NON-CROSSING TREE PARTITION COMPS 2021

ELIOT HUH[A], MEI KNUDSON[B], ALEC WANG[C], JAMES YAN[D]
[A]HUHE@CARLETON.EDU
[B]KNUDSONM2@CARLETON.EDU
[C]WANGA3@CARLETON.EDU
[D]YANJ@CARLETON.EDU

*ADVISED BY ALEXANDER GARVER; AGARVER@CARLETON.EDU

Abstract. Our research is focused on noncrossing tree partitions (NTPs), which are partitions of the interior vertices of a tree that can be represented by noncrossing curves connecting interior vertices. Given an NTP, we can apply an operation known as the Kreweras complement to produce a new NTP. However, there does not exist any algorithmic procedure for performing this operation. Our primary focus has been to develop and prove the validity of such an algorithm. We also discuss the connections of our work to quiver representation theory.

1. Introduction

The purpose of this work is to investigate properties of the Kreweras complement, an operation that can be performed on noncrossing tree partitions. We begin by defining noncrossing tree partitions, which are noncrossing sets of curves superimposed upon a tree embedded within a disk. The Kreweras complement is a bijective mapping which, when applied to a noncrossing tree partition, produces a new noncrossing tree partition.

Among the algebraic properties of the Kreweras complement is the notion of a Kreweras orbit. Central questions of this research include how one can determine if two noncrossing tree partitions are in the same orbit, and what the orbit sizes are for various families of trees. To this end, we propose an algorithm for the construction of Kreweras complements, which could potentially be implemented as a computational tool to assist in obtaining enumerative results.

Furthermore, we examine connections between Kreweras complements and quiver representation theory, with particular attention paid to Auslander-Reiten translations. We propose a conjecture relating the action of the Auslander-Reiten translation τ on representations to the action of Kreweras complementation on noncrossing tree partitions.

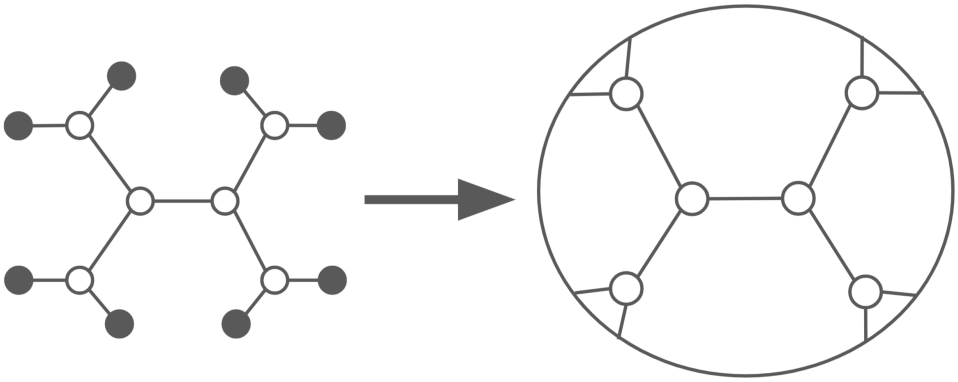
2. Non-Crossing Tree Partitions

In this section, we will review basic definitions regarding tree partitions so as to lay the foundation for understanding the algebraic and enumerative properties of non-crossing tree partitions. We begin by developing the definition of a non-crossing partition.

Definition 2.1. A *set partition* of $[n] = \{1, 2, \ldots, n\}$ is a collection of blocks $\mathbf{B} = \{B_1, \ldots, B_k\}$ of disjoint subsets of $[n]$ where $\bigcup_{i=1}^{n} B_i = [n]$.

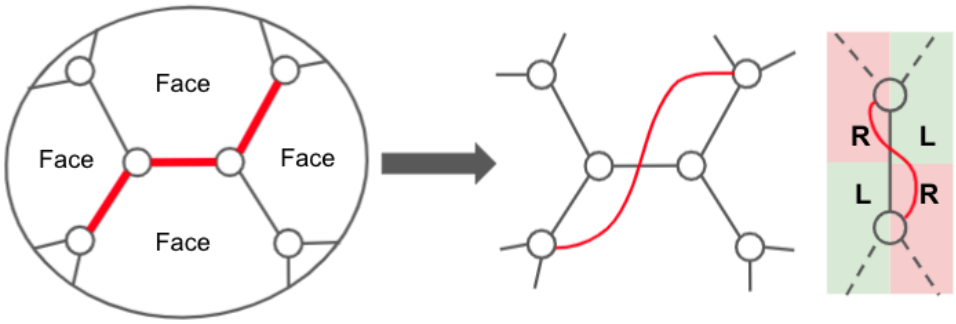
Definition 2.2. A set partition is *non-crossing* if there does not exist $1 \leq i < j < k < l \leq n$ such that i, k are in one block and j, l are in another block of $\mathbf{B}$.

For the context of this paper, we wish to consider trees embedded in a disk D^2 such that only its leaves are on the boundary, and its interior vertices have degree ≥ 3, such as the one in Figure x.



Definition 2.3. A *segment* of a tree T, written $[u, v]$ for some interior vertices u, v, is a sequence of pairwise distinct vertices of T $(u_1, \ldots, u_k)$ where u_i and u_{i+1} are connected by an edge of T for any $i \in \{1, \ldots, k-1\}$, and the edges connecting u_i to u_{i+1} and u_{i+1} to u_{i+2} share a common face for all $i \in \{1, \ldots, k-2\}$.

Each segment $[u, v]$ can be visualized as a *red admissible curve* $\gamma : [0, 1] \to D^2$ where $\gamma(0) = u, \gamma(1) = v$, γ may only intersect edges bounded by vertices $u_i, u_{i+1} \in [u, v]$, and γ leaves each endpoint to the right.



Green admissible curves proceed in the opposite direction as red admissible by connecting from left outlet to left outlet as opposed to right outlet to right outlet.

Fact. Non-crossing tree partitions can be defined using red-admissible curves as follows.
Non-crossing tree partitions can also be defined using green admissible curves.

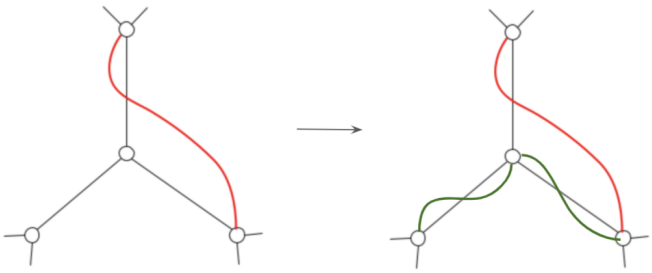
Fact. For any non-crossing tree partition **B** and a choice of red admissible curves witnessing this, there is a unique collection of green admissible curves such that when superimposed on T with the red curves on obtains a tree on V0 with the edges of T removed. Call this the "red-green tree" for **B**.

3. Kreweras Complements

The Kreweras complement is an operation that can be performed on noncrossing tree partitions to generate other noncrossing tree partitions. Its algebraic and combinatorial properties are the primary focus of this paper, with connections to representation theory made in the following section. To understand the construction of the Kreweras complement, we first introduce the concept of red-green trees.

Definition 3.1. The *red-green tree* for a noncrossing tree partition **B** is the unique collection of green admissible curves that can be superimposed upon T along with the red curves to obtain a tree on V^0 with the edges of T removed.

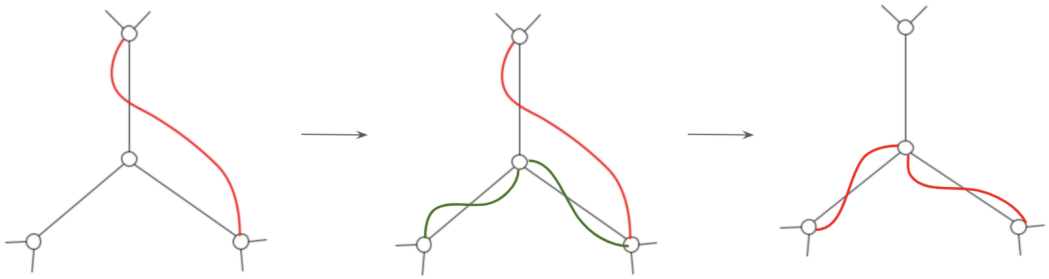
Example 3.2. Consider the noncrossing tree partition on the left, which consists of one red admissible curve. Its red-green tree can be constructed by drawing the two green admissible curve that cross the remaining two edges of the tree (right).



Definition 3.3. The *Kreweras complement* of $\mathbf{B}$, denoted $Kr(\mathbf{B})$, is the noncrossing tree partition whose green admissible curves are those of the red-green tree for $\mathbf{B}$.

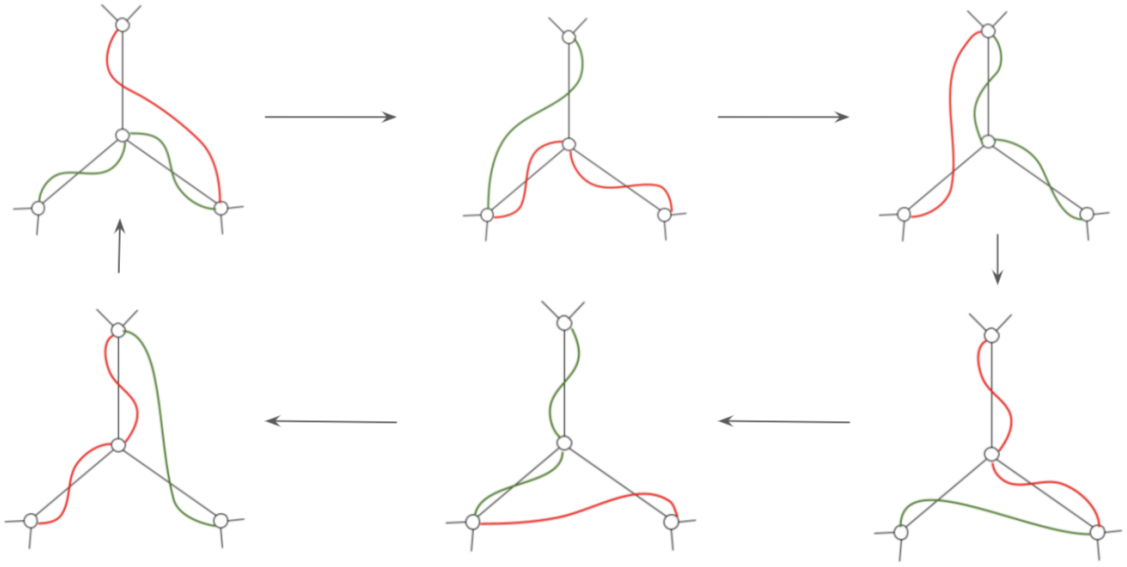
Note that the Kreweras complement exists for any noncrossing tree partition, and is always itself a noncrossing tree partition.

Example 3.4. Using the previous noncrossing tree partition (left) and its red-green tree (center), we can construct the Kreweras complement (right) by creating a new tree with red curves taking the place of the green curves in the red-green tree.



Definition 3.5. Two noncrossing tree partitions $\mathbf{B}$, $\mathbf{B'}$ are in the same *Kreweras orbit* if $Kr^m(\mathbf{B}) = \mathbf{B'}$ for some $m \geq 0$.

Example 3.6. If we repeatedly take the Kreweras complement of the noncrossing tree partition in the previous example, we find that this noncrossing tree partition is part of a Kreweras orbit of size six.



4. Representations and Quivers

Definition. A **Quiver** $Q = (Q_0, Q_1, s, t)$ consists of
Q_0 a set of vertices,
Q_1 a set of arrows,

$s : Q_1 \to Q_0$ a map from arrows to vertices, mapping an arrow to its starting point,
$t : Q_1 \to Q_0$ a map from arrows to vertices, mapping an arrow to its terminal point.
An element $\alpha \in Q_1$ can de represented by drawing an arrow from its starting point $s(\alpha)$ to its endpoint $t(\alpha)$ as follows:
$s(\alpha) \xrightarrow{\alpha} t(\alpha)$.

Definition. A **representation** $M = (M_i, \phi_\alpha)_{i \in Q_0, \alpha \in Q_1}$ of a quiver Q is a collection of $k-$ vector spaces
M_i
one for each vertex $i \in Q_0$, and a collection of $k-$ linear maps
$\phi_\alpha : M_{s(\alpha)} \to M_{t(\alpha)}$ one for each arrow $\alpha \in Q_1$ of a quiver Q is a collection of k - vector spaces
M_i
one for each vertex $i \in Q_0$, and a collection of k - linear maps
$\phi_\alpha : M_{s(\alpha)} \to M_{t(\alpha)}$
one for each arrow $\alpha \in Q_1$.
A representation M is called **finite-dimensional** if each vector space M_i is finite-dimensional. When a representation is finite-dimensional, the **dimension vector** $\underline{\dim} M$ of M is the vector $(\dim M_i)_{i \in Q_0}$ of the dimensions of the vector spaces. An **element** of a representation M is a tuple $(m_i)_{i \in Q_0}$ with $m_i \in M_i$.

Definition. For a quiver Q, the *direct sum* of its representations $M = (M_i, \phi_\alpha), M' = (M'_i, \phi'_\alpha)$ is $M \oplus M' = \left(M_i \oplus M'_i, \begin{pmatrix} \phi_\alpha & 0 \\ 0 & \phi'_\alpha \end{pmatrix} \right)$.

Definition. A representation M of a quiver is *indecomposable* if it is not a direct sum of two non-zero representations of the quiver.

Lemma. For any tree T, the indecomposable representations of $(Q_T,\ I_T)$ are in bijection with segments of T.

Fact. A morphism requires each square to commute.

Proposition For any representation M of a quiver Q, there is a decomposition
$M = M_1 \oplus \cdots \oplus M_k$, where M_i are indecomposable representations of Q unique up to order.

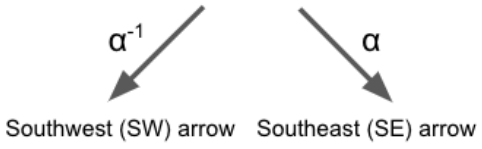
Fact. If V is a representation of Q and $dim(Hom(V, V) = 1)$, then V is indecomposable.

Definition. A *projective P(i)* is
$P(i) = (P(i)_j, \phi_\alpha)_{j \in Q_0, \alpha \in Q_1}$.
where $P(i)_j$ is the k- vector space with basis the set of all paths from i to j in Q; so the elements of $P(i)_j$ are of the form $\sum_c \lambda_c c$, where c runs over all paths from i to j, and $\lambda_c \in k$;
The arrow α induces an injective map between the bases
basis of $P(i)_j \longrightarrow$ basis of $P(i)_l$.
$c = (i|\beta_1, \beta_2, \ldots, \beta_s|j) \mapsto c\alpha = (i|\beta_1, \beta_2, \ldots, \beta_s, \alpha|l)$; and ϕ_α is defined by
$\phi_\alpha(\sum_c \lambda_c c) = \sum_c \lambda_c f(c)$.

Definition. An *injective I(i)* is
$I(i) = (I(i)_j, \phi_\alpha)_{j \in Q_0, \alpha \in Q_1}$.
where $I(i)_j$ is the k- vector space with basis the set of all paths from j to i in Q; so the elements of $I(i)_j$ are of the form $\sum_c \lambda_c c$, where c runs over all paths from j to i, and $\lambda_c \in k$;
The arrow α induces an injective map between the bases
basis of $P(i)_j \longrightarrow$ basis of $P(i)_l$.
$c = (j|\beta_1, \beta_2, \ldots, \beta_s|i) \mapsto c\alpha = (l|\beta_2, \beta_2, \ldots, \beta_s|i)$ if $\beta_1 = \alpha$; or 0 otherwise;
and ϕ_α is defined by
$\phi_\alpha(\sum_c \lambda_c c) = \sum_c \lambda_c f(c)$.

5. The Auslander-Reiten Translation for String Modules

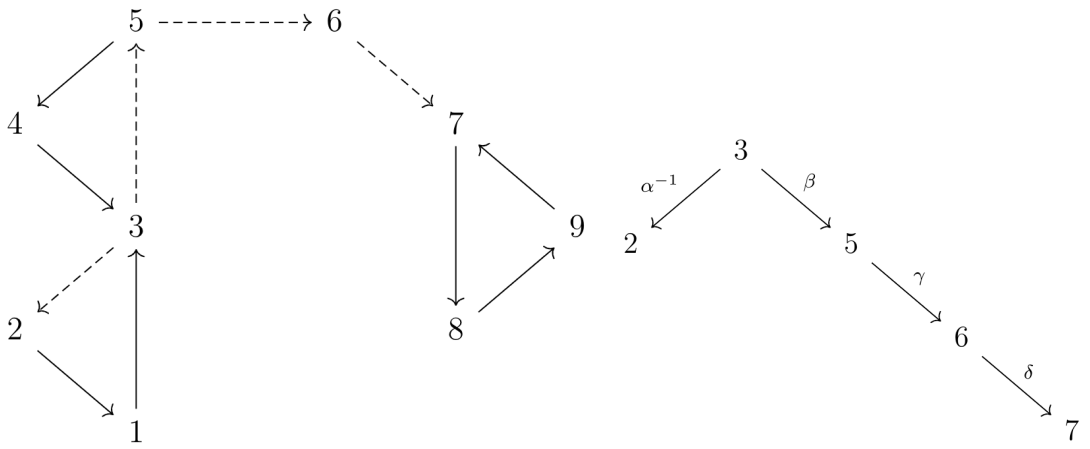
5.1. **Strings. Definition** A *string* is a sequence of arrows $w = w_{i1}^{\epsilon_1} w_{i2}^{\epsilon_2} \cdots w_{in}^{\epsilon_n}$ where each $\epsilon_i \in \{-, +\}$ and each $i \in Q$. We note that $-$ refers to a southwest (SW) arrow and $+$ refers to a southeast (SE) arrow.



Definition Given a string $w = (\epsilon, i_Q)$, $M(w)$ is the *string module*, a representation of the quiver Q.

Fact. Any representation V of (Q_T, I_T) (a quiver Q_T with the relations I_T) is supported on a string. For a representation V that is supported on a string, you can draw all the arrows that are telling you the linear transformations between the vector spaces and V. You can draw them using SW and SE arrows.

Fact. Strings can be written as a series of Greek letters with letters directed southwest including an inverse sign. For example $\alpha^{-1}\beta\gamma\delta$ for the w in our the corresponding string to our quiver below.



5.2. **Properties of Strings. Definition**. A w string *starts on a peak* if there is no arrow $\gamma \in Q$ such that γw is a string. A w string *ends on a peak* if there is no arrow $\gamma^{-1} \in Q$ such that $w\gamma^{-1}$ is a string. For any representation $M(w)$, you have 4 possibilities defined by whether w starts or ends on a peak.

Fact. All representatives I(i) for $i \in Q_0$ start on a peak and end on a peak.

Definition. A w that starts in a deep (respectively, ends in a deep) if there does not exist an arrow α such that $\alpha^{-1}w$ (respectively $w\alpha$) is a *string*.

Fact. All representatives P(i) for $i \in Q_0$ start on a deep and end in a deep.

5.3. **Hooks and Cohooks. Definition**. Hooks are built with one southeast arrow connected on its left side to as many southwest arrows as allowed, or one southwest arrow connected on its right side to as many southeast arrows as allowed.

Definition. Cohooks are built with one southeast arrow connected on its right side to as many southwest arrows as allowed, or one southwest arrow connected on its left side to as many southeast arrows as allowed.

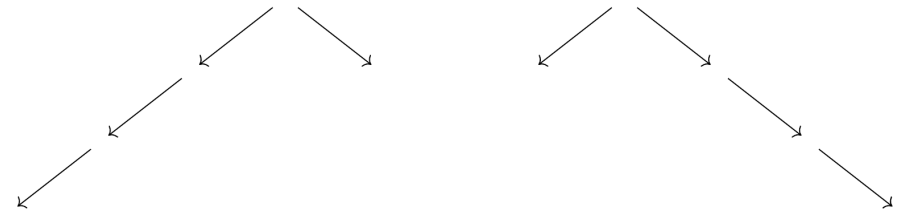


FIGURE 1. hooks

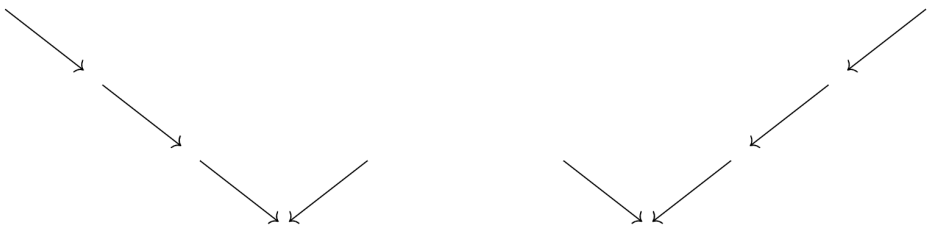


FIGURE 2. cohooks

5.4. **Applying τ^{-1} to Strings. Definition**. We define the operation τ^{-1} as the inverse of the the Auslander-Reiten Transformation, a transformation that can be applied to quivers. We instead here define it on a case by case basis on string modules. Observe that for a string module $M(w)$ that $\tau^{-1}M(w) = M(w'')$ where we find w'' using the table below. ${}_{C^{-1}}$ refers to the operation of removing a cohook (the biggest we can), while ${}_h$ refers to adding a hook to string (the biggest we can). We augment w by doing h first.

$\tau^{-1}(w) = w''$	does not start on a peak	starts on a peak
does not end on a peak	${}_h w_h$	$({}_h w)_{c^{-1}}$
ends on a peak	${}_{c^{-1}}(w_h)$	${}_{c^{-1}}w_{c^{-1}}$

Fact. $\tau^{-1}(I(i)) = 0$.

6. Useful Quiver Representations Generated From Trees

From a tree inscribed in a circle, we will construct quivers according to certain rules. The convention is for each of the edges to be represented by a vertex in the quiver. From there, each segment sharing a vertex with another is connected by a directed arrow. The arrows are arranged to point in a counterclockwise direction around the vertices in the original tree, so all of the vertices representing the edges emerging from a given point will be connected in a circle.

Within these quiver representations of trees, we also implement certain relations. For the sake of this paper, the reader simply must know that paths between arrows in a tree's quiver represent segments within our original tree, so long as a relations are not crossed. Relations are established between two consecutive arrows in a tree's quiver according to the certain rules. Any two connected directed arrows within our newly generated quiver represent a sequence of three edges in the corresponding tree. If, when attempting to connect those three edges in sequence in the original tree, we must cross an edge, we establish a relation between the two corresponding arrows in the quiver.

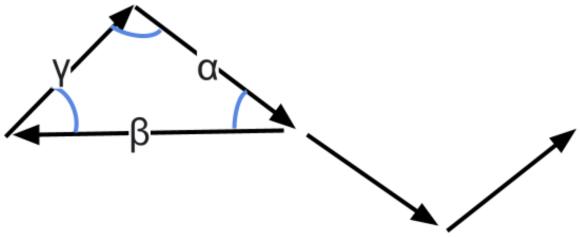


Figure 3. In this example, we must satisfy the relations that $M(\alpha)M(\beta) = 0$, $M(\beta)M(\gamma) = 0$, and $M(\gamma)M(\alpha) = 0$.

From these quivers emerges a useful way to represent segments (which also correspond to red/green curves). Any valid segment can simply be represented by a sequence of vertices in the quiver, so long as the path between the vertices does not cross any relations.

Furthermore, any such segment-representing path in the quiver can also be expressed as a string module. If we take such a path, arbitrarily starting on one of the end vertices (a vertex that does not have a directed arrow on either side) and look at the sequence of directed arrows as we move towards the other end vertex, we can list these arrows in a certain configuration. We list these arrows from left to right and connect them. For each arrow that is in the same direction we trace along the path, we draw an arrow that points downward and towards the right. If we encounter an arrow in the opposite direction we are currently tracing along the path, we draw an arrow pointing upwards and towards the right. For each consecutive arrow, we connect its tail to the head of the previous. In this way, we can represent any segment or red/green curve within a graph as a string module.

The Auslander-Reiten Transformation discussed earlier also stems from quiver representations, and is particularly of interest when applied to quivers generated from trees. If we generate an Auslander-Reiten quiver from a tree's quiver representation, each vector space in the Auslander-Reiten quiver's representation will uniquely correspond to one of the segments of minimal length in the original tree. In other words, each indecomposable representation of the quiver generated from the tree corresponds to one of the segments connecting only 2 vertices. Tau, then is defined as a movement along this Auslander-Reiten quiver, which thus corresponds to one of the tree's segments changing into another. However, we will not rigorously define Tau in terms of Auslander-Reiten Quivers, as it is not necessary

to understand the lemma we seek to prove. From this point onward, Tau will only be referenced in terms of its corresponding string module operation.

The segments of a tree are indexed by the indecomposable representations of this quiver than comes from the tree.

*** In general for quiver reresentations there is there AR translation denoted tau, and –some sort of text that says there is this thing tau and tau inverse but for our work here we will not define tau or tau inverse in general because all the representations that we are interested in are string modules and in this situation there is a combinatorial formula for calculating tau and we will explain this formula now*** We do not want to define too many things

This section will integrate 1.1, 1.2 and 1.3. What is a quiver coming from a tree. What do segments coming from modules. All of the modules coming from segments are string modules, relations on quivers, and other material that integrate together.

7. Our conjecture

Conjecture. Assume that $B \in NTP(T)$ has $|Seg(B)| = 1$. Then we have $Kr^2(B) = \tau^{-1}(B)$. Note: here we apply $\bar{\tau}^{-1}$ to the one indecomposable representation associated with B.

You may not apply tau^-1 to a non-crossing tree partition because it only applies to representations. We will need to find a way to state all of us this. We will define something else that is not tau, but that translates into tau and you can apply it to non-crossing tree partition.

A shorthand that if you took the equivalent representation, applied tau inverse and mapped it back to it.

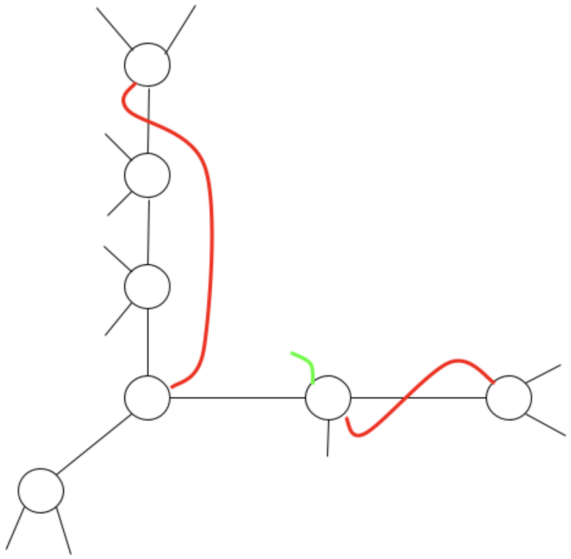
8. Algorithms For Computing Kreweras Complements

In pursuit of a modified definition of τ^-1, we attempted to develop a general algorithm for computing Kreweras complements. In this section, we outline 2 potential algorithms we have explored for computing Kreweras complements. We will first outline a disproven algorithm, which we have dubbed the "Spaghetti Noodle Algorithm," and also explain a counter example to this algorithm. We will then outline our current algorithm candidate, the "Curve Seed Algorithm," and explain why it does not fall victim to this same counterexample. We will then move on to prove various lemmas supporting this algorithm as a potentially valid candidate for complement computation.

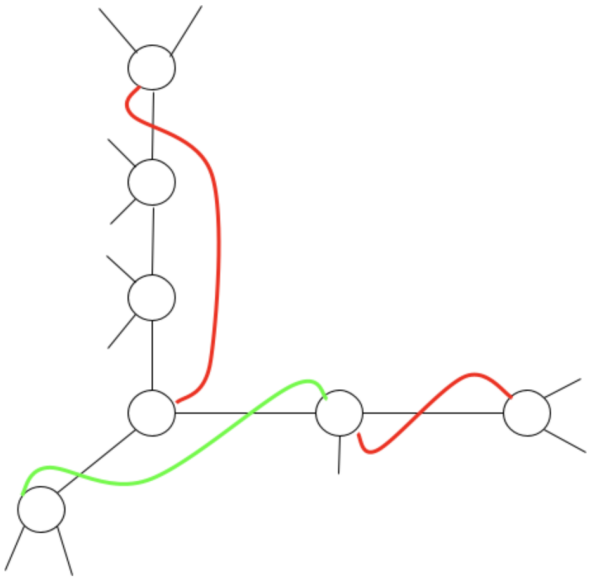
8.1. **Definition: Spaghetti Noodle Algorithm.** Given a Noncrossing Tree Partition represented by a graph G and set of red curves R, generate a series of green curves as follows:

(1) Create a queue K of curve endpoints
(2) Add all curve endpoints from R to K in no particular order.
(3) While the queue K is not empty:
    (a) Dequeue an endpoint from the front of the queue, hence referred to as the "current endpoint"
    (b) Set distance N to 1
    (c) For each vertex J that is N edges separated from the current endpoint, draw a green curve to J if and only if:
        (i) J's outlet appearing on the right as approached from the current vertex is not occupied
        (ii) No curves along the segment connecting J to the current endpoint fall under any of the cases outlined in lemma 3.2 of the paper "ORIENTED FLIP GRAPHS, NONCROSSING TREE PARTITIONS, AND REPRESENTATION THEORY OF TILING ALGEBRAS." More specifically, the segment connecting J to the current endpoint should be treated as one of the 2 "curves" and should not exhibit any of the turning behavior outlined in the lemma.
    (d) If no curves were drawn in the prior step, increment N. Continue this process until there are no vertices a distance N away from the current vertex, or a green curve is drawn.
    (e) If a curve WAS drawn in the prior step, enqueue the vertex J.

Supposedly, these newly formed green curves would form a representation of the Kreweras complement of the given NTP.



8.2. **Counterexample to the Spaghetti Noodle Algorithm.** Suppose we have a noncrossing tree partition as outlined in figure above. Because the initial endpoints are enqueued to our queue K in an arbitrary order, suppose we first enqueue the vertex in the graph. Then we will draw a green curve to the closest available outlet that does not result in a crossing.



However, this curve is not in this partition's red/green tree as illustrated in figure below.

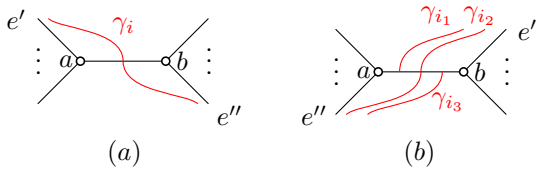
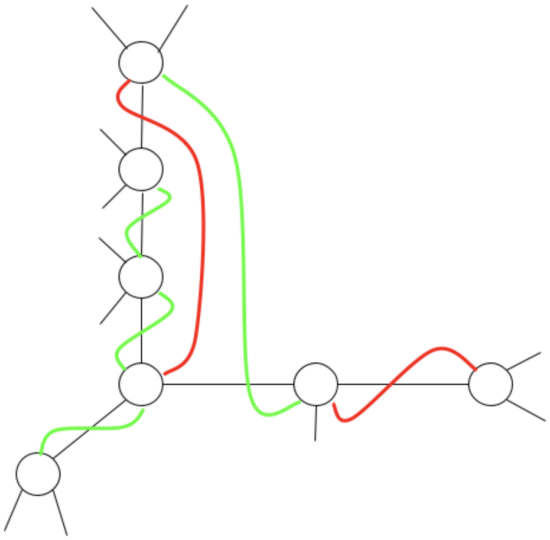


FIGURE 4. Caption



Therefore, this algorithm does not produce a valid Kreweras complement.

8.3. **Definition: Curve Seed Algorithm.** Let $\mathbf{B}$ be a noncrossing tree partition in $\mathrm{NCP}(T)$ with $\mathrm{Seg}(\mathbf{B}) = \{s_1, \ldots, s_k\}$, and let $\gamma_1, \ldots, \gamma_k$ be red admissible curves for $s_1, \ldots, s_k$, respectively, witnessing that $\mathbf{B} \in \mathrm{NCP}(T)$. Additionally, we assume that for any i, the curve γ_i only intersects T generically and the number of such intersections is minimal. This implies that any edge intersected by γ_i corresponds to a sink or a source of $w(s_i)$.

Let $\mathcal{SEG}(\mathbf{B})$ denote the topological graph embedded in D^2 whose vertices are the interior vertices of T and whose edges are the admissible curves $\gamma_1, \ldots, \gamma_k$. By construction, $\mathcal{SEG}(\mathbf{B})$ is a noncrossing topological graph (i.e., the edges are pairwise non-intersecting in their interiors).

We will show how to explicitly construct $\mathrm{Kr}(\mathbf{B})$ by adding green admissible curves to D^2 in such a way that will complete $\mathcal{SEG}(\mathbf{B})$ to a red-green tree on T.

We say that an edge e of T is an *interior edge* if both of its endpoints are interior vertices. Consider the following three types of interior edges of T:

(1) edges $e = \{a, b\}$ crossed by some curve γ_i where s_i contains edges e' and e'' as in Figure 4(a),
(2) edges $e = \{a, b\}$ crossed by some curve γ_i where s_i contains at least one of the edges e' and e'' as in Figure 4(b), and
(3) edges e not crossed by any of $\gamma_1, \ldots, \gamma_k$.

We will refer to these types of interior edges as type 1, type 2, and type 3 edges. Notice that because $\mathbf{B}$ is a noncrossing tree partition, an interior edge is either of type 1, type 2, or type 3.

Now, add several curves to the configuration $\mathcal{SEG}(\mathbf{B})$ as follows. If e is a type 1 edge with r red admissible curves intersecting it, we add $r + 1$ curves as shown in Figure 5(a). If e is a type 2 edge with r red admissible curves intersecting it, we add $r - 1$ curves as shown in Figure 5(b). Lastly, if e is a type 3 edge, we add 1 curve as in Figure 5(c). We will refer to such an added curve as a *seed*.

Next, let δ be a seed. Extend the upper end of δ so that it ends at (a, F) if possible. Assume this is not possible. Extend the upper end along e' and continuing in this direction without intersecting T or any of the red admissible curves from $\mathcal{SEG}(\mathbf{B})$ until the extension reaches another seed δ' or a corner (a', F) that is not the endpoint of any red admissible curve. If the extension first encounters such a seed δ', connect the extension to the endpoint of δ' in F. If the extension first encounters such a corner (a', F), continue the extension so that it ends at (a', F). If there is no such seed and no such corner to which can extend the upper end, we perform no such extension.

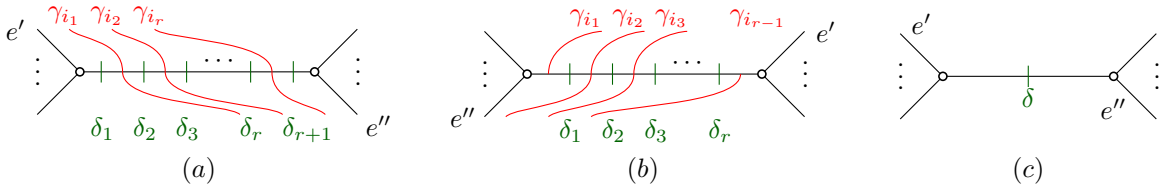
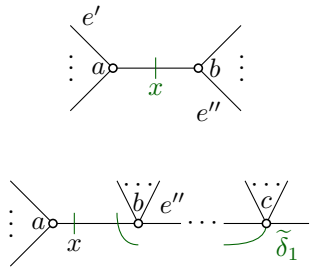


FIGURE 5. Seeds for each type of interior edge.



Now perform the same process of extending seeds to each remaining endpoint of the seeds. The result of the extension process is a configuration of curves in D^2, which we denote by $\mathcal{X}$. As we show in Lemma 8.1, any curve from $\mathcal{X}$ that is created by a sequence of extensions ends an interior vertex of T.

Lemma 8.1. *Let $\widetilde{\delta}$ be a curve in $\mathcal{X}$ that is obtained by a sequence of extensions of seeds. Then $\widetilde{\delta}$ begins and ends at corners of T. Moreover, the curve $\widetilde{\delta}$ is a green admissible curve.*

Proof. Suppose that $\widetilde{\delta}$ has an endpoint x that is not at a corner. Let F be the face of the tree containing x. Additionally, assume that $e = \{a, b\}$ is the edge that x's seed crosses. Up to isotopy, the curve containing x appears in one of the configurations from Figure 8.3.

First, assume that there are no red curves that cross e. This implies that e is a type 3 edge. By assumption x does not end at (b, F) and cannot be extended along e''. Traveling from b along e'', let (c, F) be the first corner one encounters at which a green admissible curve $\widetilde{\delta}_1$ ends. Such a green admissible curve exists because

This shows that the endpoints of $\widetilde{\delta}$ are, in fact, corners of T. Moreover, the configuration of these corners shows that $\widetilde{\delta}$ is a green admissible curve if it is supported on a segment.

To see that $\widetilde{\delta}$ is supported on a segment. By the construction of $\mathcal{X}$, the only ways in which $\widetilde{\delta}$ crosses T up to equivalence of trees are shown in Figure $\widetilde{s} = [a, b]$ denote the path in T determined by $\widetilde{\delta}$. The configurations appearing in Figure **??** show that any two consecutive edges of $\widetilde{s}$ are incident to a common face. Thus $\widetilde{\delta}$ is supported by a segment. This completes the proof. □

Theorem 8.2. *The configuration $\mathcal{X}$ is the red-green tree for $\boldsymbol{B}$.*

Proof. This is a consequence of Lemmas 8.3, 8.4, 8.5. □

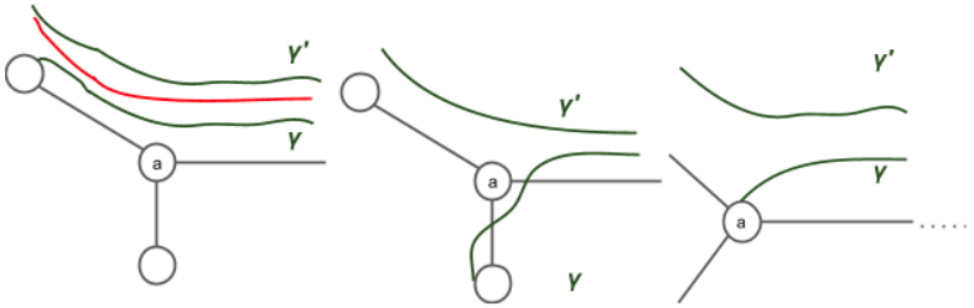
Lemma 8.3. *The configuration $\mathcal{X}$ consists of pairwise noncrossing curves.*

Proof. By definition, red curves cannot cross one another and green curves cannot cross red curves. Therefore, we set out to prove that green curves will not cross one another.

Consider two green curves, γ and γ'. Let $[a, b]$ be the unique maximal subsegment that both γ and γ' share and travel down. Since γ and γ' both travel down $[a, b]$, they may either start on the same side of $[a, b]$ or opposite sides. We break this into 2 cases.

Case 1: γ and γ' are on the same side of $[a, b]$.

Suppose γ and γ' start off on the same side of $[a, b]$. Since they both start sharing edges at vertex a, this implies that one of the two green curves must originate at a. Otherwise, they would both be sharing edges, since they would have both originated earlier vertices and traveled down the edge connected to a. Let γ be the green curve that originates at a.



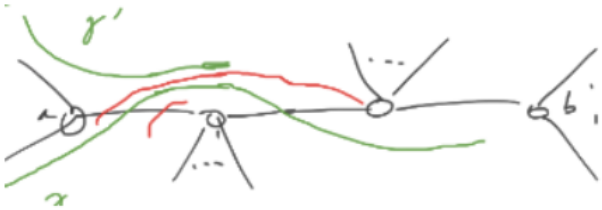
Let b' be the vertex along $[a, b]$ before b. When traveling down $[a, b']$, γ and γ' can only cross edges along $[a, b']$ together, since otherwise they are missing edges along $[a, b']$ or taking inefficient routes, which goes against the algorithm. When crossing down, they cross through a type I edge. When it crosses up, it crosses a type II edge. Note that γ and γ' need not cross together along the edge $[b', b]$, since from this point they are free to travel along different edges.

We now establish 2 subcases in terms of whether there exists an edge e along $[a, b]$ that γ and γ' both cross.

Subcase 1.

Consider the subcase that γ and γ' both cross an edge e along $[a, b]$. Since edge e must be a type I or type II edge, this implies that a red curve, π, will separate the two green curves.

Now, we will show that π will have originated before vertex a and terminate at or past vertex b. By the definition of type I and type II edges, π is confined between the two green curves. Now suppose the red curve terminated at some vertex along $[a, b]$. Then it will be obstructing a green curve from crossing edge e, making the green curve invalid.

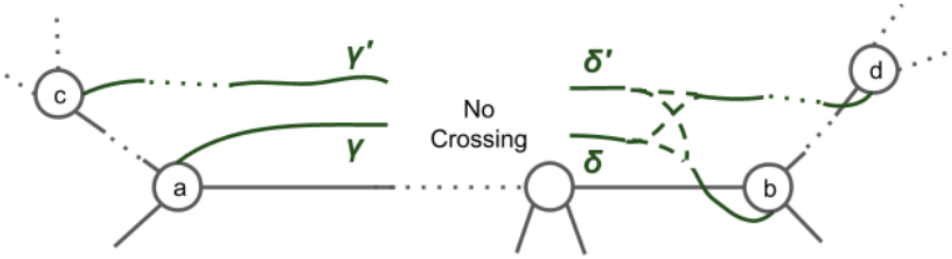


π cannot access a vertex until either γ or γ' terminates. Since γ or γ' terminates will not terminate along $[a, b]$, π cannot terminate along $[a, b]$. Now because π continues to separate γ or γ' along $[a, b]$, the green curves cannot cross one another. Since $[a, b]$ is the unique maximal subsegment of both γ and γ', this proves they will not cross.

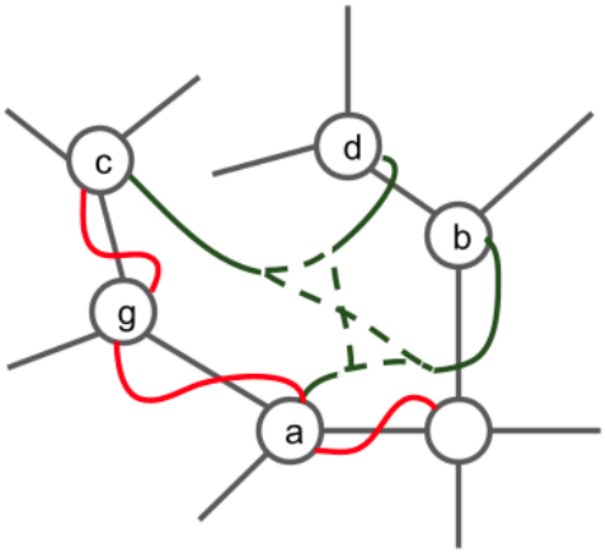
Subcase 2.

Now consider the subcase that there does not exist an edge e along $[a, b]$ that γ and γ' both cross. Since γ and γ' started on the same side of $[a, b]$ and have not crossed, when they reach vertex b, one of them must terminate. Otherwise, $[a, b]$ would not be the longest segment that they share. This implies a single crossing by one of the curves must occur in order for one of the curves to connect to the left outlet of b.

Let γ be the curve that terminates at b and let δ' continue past b. We will prove that δ is γ and δ' is γ'. Observe that because δ' does not cross along $[a, b]$, it must continue upwards above b. Let c be the vertex where γ' originates and d be the vertex where δ' terminates.



Consider connected components T_a, T_b, T_c and T_d be where T_v is the connected component made of green and red curves that includes vertex v. We will prove that T_a is not connected to T_c and T_b is not connected to T_d. We define an outlet L_{xy} as being the left outlet of vertex x when looking down the first edge in the path from x to y. R_{xy} is that for the right outlet of x. Now suppose T_a is connected to T_c. This implies that a is connected (directly or indirectly) to c by vertices in T_a and T_c. Let S be the sequence of vertices of our tree, such that a and c can be reached from one another by following the red and green curves connecting a to S to c. We consider super-subcases based on the location of the last vertex of S.



Super-subcase 1.

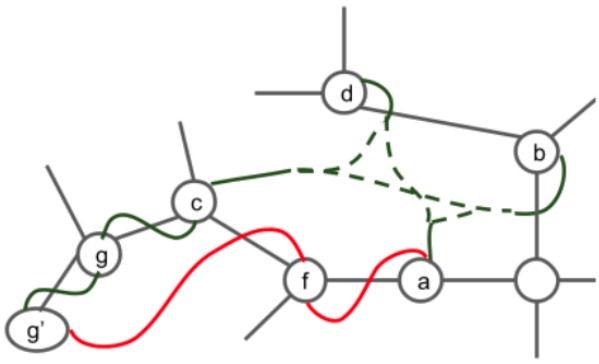
Let c be directly connected to a. Since L_{ca} is occupied by γ', it cannot be connected by a green curve to g. This implies a red curve, π, connects to c via R_{ca}. Then π connects to a at R_{ac}. Now, recall that γ connects a to either b or d. Therefore, γ occupies the outlet $L_{ab} = L_{ad}$ (they are the same because b and d are on the right side of a). Since $L_{ab} = R_{ac}$, this outlet must have two curves occupying it, which creates a contradiction.

Super-subcase 2.

Let g be the last vertex in S and let g exist to the left of $[a, c]$. Let g' be the previous vertex in S and note that g' cannot be left of g.

Consider what would happen if g' were. g' could either use a green curve or red curve. If it used a green curve, then this implies the repeated use of green curves among outlets left of g. They could not be reached then by a vertex right of g, which they must because c is right of g, because the left-most vertex would have two green curves in its outlet or a long red curve that obstructs the outlets of g or g'.

Since g' is to the right of g, then g' must connect to g at L_{ga}. Otherwise R_{ga} would be occupied, obstructing L_{ga}, which is the other outlet that g could use to connect to a. Therefore, L_{ga} must be occupied and R_{ga} is used to connect g to a with a red curve π. As before, π connects to a at R_{ac}, which creates a contradiction.



Super-subcase 3.

Let the vertex g be the last vertex in S and let g lie along the path of $[a, c]$. Let g' be the previous vertex in S.

If g' is to the left of $[a, c]$, then as in super-subcase 2, it must use a red curve to connect to a vertex to its right, in this case g. This would result in occupy $R_{gg'} = L_{ga}$ being occupied by a red curve.

If g' is to the right of $[a, c]$, then a green curve must be used to connect to g. Otherwise, the red curve would cross an edge along $[g, c]$, which would obstruct the curve from g to c. Thus, in either case L_{ga} ends up occupied.

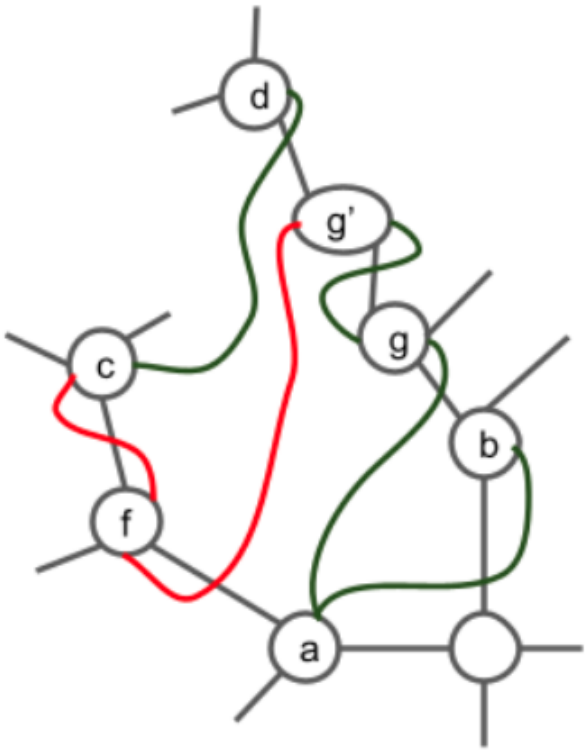
As a result, g must use its right outlet, R_{ga}, to connect to the R_{ag}, which creates a contradiction.

Super-subcase 4. Let the vertex g be the last vertex in S and let g be to the right of $[a, c]$. Let g' be the previous vertex in S.

If g' is to the left of or along $[a, g]$, then as shown in super-subcases 2 and 3, it must use a red curve to connect. In this case, R_{ga} is occupied.

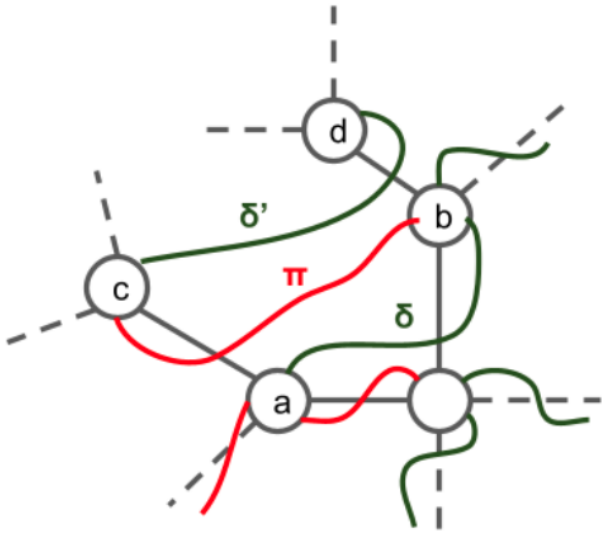
If g' to the right of $[a, g]$, then consider the previous vertex f in the sequence S that is to the left of or along $[a, g]$. Note that such a vertex must exist, since c falls under this category. As previously shown f must use a red vertex to connect to the following vertex f'. This will occupy $R_{f'f}$, so that f' must use its left outlet $L_{f'g}$ to connect to the next outlet, which will be to the right of f' by the definition of f. This will result in the continued use of green curves until g is reached. Thus, $L_{gg'} = R_{ga}$ will be again occupied.

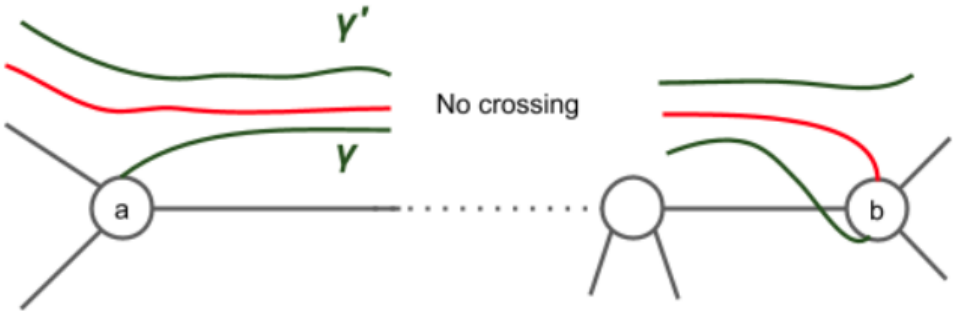
Hence, g must connect to a with L_{ga} and connect to $L_{ag} = L_{ab}$. This once more creates a contradiction.



This proves that T_a is not connected to T_c. Using the same logic, one can prove T_b is not connected to T_d.

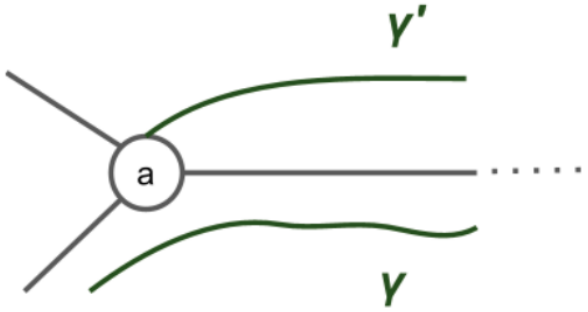
Now assume $\gamma = \delta'$ and $\gamma' = \delta$ for a proof by contradiction. Then T_a is connected to T_d, forming the connected component T_{ad}, and T_b is connected to T_c, forming the connected component T_{bc}. Using the result of Lemma 8.5, which states that the configuration from the algorithm is connected, then T_{ad} must connect to T_{bc}. However, using a proof similar to that showing adjacent connected components T_a and T_c are not connected, we can show that neither T_a is connected to T_b nor T_c is connected to T_d. Therefore we have reached a contradiction and instead $\gamma = \delta$ and $\gamma' = \delta'$.



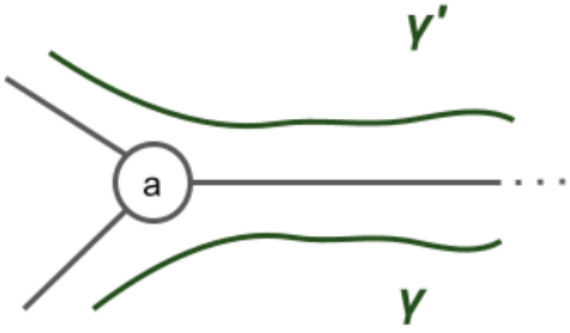


Case 2.

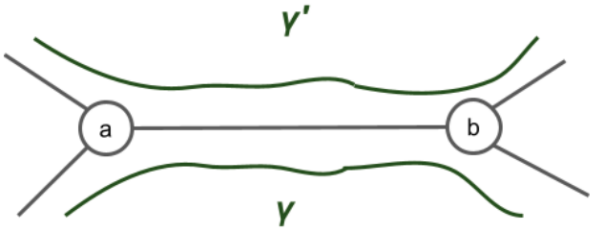
γ and γ' are on opposite sides of $[a, b]$. If one of the curves begins at a, it must be the top curve because it alone has access to the left outlet.



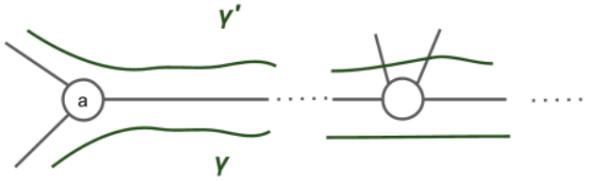
However, it is also possible for neither of the curves to begin at a.



Suppose neither of the curves cross. This implies that [a,b] has a length of 1. Since the curves don't cross the edge [a,b], then they do not cross each other.



Consider the case where length([a,b]) ¿ 1. This implies that an interior vertex v with degree at least 3 lies along [a,b]. Two of the edges, e and e', from v will be part of [a,b]. Hence, on one side of [e,e'] is empty space and the other contains the third edge of v. Both curves must pass on the side of the empty space or otherwise they would be crossing edges not contained within their segments. Since the curves begin on opposite sides, but end up on the same side, this implies exactly one curve must cross.



This tells us that, if at least one curve crosses, the first curve that crosses must cross alone so that both curves end up on the same side. From then on, γ and γ' have to always cross together along $[a, b]$. Otherwise, as described before, they will have to cross edges outside of $[a, b]$ and not be valid curves. Once the green curves are on the same side, this case then behaves the same as Case 1. Thus, this proves the green curves will not cross.

□

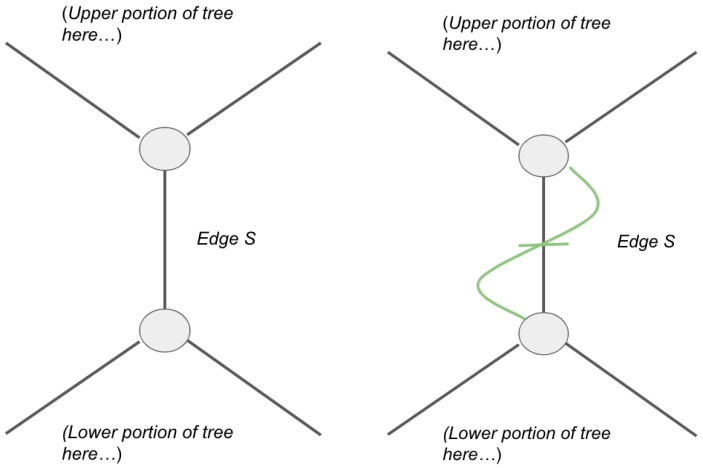
Lemma 8.4. *The configuration $\mathcal{X}$ is acyclic as a topological graph.*

Proof. Suppose, by way of contradiction, that $\mathcal{X}$ contains a cycle of pairwise noncrossing curves. Then the cycle can be expressed as a path of curves $p = \{c_1, c_2, \ldots, c_n\}$ where c_j is either some red admissible curve γ_j, or some green admissible curve δ_j. If each $c_j \in p$ corresponds to a segment whose vertices are connected by a single edge of T, then T contains a cycle comprised of the vertices and segments of p, which cannot occur by construction.
If there is some $c_j \in p$ which corresponds to a segment whose vertices are not connected by a single edge of T, then there must exist some path from v_j to v_{j+1} in order for c_j to be inclusion-minimal. If such a path exists, then all of the vertices of p are connected by edges, and T must contain a cycle. Hence, $\mathcal{X}$ must be acyclic. □

Lemma 8.5. *The configuration $\mathcal{X}$ is connected as a topological graph.*

Lemma 8.6. *The curve seed algorithm is valid for any type 3 edge S without any red curves along it and whose "inner left outlets" (the outlets that would be occupied by a green curve connecting the two endpoints) are unoccupied.*

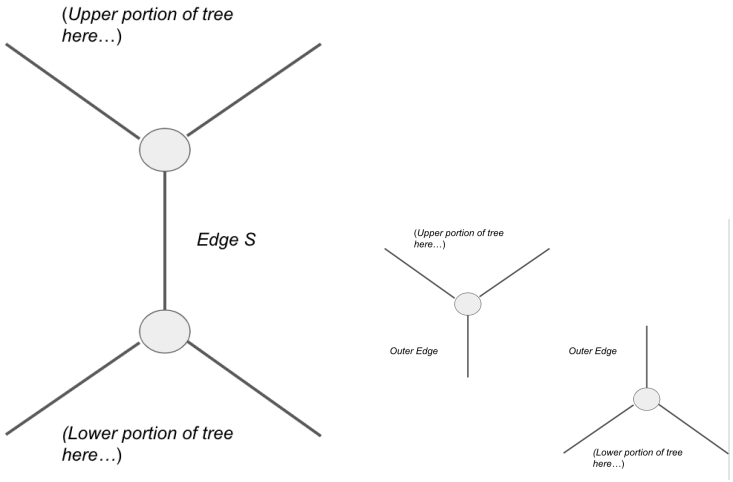
Proof. Given a type 3 edge S without any curves along it, the curve seed algorithm will generate a green curve connecting the endpoints of said edge given the corresponding outlets at the endpoints of said edge are unoccupied. (See the example below)



This is because a green curve will must grow from the seed along edge S and also must connect to the endpoints of edge S (because they are the nearest possible points of connection).

Next, we will show that the green curve drawn in this case will always be in the red/green tree of the given graph.

Suppose for a moment that the tree were split along the edge S as demonstrated in the image below. Now form the upper and lower halves of the split into 2 new distinct trees by replacing the edge S in the upper half and lower half with an "outer edge," or an edge that does not connect to an interior vertex.

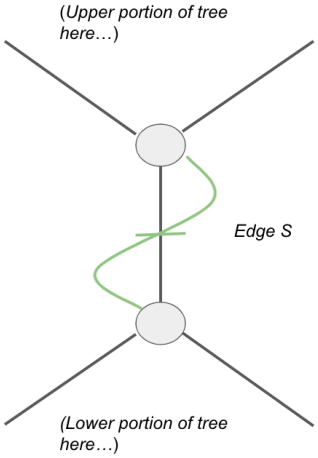


By our initial assumptions, no red curves crossed the edge S and no red curves crossed between the upper and lower halves of the tree, so they remain unchanged.

Therefore, the upper and lower trees have valid red curves and therefore valid noncrossing tree partitions.

Therefore, we know there also must exist a valid and unique set of greed curves to form a red green tree in the upper and lower halves.

Given such a set of green curves in the upper and lower halves, we could rejoin these 2 halves via edge S and a green curve along edge S to form a complete red/green tree.



Therefore, because a valid red/green tree can be constructed in conjunction with such a green curve, the green curve generated by the curve seed algorithm must be a curve in the red/green tree within this case (by the uniqueness of red/green trees).

□