| = |\mathcal{C}|$.

See Figure 4.2 for an example.

Recall from Section 1.2.3 that every partition of $[m]$ is an expanding cover. It is straightforward to see that every expanding cover has a partition that embeds into it, as follows.

Lemma 3. *For every expanding cover $\mathcal{C}$ of $[m]$, there is a partition of $[m]$ that embeds into $\mathcal{C}$.*

Proof. If all repeats of integers in $\mathcal{C}$ are deleted, so that there is one occurrence of each integer, then the result is a partition of $[m]$. □



(a) $\mathcal{C}$ (b) π_1 (c) π_2

$\mathcal{C}$ =	$1,2$	3	4	6	$5,8$	$6,7$	$8,10$	$8,9,11$
π_1 =	$1,2$	3	4		$5,8$	$6,7$	10	$9,11$
π_2 =	$1,2$	3	4	6	$5,8$	7	10	$9,11$

Figure 4.2 A labellable tree-based network with cover $\mathcal{C}$, along with two spanning trees. The partition π_1 for (b) embeds in $\mathcal{C}$ while the partition π_2 for (c) fully embeds in $\mathcal{C}$ because $|\pi_2| = |\mathcal{C}|$.

Any partition obtained in this way will be expanding, according to Lemma 1. Note, however, that each such partition may not have the same number of sets as the cover, and therefore may be expanding for a different value of n.

The notion of embedding a partition into a cover turns out to help characterize tree-based networks.

Theorem 13. *An expanding cover $\mathcal{C}$ of $[m]$ corresponds to a tree-based network if and only if there is a partition π of $[m]$ that fully embeds in $\mathcal{C}$.*

Proof. Suppose N is a tree-based network with expanding cover $\mathcal{C}$ of $[m]$. We will show that $\mathcal{C}$ has an embedded partition with length $|\mathcal{C}|$.

Label the vertices of N according to the labelling algorithm. This labelling gives rise to the expanding cover whose sets are the children of non-leaf vertices in N. Choose a support tree T for N, keeping the labels of the vertices from N. The labels of vertices in T are thus precisely $[m]$. Note that all vertices of N are present in T, but that each non-root vertex in T has in-degree 1. The set of children of each vertex in T is a subset of the set of children for the corresponding vertex in N.

Construct the cover for T using the inherited labelling of vertices, forming sets of labels of vertices that are the children of the same non-leaf vertex. Each set thus formed is a subset of one of the sets in the cover for N, because the

children of vertex i in N are a subset of the children of vertex i in T. Each set is non-empty because the only leaves in the base tree are those of N. The cover for T contains no repeated integers because T is a tree and there are no vertices with in-degree greater than 1. Thus, the cover for T with the labelling inherited from N is a partition of $[m]$ of length $|\mathcal{C}|$, as desired.

Note that the labels on the vertices in T are those inherited from N. They are not the same as the labels that would be put on vertices by the labelling algorithm applied to T. Thus the partition obtained from T is not the same as the partition that would be obtained by labelling T directly.

For the reverse direction, suppose that the expanding cover $\mathcal{C}$ has a partition π that embeds into $\mathcal{C}$, and has length $|\mathcal{C}|$. We will show that the corresponding network is tree-based.

Let N be the network constructed by using $\mathcal{C}$. The partition π embeds in $\mathcal{C}$, so there is a one-to-one map from π to $\mathcal{C}$ that maps each set A in π to a set A' in $\mathcal{C}$ such that $A \subseteq A'$. The sets in $\mathcal{C}$ correspond to vertices in N and give the set of children of each vertex. For each non-leaf vertex in N, $A' \in \mathcal{C}$ labels its children, and there is a corresponding set $A \in \pi$ that is its pre-image in the embedding of π into $\mathcal{C}$, with $A \subseteq A'$.

For the non-leaf vertex in N with children A', delete the edges in N between it and the vertices labelled by $A' \setminus A$, and repeat this for each non-leaf vertex in N. The resulting network now has vertices whose children are labelled by the sets in π. We claim that this resulting network $\hat{N}$ is a support tree for N. We need to show that $\hat{N}$ is a spanning tree whose leaves are those of N.

First, $\hat{N}$ contains all vertices of N, since only edges were removed. Second, it is a tree, since no label is repeated in π by virtue of it being a partition, and therefore no vertex has more than one parent. Third, each vertex v that is not a leaf of N has at least one child, since v has a non-empty set of children whose labels are a set in π (the length of π is $|\mathcal{C}|$), and thus the only leaves of $\hat{N}$ are those of N.

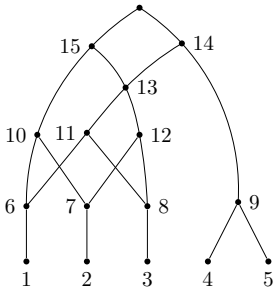
Thus, $\hat{N}$ is a support tree for N, and so N is tree-based, as required. □

This result gives an alternative way to characterize support trees for a tree-based network, as follows.

Corollary 9. *The set of support trees for a tree-based network N is in bijection with the set of full embeddings of partitions in the expanding cover for N.*

Proof. As seen in the proof of Theorem 13, each support tree for N gives rise to a full embedding of a partition in the cover for N. Conversely, every full embedding of a partition into the cover for N constitutes a choice of parent for each reticulation vertex (any element that appears more than once in the cover), and thus gives a support tree for N. □

Note that it is possible for a particular partition to embed in more than one way into a cover, and that each such embedding gives a different support tree for the network.



For example, the figure on the side shows a network with cover $\mathcal{C} = 1 \mid 2 \mid 3 \mid 4,5 \mid 6,8 \mid 6,7 \mid 7,8 \mid 11,12 \mid 9,13 \mid 10,13 \mid 14,15$. The embeddings of partitions into $\mathcal{C}$ can be enumerated as follows. First, consider the elements that appear exactly once in $\mathcal{C}$: $1,2,3,4,5,9,10,11,12,14,15$. These must appear in the partition where they are in the cover (one appearance means only one possibility), so any embedded partition into $\mathcal{C}$ has form

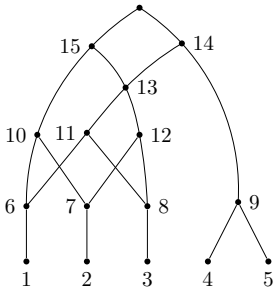
$$1 \mid 2 \mid 3 \mid 4,5 \mid \_,\_ \mid \_,\_ \mid \_,\_ \mid 11,12 \mid 9,\_ \mid 10,\_ \mid 14,15.$$

Consider then the integer 6, which, in the partition, must be either embedded into the set $\{6,7\}$ or $\{6,8\}$. If the former, then 8 must embed into the latter; otherwise, the partition would not be a full embedding (we cannot allow empty sets), which forces 7 to embed into the set $\{7,8\}$. In short, the three sets $6,8 \mid 6,7 \mid 7,8$ can only have embedded either $6 \mid 7 \mid 8$ or $8 \mid 6 \mid 7$. These amount to the same partition but two distinct embeddings that give different support trees because they correspond to different choices of child for each vertex. The other choice for embedding a partition involves the placement of 13, which can either be with 9 or 10.

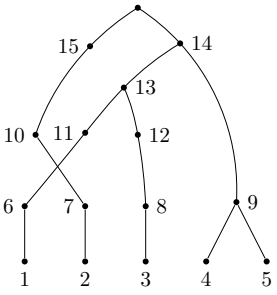
Thus, there are four full embeddings of partitions π_i into $\mathcal{C}$, as follows:

$\mathcal{C}$:	1	2	3	4,5	6,8	6,7	7,8	11,12	9,13	10,13	14,15
π_1 :	1	2	3	4,5	6	7	8	11,12	9,13	10	14,15
π_2 :	1	2	3	4,5	6	7	8	11,12	9	10,13	14,15
π_3 :	1	2	3	4,5	8	6	7	11,12	9,13	10	14,15
π_4 :	1	2	3	4,5	8	6	7	11,12	9	10,13	14,15

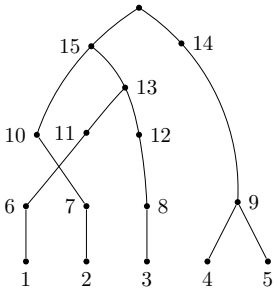
The support trees corresponding to these embeddings (denoted as $T_i(p_i)$) of partitions are shown here:



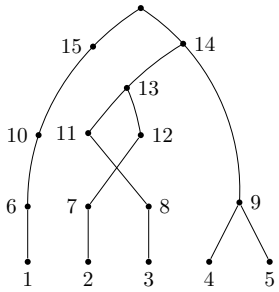
(a) N



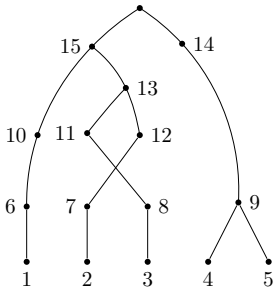
(b) $T_1(\pi_1)$



(c) $T_2(\pi_2)$



(d) $T_3(\pi_3)$



(e) $T_4(\pi_4)$

Network	**Cover**
Spanning tree	A partition embedded in $\mathcal{C}$
Support tree	A full embedding of a partition in $\mathcal{C}$

Table 4.2 Translation of concepts arising in tree-based labellable networks.

4.2.1 Support trees for a binary tree-based network

Support trees for binary tree-based networks have been counted in earlier work [Hayamizu, 2021; Pons et al., 2019], building on an upper bound from [Jetten, 2015]. Covers provide an alternative and clear approach that replicates these results.

For instance (and without giving details of all the components of the statement):

Theorem 14 ([Pons et al., 2019], Theorem 8)**.** *For a binary tree-based network N, the number of support trees is:*

$$2^c \times \prod_{P \in \pi(\mathcal{J}_N)} \frac{1}{2}(v(P)+1),$$

where

- *$\mathcal{J}_N$ is a bipartite graph derived from N with parts given by the set of vertices with a reticulate child, and reticulations without a reticulate parent,*
- *c is the number of cycle components in $\mathcal{J}_N$,*
- *$\pi(\mathcal{J}_N)$ is the set of path components in $\mathcal{J}_N$ without an omnian[1] terminal vertex, and*
- *$v(P)$ is the number of vertices in the path component P.*

This is an explicit formula based on features of the network, using a representation of key features in the bipartite graph $\mathcal{J}_N$ in particular.

It was subsequently demonstrated that this formula relied on two key structural elements of the network: the number of "crowns" and the lengths of each "M-fence" [Hayamizu, 2021, Section 5.3]. These are types of "zig-zag trails", which are undirected paths of vertices in the network that alternate between tree and reticulation vertices [Zhang, 2016]. A maximal length zig-zag trail is called a *crown* if it forms a cycle, and is called an *M-fence* if the ends of the path are tree vertices. Crowns and fences arise naturally when looking at

[1] An omnian vertex in a network is one whose children are all reticulations.

the problem through the lens of covers. We are able to obtain, by using covers, a formula that is analogous to that of Theorem 14, as follows.

Suppose N is a binary tree-based network. We allow degenerate vertices with in-degree 2 as well as out-degree 2. The cover $\mathcal{C}$ for N then consists of sets of size 1 or 2, and each integer appearing in $\mathcal{C}$ appears either once, if it is a tree vertex (in-degree 1), or twice if it is a reticulation (in-degree 2).

We will now describe an algorithm for obtaining an embedded partition (support tree) from $\mathcal{C}$, and this will allow us to count the number of such support trees.

The sets in $\mathcal{C}$ fall into exactly five categories:

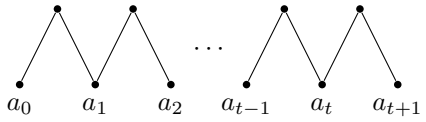
1. Singletons containing integers appearing once in $\mathcal{C}$,
2. Singletons containing integers appearing twice in $\mathcal{C}$,
3. Pairs containing integers each appearing once in $\mathcal{C}$,
4. Pairs containing integers each appearing twice in $\mathcal{C}$, and
5. Pairs containing one integer appearing once and the other appearing twice in $\mathcal{C}$.

Sets that contain elements that appear only once in $\mathcal{C}$ must be fully retained in any embedded partition. Thus sets from categories (1) and (3) must be in the embedded partition, and there is no choice.

Because the partition embeds into $\mathcal{C}$, a set containing a singleton $\{a\}$ in $\mathcal{C}$ must also appear in the embedded partition. Therefore, if $\{a\}$ is in category (2), none of the other occurrences of a in other sets in $\mathcal{C}$ can appear in the partition, and we delete them from the sets in the cover. This will create new sets of size 1, and possibly of category (2). We repeat this process until all sets in category (2) are gone, creating a new cover we denote $\mathcal{C}_1$. Note that $\mathcal{C}_1$ is uniquely determined from $\mathcal{C}$ and embeds into it. Note also that $\mathcal{C}_1$ does not contain any sets in category (2) above.

This leaves sets from categories (4) and (5) to deal with. These sets are connected. If a set is in category (5), then one of its elements appears elsewhere, and it can only be in a set from category (5) or (4). We can thus form sequences

of such sets in $\mathcal{C}_1$ by connecting a set from category (5) with a sequence of sets from category (4) and ending with another set from category (5). These sequences are uniquely determined by $\mathcal{C}_1$, and every set from category (5) is in precisely one sequence of this form. For example, such sequences are of the form

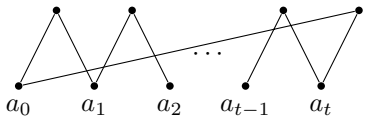


$$a_0, a_1 \mid a_1, a_2 \mid \cdots \mid a_{t-1}, a_t \mid a_t, a_{t+1} \tag{4.1}$$

where a_0 and a_{t+1} do not appear elsewhere in $\mathcal{C}_1$ (note that t could be 1). We call such sequences *fences* (they correspond to the M-fences defined above). The notions of crowns and fences for covers are summarized in Table 4.3.

Let $\mathcal{F}$ denote the set of fences in N. For each fence f, let $r(f)$ denote the number of repeated integers in f, which we call its length. The fence in Equation 4.1 has length $r(f) = t$.

A set from category (4) may be in a sequence such as the one above, or in a sequence of at least three sets from the same category:



$$a_0, a_1 \mid a_1, a_2 \mid \cdots \mid a_{t-1}, a_t \mid a_t, a_0 \tag{4.2}$$

where $t \geq 2$. These correspond precisely to the 'crowns' of [Hayamizu, 2021].

For either fences or crowns, we can count the number of selections of unique elements as follows.

In the case of fences of length t (Equation 4.1), the number of choices is simply $t + 1$, since there are $t + 2$ elements to go into $t + 1$ non-empty sets, so one has two elements and the rest have one element. There are $t + 1$ choices for the set with two elements. For example, with the fence $a, b \mid b, c \mid c, d \mid d, e$,

we have $t = 3$ and the choices are:

$$a, b \mid c \mid d \mid e$$
$$a \mid b, c \mid d \mid e$$
$$a \mid b \mid c, d \mid e$$
$$a \mid b \mid c \mid d, e.$$

In the case of a crown, as in Equation 4.2, there is only one embedded partition. We have the same number of elements as we have non-empty sets, and so there is only one option for selecting unique elements. Each element forms a singleton. For example, in the crown $a, b \mid b, c \mid c, d \mid d, a$, we have only $a \mid b \mid c \mid d$. However, although there is only one embedded partition, that partition has exactly two distinct embeddings. We could have:

$$a \mapsto \{a, b\},\ b \mapsto \{b, c\},\ c \mapsto \{c, d\} \text{ and } d \mapsto \{d, a\},\ \text{or}$$
$$b \mapsto \{a, b\},\ c \mapsto \{b, c\},\ d \mapsto \{c, d\} \text{ and } a \mapsto \{d, a\}.$$

Therefore, we have shown the following result, which is equivalent to Theorem 14:

Theorem 15. *Let N be a binary tree-based network with cover $\mathcal{C}$. The number of embedded partitions in $\mathcal{C}$, and therefore the number of support trees for N, is*

$$2^c \times \prod_{f \in \mathcal{F}} (r(f) + 1)$$

if $\mathcal{F}$ is non-empty, and is 2^c if $\mathcal{F} = \emptyset$, where c is the number of crowns in $\mathcal{C}$.

Note that the number of crowns c is the same as the number of components referred to in Theorem 14.

Given a cover $\mathcal{C}$, we can compute the number of crowns and the lengths of fences, and thus the number of embedded partitions, by using Algorithm 7, which uses the definition of 'acquaints'.

Definition 17. Set $x \sim y$ if $x = y$ or x, y are siblings, and consider the transitive closure of $\sim$, which is an equivalence relation on the set of vertices of the network. Two vertices in an equivalence relation are said to be *acquaints* of each other.

Acquaints can be defined self-referentially by saying that an *acquaint* of a vertex x is a sibling of x or is a sibling of an acquaint of x.

Fences and crowns can be described in terms of acquaints, as follows.

Theorem 16. *Let N be a binary tree-based network with cover $\mathcal{C}$. Then*

1. *N has a fence if and only if there exists a set of acquaints in which exactly two vertices that appear uniquely in $\mathcal{C}$ have one sibling.*
2. *N has a crown if and only if there exists a set of acquaints in which no vertex has one sibling.*

Proof. (1) For the forward direction, suppose that we have a fence like that in Table 4.3. The integers in the set $\{a_0, a_1, a_2, \ldots, a_{t-1}, a_t, a_{t+1}\}$ are acquaints, a_0 and a_{t+1} have only one sibling (a_1 and a_t respectively), and they appear uniquely by assumption.

Conversely, assume there is a set of acquaints in which exactly two vertices (say a_i and a_j) that appear uniquely in $\mathcal{C}$ have one sibling. Since we assume that the network is binary, a_i and a_j appear in only one subset, but they can not be in the same one; otherwise, they would not be acquainted with the other vertices.

It is also the case that every other vertex will appear in exactly two subsets; otherwise, it would imply an in-degree greater than 2, which is not allowed in a binary network. Therefore, we have a set of a type described in Table 4.3, and the network has a fence.

(2) For the forward direction, suppose we have a crown (as indicated in Table 4.3). The integers in the set $\{a_0, a_1, a_2, \ldots, a_{t-1}, a_t\}$ are acquaints, and none of them has exactly one sibling.

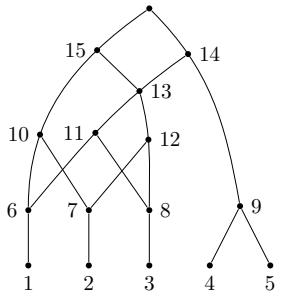
Conversely, assume there is a set of acquaints in which no vertex has one sibling. Since we assume the network is binary, every vertex will appear in

exactly two subsets; otherwise, it would imply an in-degree greater than 2, which is not allowed in a binary network. On the other hand, if a vertex appeared in exactly one subset, this would imply that it had only one sibling, which violates the assumption. Therefore, we have a set of a type described in Table 4.3, and the network has a crown. □

According to the theorem above, we can use Algorithm 7 to count the number of embedded partitions by enumerating the acquaints of all integers that are inside a set of size 2, because, in the definitions of crown and fences (Table 4.3), they do not contain sets of any other sizes. The algorithm takes a cover $\mathcal{C}$ and works as follows:

1. Let $c = 0$ and $f = 1$
2. List all integers in $\mathcal{C}$ and record their siblings
3. Mark every integer as "not visited"
4. While there is an integer i marked as "not visited"
   (a) Mark i as "visited"
   (b) Compute the set of acquaints A of i, marking all of them as "visited"
   (c) If A contains zero integers with one sibling
   - A is a crown, $c \leftarrow c + 1$

   (d) If A contains two integers a, b with one sibling and both appear only once in $\mathcal{C}$
   - A is a fence, $f \leftarrow f \times (|A| - 1)$. (note that $|A| = r(f) + 2$)
5. Return $2^c \times f$



We saw in the previous section that the cover for the binary tree-based network on the side has four embedded partitions, and hence the network has four support trees. These can be counted using Theorem 15 as follows. The cover $\mathcal{C} = 1 \mid 2 \mid 3 \mid 4, 5 \mid 6, 8 \mid 6, 7 \mid 7, 8 \mid 11, 12 \mid 9, 13 \mid 10, 13 \mid 14, 15$ has one crown, namely $6, 8 \mid 6, 7 \mid 7, 8$, and one fence $9, 13 \mid 10, 13$, which has length 1 (a single reticulation). Hence, the number of support trees is $2^1 \times (1 + 1) = 4$, as expected.

Algorithm 7 Count the number of support trees in a binary tree-based labellable network.

procedure TraverseAcquaints(τ, i, A)
  add i to A
  mark i in τ as visited
  for $s \in \tau_i$ such that s is not marked as visited **do**
    TraverseAcquaints(τ, s, A)

procedure CountSupportTreesBinaryNetworks($\mathcal{C}$)
  $\sigma_i \leftarrow$number of times integer i appears in $\mathcal{C}$
  $\mathcal{C}^{=2} \leftarrow$ all subsets of $\mathcal{C}$ of size 2
  $I \leftarrow$ integers appearing in $\mathcal{C}^{=2}$
  $\tau \leftarrow$ table in which index $i \in I$ contains all siblings of i ▷ See Table 4.1
  $c \leftarrow 0$ ▷ Number of crowns
  $f \leftarrow 1$ ▷ Number of fences
  while τ not all i are marked as visited **do**
    select i not marked as visited in τ
    $A \leftarrow \emptyset$
    TraverseAcquaints(τ, i, A) ▷ A will contain the acquaints of i
    $A_1 \leftarrow$ set of integers i in A that have one sibling in τ_i
    if $|A_1| = 0$ **then**
      $c \leftarrow c + 1$ ▷ Theorem 16
    if $|A_1| = 2$ **then**
      $a, b \in A_1$
      if $\sigma_a = 1$ and $\sigma_b = 1$ **then**
        $f \leftarrow f \times (|A| - 1)$ ▷ Theorem 16. By Theorem 15, we have to add 1 to the length (note that $|A| = r(f) + 2$)
  return $2^c \times f$

Network	Cover
Crown	Collection of sets $a_0, a_1 \mid a_1, a_2 \mid \cdots \mid a_{t-1}, a_t \mid a_t, a_0$.
Fence	Collection of sets $a_0, a_1 \mid a_1, a_2 \mid \cdots \mid a_{t-1}, a_t \mid a_t, a_{t+1}$ with $a_0 \neq a_{t+1}$ both appearing uniquely.

Table 4.3 Translation of concepts arising from counting support trees for binary tree-based labellable networks.

4.3 Tree-child networks

Tree-child networks are phylogenetic networks for which every vertex has a child that is a tree vertex [Cardona et al., 2008b]. They satisfy a number of important properties. For instance, they have the property that every vertex is *visible*. This is a property that we describe in Section 4.3.1, but first, tree-child networks turn out to have a very natural description in terms of covers, as follows.

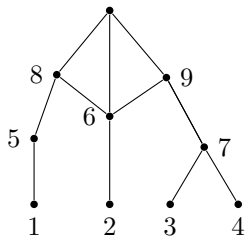


Figure 4.4 Example of a tree-child network. Its cover is $\mathcal{C} = \mathbf{1} \mid \mathbf{2} \mid \mathbf{3}, \mathbf{4} \mid \mathbf{5}, 6 \mid 6, \mathbf{7} \mid 6, \mathbf{8}, \mathbf{9}$. Integers appearing once in the cover have been highlighted in bold. Note how all of them have in-degree 1 and therefore are tree-vertices.

Theorem 17. *Tree-child networks are in bijection with expanding covers for which each set contains an integer that appears exactly once in the cover.*

Proof. The proof relies on the fact that the integers that appear precisely once in a cover are exactly the tree vertices.

Let N be a tree-child network with expanding cover $\mathcal{C}$. Each non-leaf vertex v in N corresponds to a specific set C_v in $\mathcal{C}$, whose elements label the children of v in N. Because N is a tree-child network, each such vertex v has at least one child that is a tree vertex. The labels of the tree vertices appear precisely once in the cover, so the set C_v contains at least one element that appears precisely once in the cover. This holds for every non-leaf vertex, and so for every set in $\mathcal{C}$, which establishes the forward direction.

The reverse direction is also straightforward. Suppose that every set in an expanding cover $\mathcal{C}$ has an element that appears precisely once in $\mathcal{C}$. Since each set in the cover is the set of labels of the children of a non-leaf vertex, this implies that every non-leaf vertex has at least one child whose label appears once in the cover. In other words, it is a tree vertex. Thus, the network corresponding to $\mathcal{C}$ is a tree-child network. □

4.3.1 Visible vertices

An important property of tree-child networks is that all of their vertices are *visible* [Cardona et al., 2008b, Lemma 2]. A vertex v in a network is visible if there is a leaf x for which every path from the root to x passes through v. In this section, we show how visibility can be interpreted by using covers, beginning with the definition of the *backtrack* of a label in a cover.

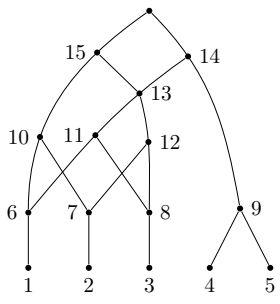
Definition 18. Let $\mathcal{C}$ be an expanding cover in labelling order and let x be an element of $[m]$. Then a *backtrack for* x is a sequence of sets $S_1, \ldots, S_t$ in $\mathcal{C}$ for which the label of a set containing x is in S_1, and the label of S_i is an element of S_{i+1} for each $i = 1, \ldots, t-1$. This corresponds to the output of Algorithm 8. Let $B_{\mathcal{C}}(x)$ denote the set of all backtracks of x in $\mathcal{C}$.

Algorithm 8 Backtracking algorithm

Require: Expanding cover $\mathcal{C}$ in labelling order
  procedure BACKTRACK($\mathcal{C}$, x)
    $s \leftarrow$ a subset of $\mathcal{C}$ containing x
    seq $\leftarrow [s]$ ▷ A list containing s
    while label of s is not ρ **do**
      $s \leftarrow$ a subset of $\mathcal{C}$ that contains the label of s as an element
      add s to seq
    return seq

For example, recall the cover in labelling order for the network on the side:

$$\{1\}_6, \{2\}_7, \{3\}_8, \{4,5\}_9, \{6,7\}_{10}, \{6,8\}_{11}, \{7,8\}_{12}, \{11,12\}_{13}, \{9,13\}_{14}, \{10,13\}_{15}, \{14,15\}_{\rho}$$



A backtrack for $x = 3$ starts with a subset containing 8 (with the label of $\{3\}$ in the labelling order). There are two choices; suppose we pick $\{7,8\}$. The label of $\{7,8\}$ is 12, so now we must find a set containing 12. There is only one, so we add $\{11,12\}$ to the backtrack sequence. $\{11,12\}$ has label 13, so we look for a set containing 13 and choose one of the two options, say $\{10,13\}$. This has label 15 in the order, so we look for a set containing 15. There is one, namely $\{14,15\}$, and its label is ρ, which means the algorithm stops and outputs the backtrack $\beta = \{3\}_8, \{7,8\}_{12}, \{11,12\}_{13}, \{10,13\}_{15}, \{14,15\}_{\rho}$. Note, the sequence of labels of β defines a path from 3 to the root ρ; in this case, $8 \to 12 \to 13 \to 15 \to \rho$.

We can characterize visibility in a network by using the backtracking algorithm. Given $x \in [m]$ and a backtrack β for x, we define $L(\beta) = \{\text{label of } s \mid s \in \beta\}$. In this way, $L(\beta)$ contains the vertices of a path from x to ρ (the root), $\bigcup_{\beta \in B_C(x)} L(\beta)$ is the set of all vertices that *can be* visited with a path from x to ρ, and $\bigcap_{\beta \in B_C(x)} L(\beta)$ is the set of all vertices that *must be* visited on a path from x to ρ.

Theorem 18. *Given a cover $\mathcal{C}$ in labelling order and $x \in [m]$, x is a visible vertex in the corresponding network if and only if there exists $y \in [n]$ such that $x \in \bigcap_{\beta \in B_{\mathcal{C}}(y)} L(\beta)$.*

Proof. For the forward direction, assume that a vertex x of a network is visible. By definition, there exists a leaf y (in other words, $y \in [n]$) such that all paths from the root to y pass through x. Since $\bigcap_{\beta} L(\beta)$ is the set of all vertices we *have to* visit from y to ρ, x must be in this intersection.

For the backward direction, let $y \in [n]$ and $x \in \bigcap_{\beta \in B_{\mathcal{C}}(y)} L(\beta)$. Then it means that all paths from y to ρ contain x. Therefore x is visible in the corresponding network. □

Since all $x \in \bigcap_{\beta \in B_C(x)} L(\beta)$ are visible vertices and vice versa, we obtain the following corollary.

Corollary 10. *Given a cover $\mathcal{C}$ in labelling order, then all $x \in \bigcup_{y \in [n]} \bigcap_{\beta \in B_{\mathcal{C}}(y)} L(\beta)$ are visible vertices in the corresponding network and vice versa.*

Network	**Cover**
Path from node x to the root	A backtrack for x
Visible vertex x	There is a $y \in [n]$ such that $x \in \bigcap_{\beta \in B_{\mathcal{C}}(y)} L(\beta)$.

Table 4.4 Translation of the concepts arising in tree-child networks.

4.3.2 Support trees for tree-child networks

Theorem 17 allows us to provide an alternative proof of a result about support trees in tree-child networks, as follows.

Corollary 11 ([Francis and Moulton, 2018], Theorem 3.3)**.** *A binary tree-child network with k reticulations has 2^k support trees.*

Proof. Since each set in the cover for a tree-child network has a uniquely appearing element, there are no sets containing only reticulations (i.e. no singletons with elements that appear elsewhere, and no pairs in which both elements are repeated). Using the categories above, all sets in such a cover are from Categories (1), (3), or (5).

As a consequence, there are no crowns, which require sets with two reticulations, and each fence can only have length 1, being of the form $a, b \mid b, c$, and containing only one reticulation (b in this case). Furthermore, each repeated integer in the cover (i.e., each reticulation) is in a fence, since it must be part of a pair with a uniquely appearing element (a tree vertex). Thus, the number of fences is the number of reticulations, and each fence has length 1. Therefore, by Theorem 15, there are 2^k support trees. □

Corollary 11 also follows immediately by combining both parts of the following result.

Theorem 19.

(i) The number of spanning trees in a phylogenetic network is the product of all the in-degrees of the reticulation vertices.

(ii) A network is a tree-child network if and only if every spanning tree is also a support tree.

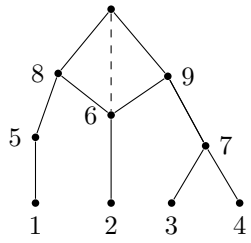
Proof. Part (i): A reticulation vertex x is an integer contained in $k > 1$ subsets of $\mathcal{C}$ (Table 4.1) and a spanning tree is an embedded partition (Table 4.2). Thus, to obtain an embedded partition from a cover, we have to remove $k - 1$ instances of x from $\mathcal{C}$. This can be done in $\binom{k}{k-1} = k$ different ways, and each choice is independent of the others. Since k is also the in-degree for

vertex x, it follows than the number of embedded partitions (spanning trees) is $\prod_x$ *in-degree*(x), where x is a reticulation vertex. If x is a tree vertex, then in-degree$(x) = 1$ and, therefore, it does not contribute to the product.

Part (ii): By Theorem 17, every subset of a tree-child cover has at least one element that is not present in any other subset. This implies that every embedding partition must contain at least one element for each subset; hence, it has the same size as $|\mathcal{C}|$.

To show the forward direction, suppose that N is not a tree-child network. We will show that there must be a spanning tree for N that is not a support tree. If N is not tree-child, then it has at least one vertex that is not visible. Let v be a non-visible vertex that is maximally distant from the root, so that all vertices descended from v are visible. If we delete each arc out of v, then there is a path from the root to each vertex, so N has a spanning tree T. However, in this tree, T has v as a leaf. The tree T is therefore a spanning tree of N and not all its leaves are in X, so T is not a support tree. □

4.4 Normal networks



Normal networks are a subclass of the tree-child networks, with the added constraint that they contain no "shortcuts" [Willson, 2010]. A *shortcut* is an edge (u, v) for which there is an alternative directed path from u to v in the network. The network on the side, for example, is the tree-child network we have seen in the previous section but with the edge $(\rho, 6)$ removed (depicted with a dashed line). That edge was a shortcut because the network has two other paths from ρ to 6, namely $\rho \to 8 \to 6$ and $\rho \to 9 \to 6$. To capture this information in terms of covers, we need a way to record paths in that context. This motivated the definition of backtrack (Definition 18), which requires the labelling order that was defined in Section 1.2.3. The backtrack algorithm identifies a path from the vertex labelled x back to the root, expressing the path in terms of a sequence of sets in the cover. The edges between the parent vertices that correspond with these sets defines the path.

Theorem 20. *Let N be a phylogenetic network with expanding cover $\mathcal{C}$, in labelling order. Then N has a shortcut if and only if there is a backtrack for an $x \in [m]$ that includes at least two subsets containing x.*

Proof. All backtracks β will contain a parent of x, and therefore x will be contained in the first subset of β. For this proof, we will ignore it and prove that there exists another subset of β containing x.

Suppose N has a shortcut. Then there is a vertex x with a non-trivial path from some vertex v to x, and there is also an edge (v, x). The existence of a non-trivial path from v to x means that the cover has a non-trivial backtrack from x, which includes the children of v as a set. However, x is also a child of v, so x is in a set in the backtrack.

Conversely, suppose that the cover contains a backtrack for x that includes a set S containing x. Let v be the label of the parent of S. Then x is a child of v, meaning there is an edge (v, x) in N. However, the backtrack provides a non-trivial path in N from v to x through S. That is, N contains a shortcut. □

Corollary 12. *Let $\mathcal{C}$ be a cover in labelling order for a tree-child network. Then $\mathcal{C}$ is a cover for a normal network if and only if, for all $x \in [m]$, no backtrack for x has at least two subsets that contain x.*

Without loss of generality, in Theorem 20 and Corollary 12, we can assume that x is a reticulation vertex (i.e., a value in $[m]$ that is contained in more than one subset of $\mathcal{C}$), since, by definition, reticulations have in-degree greater than one and thus are the only vertices that can have shortcuts. Using Theorem 20, we can construct an algorithm that removes all the shortcuts from a cover. This implies that, given a tree-child network, we can transform it to a normal network by removing all the shortcuts via Algorithm 9.

Algorithm 9 Remove all shortcuts from a cover

```
Require: Expanding cover C in labelling order
  procedure RemoveShortcuts(C)
      Compute all backtracks for all reticulation vertices
      for backtrack β for reticulation vertex x do
          Remove the first subset of β                ▷ The parent of x
          for s ∈ β do
              if x ∈ s then
                  Remove x from s
```

Network	**Cover**
Shortcut to x	A backtrack β of x that includes a set containing x, ignoring the first subset of β.

Table 4.5 A translation of a shortcut into a feature of the corresponding expanding cover.

4.5 Tree-sibling networks

Tree-sibling networks are also amenable to a description in terms of covers.

Definition 19 ([Cardona et al., 2008a])**.** A tree-sibling network is a network in which every reticulation vertex is a sibling of a tree vertex.

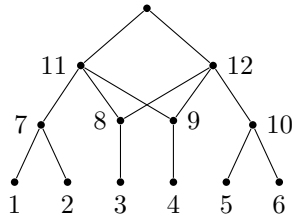


Figure 4.5 A tree-sibling network. Its cover is $1,2 \mid 3 \mid 4 \mid 5,6 \mid \mathbf{7},\underline{8},\underline{9} \mid \underline{8},\underline{9},\mathbf{10} \mid 11,12$. Notice how repeated integers (reticulations, underlined) are in the same sets with non-repeating integers (tree vertices, highlighted in bold).

Theorem 21. *Tree-siblings networks are in bijection with those expanding covers for which every repeated integer lies in at least one set with an integer that appears only once.*

Proof. The statement is a direct translation of the definition of tree-sibling into the language of covers, according to Table 4.1. Reticulation vertices are those that appear more than once in the cover, and vertices are siblings when they appear in the same set in the cover. □

We have already seen a characterization of tree-child networks using covers in Theorem 17. Covers for tree-child networks are those for which every set has a uniquely appearing element. However, there is a close connection between tree-child and tree-sibling networks, which can be captured in a cover description for tree-child networks, as follows.

Theorem 22. *Tree-child networks are in bijection with expanding covers for which, for every repeated element k in $\mathcal{C}$, every subset containing k also contains an integer that appears only once.*

Proof. We will prove that this statement is equivalent to Theorem 17.

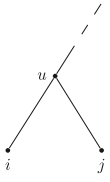
For the forward direction, suppose that a cover satisfies the condition in Theorem 17. If every subset contains a uniquely occurring integer, then all subsets that contain a reticulation will do also do so.

For the backward direction, by assumption, every subset that contains a reticulation vertex has an integer that is not contained in another subset. This implies that all other subsets do not contain a reticulation vertex, and therefore, it contains a tree vertex that is not contained in any other subset (Table 4.1). □

In other words, tree-child networks are networks in which every parent of a reticulation vertex has a tree-vertex as a child. Therefore, we recover the well-known fact that all tree-child networks are tree-sibling networks.

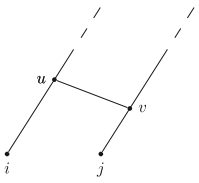
4.6 Orchard networks

Orchard networks are non-degenerate phylogenetic networks defined by the property that they can be reduced to a trivial network (a network consisting of a single vertex) by a series of cherry or reticulated cherry reductions [Erdős et al., 2019; Janssen and Murakami, 2021; van Iersel et al., 2022]. In this work, we restricted our attention to *binary* orchard networks.



cherry

A *cherry* is a pair of leaves that are siblings; a *reticulated cherry* is a pair of leaves, one of which has a reticulate parent and the other is the sibling of that reticulate parent. Cherry reduction involves replacing the cherry with a single vertex. Reticulated cherry reduction involves deleting the arc between the parents of the two leaves and then suppressing degree-2 vertices. By a theorem of [Erdős et al., 2019; Janssen and Murakami, 2021] for orchard networks, the order in which these are performed is not important.



reticulated cherry

To translate this definition into covers, we need to first characterize cherries and reticulated cherries as they are manifested in covers, and then describe the action of such reductions in terms of the cover. The first of these requirements is routine; the second not, as it requires us to augment the cover with its set of leaves. We will describe a test for orchard that reduces an expanding cover to a trivial cover but, along the way, passes through covers that are not expanding.

In covers, a cherry is given by a set consisting of two elements of $[n]$ (the leaves), whereas a reticulated cherry is given by a singleton subset of $[n]$ appearing in position j in the labelling order, and a pair $\{n+j, i\}$ where $i \in [n]$ (summarized in Table 4.7). Here is an example:

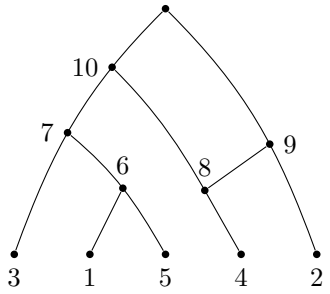


Figure 4.6 An orchard network.

The network above is a labellable network with a cherry and a reticulated cherry. Its cover, in labelling order, is $1,5 \mid 3,6 \mid 4 \mid 2,8 \mid 7,8 \mid 9,10$. The cherry can be identified in the cover as a pair of integers that are a subset of the leafset [5]. In this cover, a cherry is $\{1,5\}$. The reticulated cherry is identified in the cover as a pair of sets: one is a singleton subset of the leafset, in position j; the other is the pair $\{n+j, i\}$ with i in the leafset. In this cover, there is a reticulated cherry consisting of the singleton $\{4\}$ (contained in the leafset), which appears in position 3 in the labelling order, and the pair $\{2,8\}$, noting that $8 = n+3$ and 2 is in the leafset.

4.6.1 The cherry reduction process via covers

The cherry reduction test for orchard networks can be defined efficiently using covers by keeping track of the changing set of leaf labels $\mathcal{L}$ within the algorithm. Identifying a cherry or reticulated cherry in a cover can be done using the translations given in Table 4.7. The process in Algorithm 10 chooses to reduce

a cherry first, if there is one, as it involves fewer checks. In general, the set $\mathcal{C}$ that is redefined during Algorithm 10 may not be an expanding cover, but these processes do nevertheless model the network cherry and reticulated cherry reduction steps, applied to a labellable network.

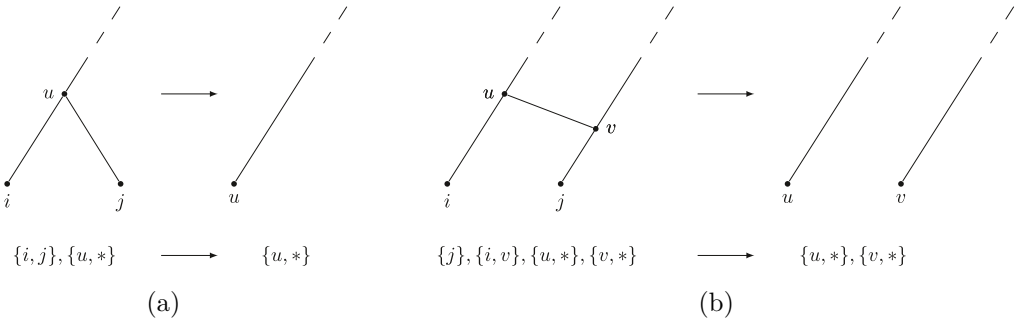


Figure 4.7 Cherry (a) and reticulated cherry (b) reductions and their effects on the covers. Here, $*$ represents other sibling vertices, which could be an empty set.

Algorithm 10 takes a cover $\mathcal{C}$ as input and decides if it corresponds to an orchard network as follows:

1. While $|\mathcal{C}| > 0$ and a reduction is applicable
   (a) If a cherry reduction is possible, apply the reduction shown in Figure 4.7a
   (b) Otherwise, apply a reticulated cherry reduction as shown in Figure 4.7b
2. If $|\mathcal{C}| = 0$ is an orchard network, otherwise it is not

Theorem 23. *Algorithm 10 determines whether the network from the expanding cover $\mathcal{C}$ is orchard.*

Proof. A network is orchard, by definition, if and only if it can be reduced to a trivial network by cherry or reticulated cherry reductions. According to a result of [Erdős et al., 2019; Janssen and Murakami, 2021], the order of such reductions is not important. The procedures in Algorithm 10 exactly reflect the effect on the cover of these operations on the network, as can be seen in Figure 4.7. □

To illustrate the cherry reduction process visually, let's take again the network shown in Figure 4.6, whose cover is $\mathcal{C} = 1, 5 \mid 3, 6 \mid 4 \mid 2, 8 \mid 7, 8 \mid 9, 10$, and reduce it using Algorithm 10. The process is described in Table 4.6.

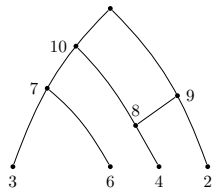
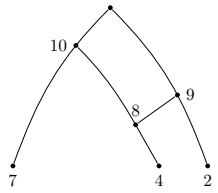
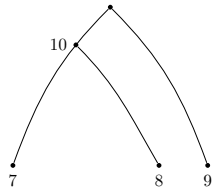
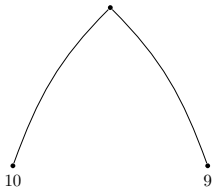
<table>
<tr><td colspan="3">Cherry reduction of the cover $\mathcal{C}$, following Algorithm 10.</td></tr>
<tr><td colspan="3">$\mathcal{C} = \{\{1,5\}_6, \{3,6\}_7, \{4\}_8, \{2,8\}_9, \{7,8\}_{10}, \{9,10\}_\rho\}$
$\mathcal{L} = \{1,2,3,4,5\}$</td></tr>
<tr><td rowspan="2">1</td><td colspan="2">$\mathcal{C}$ contains the cherry $\{1,5\}_6$ (and the reticulated cherry, $\{4\}_8, \{2,8\}_9$). We reduce the cherry.</td></tr>
<tr><td>$\mathcal{C}_1 = \mathcal{C} \setminus \{\{1,5\}_6\}$
$= \{\{3,6\}_7, \{4\}_8, \{2,8\}_9, \{7,8\}_{10}, \{9,10\}_\rho\}$
$\mathcal{L}_1 = (\mathcal{L} \setminus \{1,5\}) \cup \{6\}$
$= \{2,3,4,6\}$</td><td> 
</td></tr>
<tr><td rowspan="2">2</td><td colspan="2">$\mathcal{C}_1$ contains cherry $\{3,6\}_7$ (and the reticulated cherry $\{4\}_8, \{2,8\}_9$). We reduce the cherry.</td></tr>
<tr><td>$\mathcal{C}_2 = \mathcal{C}_1 \setminus \{\{3,6\}_7\}$
$= \{\{4\}_8, \{2,8\}_9, \{7,8\}_{10}, \{9,10\}_\rho\}$
$\mathcal{L}_2 = (\mathcal{L}_1 \setminus \{3,6\}) \cup \{7\}$
$= \{2,4,7\}$</td><td> 
</td></tr>
<tr><td rowspan="2">3</td><td colspan="2">$\mathcal{C}_2$ contains no cherry, but contains the reticulated cherry $\{4\}_8, \{2,8\}_9$, which we reduce.</td></tr>
<tr><td>$\mathcal{C}_3 = \mathcal{C}_2 \setminus \{\{4\}_8, \{2,8\}_9\}$
$= \{\{7,8\}_{10}, \{9,10\}_\rho\}$
$\mathcal{L}_3 = (\mathcal{L}_2 \setminus \{2,4\}) \cup \{8,9\}$
$= \{7,8,9\}$.</td><td> 
</td></tr>
<tr><td rowspan="2">4</td><td colspan="2">$\mathcal{C}_3$ contains the cherry $\{7,8\}_{10}$, which we reduce.</td></tr>
<tr><td>$\mathcal{C}_4 = \mathcal{C}_3 \setminus \{7,8\}_{10}$
$= \{\{9,10\}_\rho\}$
$\mathcal{L}_4 = (\mathcal{L}_3 \setminus \{7,8\}) \cup \{10\}$
$= \{9,10\}$.</td><td> 
</td></tr>
<tr><td rowspan="2">5</td><td colspan="2">$\mathcal{C}_4$ contains (only) the cherry $\{9,10\}_\rho$, which we reduce.</td></tr>
<tr><td>$\mathcal{C}_5 = \mathcal{C}_4 \setminus \{9,10\}_\rho = \emptyset$, which means the algorithm ends.</td><td></td></tr>
</table>

Table 4.6 Cherry reduction algorithm (Algorithm 10), with the effects of reduction on the network shown at right (for illustration only).

Algorithm 10 Test whether the expanding cover $\mathcal{C}$ corresponds to an orchard network

Require: Expanding cover $\mathcal{C}$ in labelling order

procedure IsOrchard($\mathcal{C}$)
  $n \leftarrow ||\mathcal{C}|| - |\mathcal{C}| + 1$
  $\mathcal{L} \leftarrow [n]$
  $reduced \leftarrow true$
  while $reduced = true$ **and** $|\mathcal{C}| > 0$ **do**
    $reduced \leftarrow false$
    if there is a set of form $\{a, b\}_j \in \mathcal{C}$ with $a, b \in \mathcal{L}$ **then**
      $\mathcal{C} \leftarrow \mathcal{C} \setminus \{a, b\}$ ▷ Cherry reduction
      $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{a, b\}) \cup \{j\}$
      $reduced \leftarrow true$
    else
      if there is a set of form $\{a\}_j$ in $\mathcal{C}$ and a set of form $\{j, b\} \in \mathcal{C}$, with $a, b \in \mathcal{L}$ **then**
        $\mathcal{C} \leftarrow \mathcal{C} \setminus \{\{a\}, \{j, b\}\}$ ▷ Reticulated cherry reduction
        $\mathcal{L} \leftarrow (\mathcal{L} \setminus \{a, b\}) \cup \{j, k\}$
        $reduced \leftarrow true$
  if $\mathcal{C} = \emptyset$ **then**
    return "$\mathcal{C}$ is orchard"
  else
    return "$\mathcal{C}$ is not orchard"

Network	Cover
Cherry	A set consisting of two elements of $[n]$
Reticulated cherry	A singleton subset of $[n]$ appearing in position j in the labelling order, and a pair $\{n + j, i\}$ where $i \in [n]$

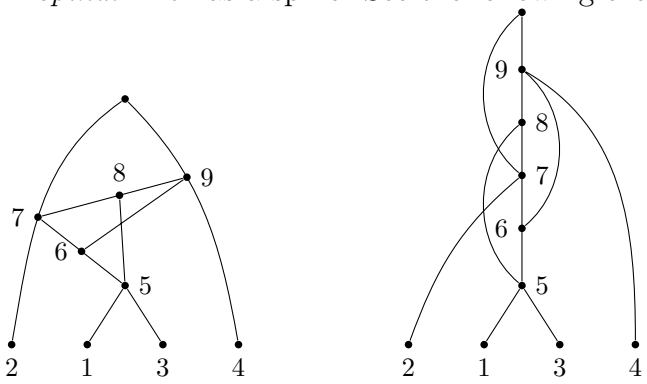
Table 4.7 A translation of features that are relevant to orchard networks into features of the corresponding expanding cover.

4.7 A new class of networks detected through the lens of covers

We have used covers to describe several classes of phylogenetic networks. However, the encoding into covers also creates the opportunity to define new classes of networks that correspond to particular features of covers. Such classes might currently have little direct utility for application to phylogenetics, but they may have an indirect value in that algorithms and methods using covers may involve such classes in passing. We introduce one such class as an example of this opportunity.

Recall that the definition of an expanding cover has two criteria (Definition 7). The first is that elements of the leafset $[n]$ are not repeated, and the second ensures that the labelling algorithm is well-defined by requiring at least i subsets of $[n+i-1]$ to be in the cover.

If a cover contains *exactly* i subsets of $[n+i-1]$, it has a strong consequence for the network, as follows. We define a *spine* in a phylogenetic network to be a directed path from a leaf to the root that traverses all non-leaf vertices, and we call a network *spinal* if it has a spine. See the following example:



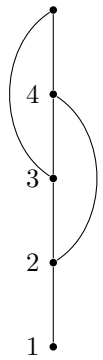
The above is a spinal network with cover $1,3 \mid 5 \mid 2,6 \mid 5,7 \mid 4,6,8 \mid 7,9$. Note that $n = 4$ and the cover has one set in [4], two in [5], three in [6], four in [7], five in [8], and six in [9]. There is a path from the elements of the set that is in [4], namely 1 and 3, to the root, and this path traverses every non-leaf vertex. Observe that this path is in labelling sequence. The spine is particularly clear when the network is drawn as shown on the right.

Theorem 24. *A network is spinal if and only if its cover $\mathcal{C}$ has exactly i subsets of $[n+i-1]$, for each $i = 1, \ldots, |\mathcal{C}|$.*

Proof. We prove the reverse direction first. Suppose that the cover $\mathcal{C}$ has exactly i subsets of $[n+i-1]$, for each $i = 1, \ldots, |\mathcal{C}|$, and consider its labelling order. The first set in the labelling order is the unique set that is contained in $[n]$, and its label is $n+1$. For each $i = 1, \ldots, |\mathcal{C}|-1$, the i-th set in the labelling order is contained in $[n+i-1]$, according to the expanding property, but is not contained in $[n+i-2]$, according to our assumption about $\mathcal{C}$. Therefore, it must contain the integer $n+i-1$. The i-th set in the labelling order has label $n+i$, which means that $n+i$ is a parent of $n+i-1$. Since this holds for each $i > 1$, this determines a path from a leaf (labelled by an element of the first set in the labelling order) through every vertex with label $n+1$ to $n+|\mathcal{C}|-1$, and the last set, containing $||\mathcal{C}|| = n+|\mathcal{C}|-1$, has the root as parent. Thus the network is spinal.

We now prove the forward direction. Suppose that N is spinal with cover $\mathcal{C}$. Being spinal means that N has a path of length $|\mathcal{C}|$ from a leaf to the root. This means that there is a backtrack of a leaf that has length $|\mathcal{C}|-1$. That is, a sequence of sets from the cover such that the label of one set (from the labelling order on $\mathcal{C}$) is an element of the next set in the backtrack sequence. Because the label of a set in the cover is strictly greater than all the elements of the set, the maximal elements of the sets in a backtrack are strictly increasing.

Now consider the backtrack arising from the spine (the path from a leaf to the root traversing all non-leaf vertices). The leaf at the base of the spine must be in a set contained in $[n]$; otherwise, there would be no path from it to the vertex labelled $n+1$. Therefore, the first set in the backtrack contains $n+1$ as its maximal element because that is the parent label for the set containing the initial leaf. The spine has $|\mathcal{C}|-1$ vertices in it, including the initial leaf, because it includes all except $n-1$ of the vertices in the network (the network has $||\mathcal{C}||+1 = |\mathcal{C}|+n$ vertices in total). Therefore, the backtrack for the initial leaf has $|\mathcal{C}|-1$ sets. The maximal elements of these $|\mathcal{C}|-1$ sets are strictly increasing, and run from $n+1$ to $m = ||\mathcal{C}|| = |\mathcal{C}|+n-1 = n+(|\mathcal{C}|-1)$. This forces each set in the backtrack to have a distinct maximal element. Put together with the set containing the initial leaf, which is a subset of $[n]$, this means that there are exactly i subsets of $[n+i-1]$, for each $i = 1, \ldots, |\mathcal{C}|$, as required. □



In light of Theorem 24, we say that a cover $\mathcal{C}$ is *spinal* if it contains exactly i subsets of $[n+i-1]$, for each $i = 1, \ldots, |\mathcal{C}|$. Spinal networks have some non-trivial intersections with other classes; for example, the spinal network shown on the side, with cover $1 \mid 2 \mid 2,3 \mid 3,4$, is not a tree-child, tree-sibling, or orchard network. It can, however, be shown that the class of spinal networks lies within the intersection of the labellable and tree-based classes of networks.

Network	**Cover**
Spine	Exactly i subsets of $[n+i-1]$, for each i

Table 4.8 A translation of the feature of spinal networks into a feature of the corresponding expanding cover.

4.8 Discussion

Sometimes a relatively small shift in perspective can open up new possibilities in surprising ways. What seems like a fairly straightforward idea in a paper by Diaconis and Holmes (the idea that rooted binary phylogenetic trees correspond to perfect matchings [Diaconis and Holmes, 1998]), itself building on an elegant but simple way to label internal vertices [Erdős and Székely, 1989], was loosened slightly to yield a correspondence between phylogenetic forests and all partitions of finite sets, as well as a raft of interesting questions in semigroup theory [Francis and Jarvis, 2022]. This subtle twist of an idea, like something from a Philip Pullman novel [Pullman, 2015], seems to have opened up further opportunities that, with a further gentle twist, have opened a new canvas on which to draw phylogenetic networks [Francis et al., 2023]. Capturing the features that define different network classes on this canvas provided the underlying motivation for this work.

Many core features discussed in the context of networks, such as reticulations, paths, cherries, siblings, and so on, have been translated into the language of covers; a summary is given in Table 4.9. These translations of features have been

necessary for characterizing several important classes of phylogenetic network in the language of covers. This includes some of the most prominent classes, including normal, tree-child, tree-sibling, orchard, and tree-based networks (relationships among the classes, determined by properties of their covers, are represented in Figure 4.8). However there are many classes, each of which is important for its own reasons, and this list is not complete. Some classes that have been omitted in the present work might be difficult to define with covers (for instance, level-k networks or HGT networks), whereas others might just be a matter of following through with the first steps we have taken here (for example, reticulation-visible networks, and non-binary orchard networks).

Network N	**Cover** $\mathcal{C}$
Non-root vertex	An integer in $[m]$
Leaf	An integer in $[n]$
Tree vertex	An integer contained in just one subset
Reticulation vertex	An integer contained in more than one subset
In-degree of x	The number of subsets that contain x
Out-degree of x	Size of the subset with label x in the labelling order
Parents of x	All the subsets that contain x
Siblings of x	All the other integers contained in the subsets that contain x
Children of x	The subset with label x in the labelling order
Spanning tree	A partition embedded in $\mathcal{C}$
Support tree	A full embedding of a partition in $\mathcal{C}$
Crown	Collection of sets $a_0, a_1 \mid a_1, a_2 \mid \cdots \mid a_{t-1}, a_t \mid a_t, a_0$.
Fence	Collection of sets $a_0, a_1 \mid a_1, a_2 \mid \cdots \mid a_t, a_{t+1}$, with $a_0 \neq a_{t+1}$ both unique.
Path from x to ρ	A backtrack for x
Visible x	There is a $y \in [n]$ such that $x \in \bigcap_{\beta \in B_{\mathcal{C}}(y)} L(\beta)$.
Shortcut to x	A backtrack β of x that includes a set containing x, ignoring the first subset of β.
Cherry	A set consisting of two elements of $[n]$
Reticulated cherry	A subset $\{a\}_k$ of $[n]$, and a pair $\{k, b\}$ with $b \in [n]$
Spine	Exactly i subsets of $[n + i - 1]$, for each i

Table 4.9 A translation of the features of a labellable network with n leaves and m non-root vertices into the features of the corresponding expanding cover.

Defining a language is not the goal, however, despite it being a necessary step. The goal is to be able to efficiently work with phylogenetic networks —

computationally, algorithmically, and mathematically — in order to establish robust methods of inference for networks that will eventually be of practical use for biological researchers. To that end, encoding various classes of phylogenetic networks in terms of expanding covers provides an opportunity to make computation more effective and allow their structure to be seen more clearly.

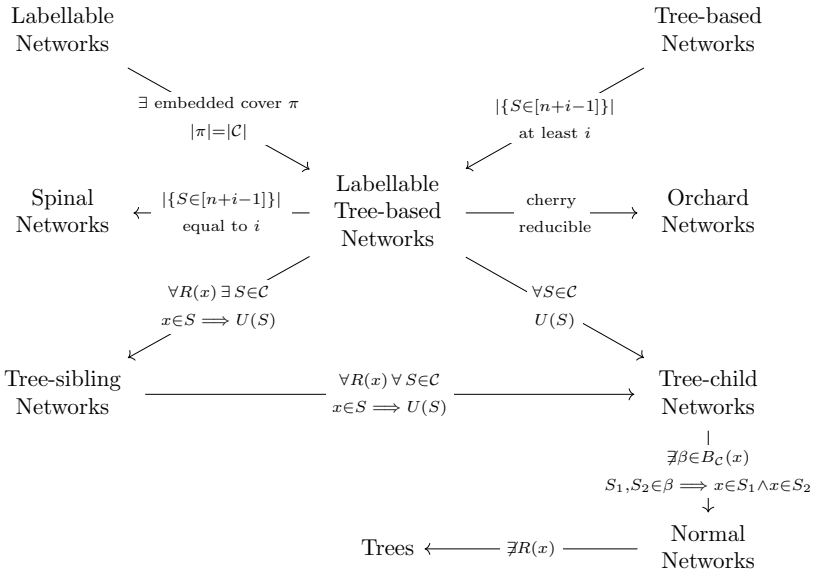


Figure 4.8 A diagram showing the hierarchy of networks. The nodes are classes of networks, the arrows represent inclusion and the labels indicate which axiom we add to obtain that class. $R(x)$ means "x is repeated in more than one subset" and $U(S)$ means "S contains an unique integer".

Bibliography

Aicardi, F., Arcis, D., and Juyumaya, J. (2023). Brauer and Jones tied monoids. *Journal of Pure and Applied Algebra*, 227(1):107161.

Bapteste, E., van Iersel, L., Janke, A., Kelchner, S., Kelk, S., McInerney, J., Morrison, D., Nakhleh, L., Steel, M., Stougie, L., and Whitfield, J. (2013). Networks: Expanding evolutionary thinking. *Trends in Genetics*, 29:439–441.

Beaudry, M. (1988). Membership testing in commutative transformation semigroups. *Information and Computation*, 79(1):84–93.

Björner, A. and Brenti, F. (2005). *Combinatorics of Coxeter groups*, volume 231. Springer.

Bon, M., Vernizzi, G., Orland, H., and Zee, A. (2008). Topological classification of RNA structures. *Journal of molecular biology*, 379(4):900–911.

Brauer, R. (1937). On algebras which are connected with the semisimple continuous groups. *Annals of Mathematics*, pages 857–872.

Cardona, G., Llabrés, M., Rosselló, F., and Valiente, G. (2008a). A distance metric for a class of tree-sibling phylogenetic networks. *Bioinformatics*, 24(13):1481–1488.

Cardona, G., Rosselló, F., and Valiente, G. (2008b). Comparison of tree-child phylogenetic networks. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 6(4):552–569.

Chlouveraki, M. and Pouchin, G. (2017). Representation theory and an isomorphism theorem for the framisation of the Temperley–Lieb algebra. *Mathematische Zeitschrift*, 285(3-4):1357–1380.

Clavel, M., Durán, F., Eker, S., Escobar, S., Lincoln, P., Martı-Oliet, N., Meseguer, J., Rubio, R., and Talcott, C. (2022). *Maude Manual (Version 3.2.1)*. The Maude System (https://maude.cs.illinois.edu/).

Diaconis, P. W. and Holmes, S. P. (1998). Matchings and phylogenetic trees. *Proceedings of the National Academy of Sciences*, 95(25):14600–14602.

Dolinka, I. and East, J. (2018). Twisted Brauer monoids. *Proceedings of the Royal Society of Edinburgh Section A: Mathematics*, 148(4):731–750.

Dolinka, I., East, J., and Gray, R. D. (2017). Motzkin monoids and partial Brauer monoids. *Journal of Algebra*, 471:251–298.

East, J., Egri-Nagy, A., Mitchell, J. D., and Péresse, Y. (2019). Computing finite semigroups. *Journal of Symbolic Computation*, 92:110–155.

Egri-Nagy, A. and Nehaniv, C. L. (2008). Hierarchical coordinate systems for understanding complexity and its evolution, with applications to genetic regulatory networks. *Artificial Life*, 14(3):299–312.

Erdős, P. L., Semple, C., and Steel, M. (2019). A class of phylogenetic networks reconstructable from ancestral profiles. *Mathematical Biosciences*, 313:33–40.

Erdős, P. L. and Székely, L. (1989). Applications of antilexicographic order. I. An enumerative theory of trees. *Advances in Applied Mathematics*, 10(4):488–496.

Erickson, J. (2023). *Algorithms*. Independently published (https://jeffe.cs.illinois.edu/teaching/algorithms/).

Ernst, D. C., Hastings, M. G., and Salmon, S. K. (2016). Factorization of Temperley–Lieb diagrams. *Involve, a Journal of Mathematics*, 10(1):89–108.

Francis, A., Huber, K. T., Moulton, V., and Wu, T. (2023). Encoding and ordering X-cactuses. *Advances in Applied Mathematics*, 142:102414.

Francis, A. and Jarvis, P. D. (2022). Brauer and partition diagram models for phylogenetic trees and forests. *Proceedings of the Royal Society A*, 478(2262):20220044.

Francis, A., Marchei, D., and Steel, M. (2024). Phylogenetic network classes through the lens of expanding covers. *Journal of Mathematical Biology*, 88(5):58.

Francis, A. and Moulton, V. (2018). Identifiability of tree-child phylogenetic networks under a probabilistic recombination-mutation model of evolution. *Journal of Theoretical Biology*, 446:160–167.

Francis, A. and Steel, M. (2015). Which phylogenetic networks are merely trees with additional arcs? *Systematic Biology*, 64(5):768–777.

Francis, A. and Steel, M. (2023). Labellable phylogenetic networks. *Bulletin of Mathematical Biology*, 85(6):46.

Friend, E. H. (1956). Sorting on electronic computer systems. *Journal of the ACM (JACM)*, 3(3):134–168.

Froidure, V. and Pin, J.-E. (1997). Algorithms for computing finite semigroups. In *Foundations of Computational Mathematics: Selected Papers of a Conference Held at Rio de Janeiro, January 1997*, pages 112–126. Springer.

Garrett, J., Jonoska, N., Kim, H., and Saito, M. (2019). Algebraic systems motivated by DNA origami. In Ćirić, M., Droste, M., and Pin, J.-É., editors, *Algebraic Informatics*, pages 164–176, Cham. Springer International Publishing.

Garrett, J., Jonoska, N., Kim, H., and Saito, M. (2021). DNA origami words, graphical structures and their rewriting systems. *Natural Computing*, 20(2):217–231.

Giegerich, R., Voß, B., and Rehmsmeier, M. (2004). Abstract shapes of RNA. *Nucleic acids research*, 32(16):4843–4851.

Gilbert, W. (1986). Origin of life: The RNA world. *Nature*, 319(6055):618–618.

Ginzburg, A. and Yoeli, M. (1965). Products of automata and the problem of covering. *Transactions of the American Mathematical Society*, 116:253–266.

Hayamizu, M. (2021). A structure theorem for rooted binary phylogenetic networks and its implications for tree-based networks. *SIAM Journal on Discrete Mathematics*, 35(4):2490–2516.

Huson, D. H., Rupp, R., and Scornavacca, C. (2010). *Phylogenetic Networks: Concepts, algorithms and applications*. University Press, Cambridge, UK.

Janssen, R. and Murakami, Y. (2021). On cherry-picking and network containment. *Theoretical Computer Science*, 856:121–150.

Jetten, L. (2015). *Characterising tree-based phylogenetic networks (Karakterisatie van fylogenetische netwerken die een boom als basis hebben)*. PhD thesis, Delft University of Technology.

Jones, V. F. R. (1983). Index for subfactors. *Inventiones mathematicae*, 72(1):1–25.

Kauffman, L. and Magarshak, Y. (1995). Vassiliev knot invariants and the structure of RNA folding. *Knots and Applications*.

Kleinberg, J. and Tardos, E. (2006). *Algorithm design*. Pearson Education India.

Kong, S., Pons, J. C., Kubatko, L., and Wicke, K. (2022). Classes of explicit phylogenetic networks and their biological and mathematical significance. *Journal of Mathematical Biology*, 84(6):1–44.

Krohn, K. and Rhodes, J. (1965). Algebraic theory of machines. i. prime decomposition theorem for finite semigroups and machines. *Transactions of the American Mathematical Society*, 116:450–464.

Kudryavtseva, G. and Mazorchuk, V. (2006). On presentations of Brauer-type monoids. *Central European Journal of Mathematics*, 4(3):413–434.

Maestri, S. and Merelli, E. (2019). Process calculi may reveal the equivalence lying at the heart of RNA and proteins. *Scientific reports*, 9(1):1–9.

Marchei, D. and Merelli, E. (2022). RNA secondary structure factorization in prime tangles. *BMC bioinformatics*, 23(6):1–18.

Markov, A. (1951). Impossibility of certain algorithms in the theory of associative systems. *Doklady Akademii Nauk SSSR*, 77(1).

Meseguer, J. and Montanari, U. (1990). Petri nets are monoids. *Information and computation*, 88(2):105–155.

Moore, E. F. (1959). The shortest path through a maze. In *Proc. of the International Symposium on the Theory of Switching - Part II*, pages 285–292. Harvard University Press.

Nyberg-Brodda, C.-F. (2022). The Adian-Rabin Theorem – An English translation. arXiv:2208.08560 (https://arxiv.org/pdf/2208.08560.pdf).

OEIS Foundation Inc. (2024). The On-Line Encyclopedia of Integer Sequences. Published electronically at http://oeis.org.

Pin, J.-E. (1995). Finite semigroups and recognizable languages: an introduction. *NATO Advanced Study Institute Semigroups, Formal Languages and Groups, J. Fountain (ed.), Kluwer academic publishers*, pages 1–32.

Pons, J. C., Semple, C., and Steel, M. (2019). Tree-based networks: characterisations, metrics, and support trees. *Journal of Mathematical Biology*, 78:899–918.

Pullman, P. (2015). *The Subtle Knife: His Dark Materials*. Random House.

Quadrini, M., Merelli, E., and Piergallini, R. (2019a). Loop grammars to identify RNA Structural Patterns. In *Bioinformatics*, pages 302–309.

Quadrini, M., Tesei, L., and Merelli, E. (2019b). An algebraic language for RNA pseudoknots comparison. *BMC bioinformatics*, 20(4):1–18.

Ram, A. (1995). Characters of Brauer's centralizer algebras. *Pacific journal of Mathematics*, 169(1):173–200.

Reeder, J. and Giegerich, R. (2004). Design, implementation and evaluation of a practical pseudoknot folding algorithm based on thermodynamics. *BMC Bioinformatics*, 5(1):104.

Reidys, C. M., Huang, F. W., Andersen, J. E., Penner, R. C., Stadler, P. F., and Nebel, M. E. (2011). Topology and prediction of RNA pseudoknots. *Bioinformatics*, 27(8):1076–1085.

Reidys, C. M. and Wang, R. R. (2010). Shapes of RNA pseudoknot structures. *Journal of Computational Biology*, 17(11):1575–1590.

Rothemund, P. W. K. (2006). Folding DNA to create nanoscale shapes and patterns. *Nature*, 440(7082):297–302.

Schröder, E. (1870). Vier combinatorische probleme. *Z. Math. Phys*, 15:361–376.

Stanley, R. P. (1984). On the number of reduced decompositions of elements of Coxeter groups. *European Journal of Combinatorics*, 5(4):359–372.

Temperley, H. N. and Lieb, E. H. (1971). Relations between the 'percolation' and 'colouring' problem and other graph-theoretical problems associated with regular planar lattices: some exact results for the 'percolation' problem. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, 322(1549):251–280.

van Iersel, L., Janssen, R., Jones, M., and Murakami, Y. (2022). Orchard networks are trees with additional horizontal arcs. *Bulletin of Mathematical Biology*, 84(8):1–21.

Vernizzi, G., Orland, H., and Zee, A. (2016). Classification and predictions of RNA pseudoknots based on topological invariants. *Physical Review E*, 94(4):042410.

Willson, S. J. (2010). Properties of normal phylogenetic networks. *Bulletin of Mathematical Biology*, 72:340–358.

Zhang, L. (2016). On tree-based phylogenetic networks. *Journal of Computational Biology*, 23(7):553–565.

Zhao, Q., Zhao, Z., Fan, X., Yuan, Z., Mao, Q., and Yao, Y. (2021). Review of machine learning methods for RNA secondary structure prediction. *PLoS computational biology*, 17(8):e1009291.

Zuse, K. (1946). Plankalkül. Konrad Zuse Internet Archive (http://zuse.zib.de).

Appendix A

Testing the Assumptions for $\mathcal{B}_N$

Here we present our methodology for empirically testing Assumption 1 and Assumption 2. The interested reader can find the Python code in the GitHub repository https://github.com/DanieleMarchei/BrauerMonoidFactorization.

A.1 Preparation

To test the two assumptions we need a database of minimal factorizations. We cannot use Algorithm 6, as it would be circular reasoning. This is because its correctness relies on them being true (they were used in Section 2.4.2 to partially apply τ and in Section 2.4.4 to argue that ℓ_P is independent of the factorization). Thus, we decided to create this database by exploring the right Cayley Graph of $\mathcal{B}_N$ using a Breadth First Search (BFS)[1] starting from the identity tangle. The BFS algorithm has the nice property of always returning the shortest path in a graph (which will be a minimal factorization in our case) and by instructing it to first explore the edges labelled with a U-prime, and then all edges labelled with a T-prime, every time we reach an unexplored tangle we know we have obtained a minimal factorization that uses the least

[1] According to [Erickson, 2023], the BFS algorithm was first proposed in [Zuse, 1946] and not in [Moore, 1959], as it is often attributed.

amount of T-primes. Due to time and memory space constraints, we ran this procedure up to $\mathcal{B}_8$.

A.2 Assumption 1

Assumption 1 stated that every factorization in the Brauer monoid can be reduced to a minimal one by a sequence of delete, braid and swap rules. In other words, we do not need to increase the factorization length in order to find a shorter one.

To test this empirically, we implemented the axioms for $\mathcal{B}_N$ (Table 1.1) as a Term Rewriting System (TRS) using the Maude System [Clavel et al., 2022]. The approach is similar to what described in Section 2.3.3.

The actual test for the Assumption 1 is performed as follows, all random choices are done uniformly:

1. Pick a random integer $k \in [2, s\frac{N(N-1)}{2}]$ and sample without replacement k prime tangles. By concatenating them we will obtain a random factorization F for a tangle $X \in \mathcal{B}_N$:
    - s is a scale factor that parametrizes the maximum length possible that can be generated;
    - If the factorization for X stored in the database has the same length of F, then repeat step 1. We only want to test non-minimal factorizations.
2. Get the minimal factorization F^* for X in the database;
3. Reduce F using the TRS, obtaining $\hat{F}$;
4. If $|\hat{F}| \neq |F^*|$, then we have found a counterexample, otherwise repeat from step 1.

To avoid spending too much time on this procedure, we keep track of the tangles generated, effectively testing the assumption once for each tangle. Since the probability of generating a tangle we haven't already generated decreases each time we find a new one, we implemented a "patience" counter that is decreased

N	2	3	4	5	6
$\lvert\mathcal{B}_N\rvert$	3	15	105	945	10395
n. tangles tested	3	15	105	942	10043

Table A.1 Test table for Assumption 1. We ran the test up to $\mathcal{B}_6$ and, among all tangles tested, we never found a counterexample.

each time we do not generate a new tangle. The procedure stops when the patience reaches to zero or all tangles are tested. In our test we set $s = 2$ and the patience counter to 2000000. Because this procedure takes a long time to terminate, we ran it up to $N = 6$ with the results shown in Table A.1. We never found a counterexample.

A possible way for increasing the maximum $\mathcal{B}_N$ testable would be to find an algorithm for sampling factorizations such that their corresponding tangles are uniformly distributed, in fact, the naive procedure we used does not generate tangles uniformly. We generated $10 \times |\mathcal{B}_N|$ random factorizations for all $N \leq 7$ (due to memory and time constraints) and found that the procedure has a bias towards tangles containing U-primes; probably because they do not have an inverse, therefore they cannot be completely removed from a factorization, unlike T-primes. Furthermore, we suspect that there is also a bias towards tangles with a large number of equivalent factorizations because, intuitively, they would be more probable to be generated. We do not have a way to calculate the number of equal factorizations of fixed length a tangle has, so we cannot test this hypothesis.

Although we believe in the validity of this assumption, and our empirical test partially supports that, there is still a chance for the existence of a non-minimal factorization upon which no delete rule can be applied after we have exhausted every possible braid and swap rules applicable. This would imply that we would need to increase the length of the factorization in order to find another one capable of being reduced further, which would invalidate the proof for Theorem 7. A possible technique we could use to find an actual proof for this assumption was presented in [Garrett et al., 2021, Lemma 1], which proves the same statement for the Jones monoid. We leave the exploration of this technique as a future research direction.

A.3 Assumption 2

Assumption 2 stated that if a tangle X has k crossings, then there exists a minimal factorization F with exactly k T-primes and no other factorization with fewer T-primes exists. This assumption is easier and faster to check:

1. pick a tangle X with minimal factorization F from the database;
   - by construction, F will have the minimal amount of T-primes.
2. count the number of crossings c in X;
3. if $c \neq \#T(F)$ then we have found a counterexample, otherwise repeat from step 1.

We ran this procedure up to $N = 8$ and never found a counterexample.

For S_N and $\mathcal{J}_N$ this assumption is trivial, but not for $\mathcal{B}_N$, since there are tangles with a minimal factorization whose number of T-primes does not correspond to the number of crossings. For example:

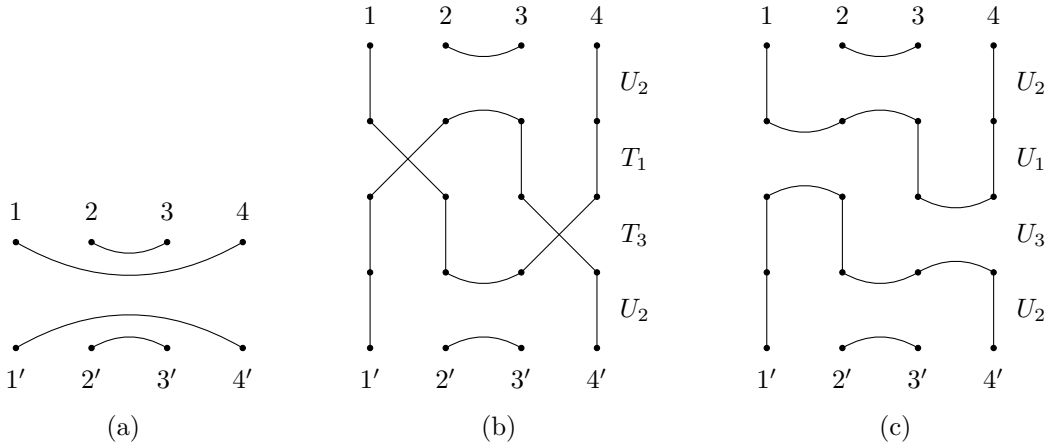
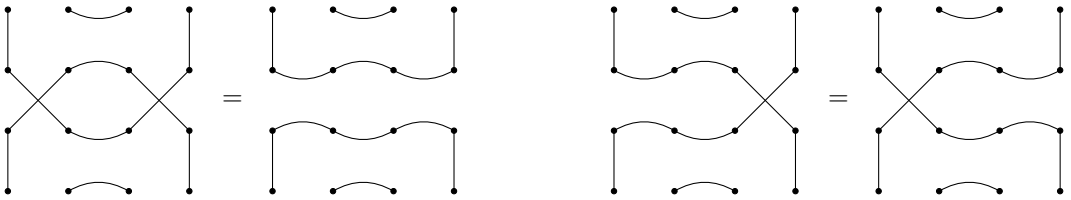


Figure A.1 A tangle along with two of its minimal factorizations having different amount of T-primes.

The tangle shown in Figure A.1a has zero crossings and can be factorized as $U_2T_1T_3U_2$, using two T-primes, or $U_2U_1U_3U_2$, using zero T-primes. It is worth noting that $U_2T_1T_3U_2$ cannot be rewritten as $U_2U_1U_3U_2$ without expanding it, in fact:

$$U_2T_1T_3U_2 = U_2T_1T_2T_2T_3U_2 = U_2U_1T_2T_3U_2 = U_2U_1U_3U_2$$

If Assumption 2 was not true, then there could be a tangle with more T-primes than crossings, which would imply that $\#T(e)$, for some edge e, cannot be computed by counting its crossings, invalidating the proposition that ℓ_P is independent of the factorization. During research, we tried to prove Assumption 2 by augmenting the axioms for $\mathcal{B}_N$ with the following ones:



(a) Annihilation axiom
$U_{i+1}T_iT_{i+2}U_{i+1} = U_{i+1}U_iU_{i+2}U_{i+1}$

(b) Slide axiom
$U_{i+1}U_iT_{i+2}U_{i+1} = U_{i+1}T_iU_{i+2}U_{i+1}$

Figure A.2 Two more axioms we tried to add in order to prove Assumption 2.

The hypothesis was that the existing axioms plus the two above would be sufficient to reduce any minimal factorization to another one with the least amount of T-primes. However, we did not succeed in proving this hypothesis, so we leave it as a future research direction.