Enumeration of Minimum Path Covers of Trees

C.R. Johnson and M. Sher

Abstract - In the study of possible multiplicity lists for eigenvalues of real symmetric matrices with a given tree T as graph, the value of maximum multiplicity, $M(T)$, has been of great interest. $M(T)$ turns out to be the same as $P(T)$, the path cover number of T. This is the minimum number of induced paths of T that are vertex disjoint and cover all its vertices. For a given tree T, $P(T)$ can be achieved in several ways. Here, we give a method to enumerate all possible minimum path covers for any given tree through a reductive procedure.

Keywords : maximum multiplicity; minimum path cover; real symmetric matrices; trees

Mathematics Subject Classification (2020) **:** 05C30; 05C70; 15A18

1 Introduction

In the past 2+ decades a theory of possible multiplicity lists for the eigenvalue of real symmetric matrices with a given graph has been heavily studied [1]. The theory is most well developed for trees. A key fact is that, for a tree T, the maximum multiplicity $M(T)$ is the path cover number $P(T)$ [2]. This is the fewest induced paths of the tree that are vertex disjoint and include all its vertices; a set of such paths is called a minimum path cover (MPC) of T. A single vertex is allowed to count as a path. An equivalent number is $\max(p-q)$ in which p is the number of (simple) paths remaining after the removal of q vertices from T [2]. Both $P(T)$ and $\max(p-q)$ may be attained in multiple ways, but there is little relationship in how they are attained.

The path cover number is a concept that seems to have arisen from the (new) multiplicity theory. There are simple algorithms to evaluate $P(T)$, or $\max(p-q)$ [3]. But, a natural question to ask is how many distinct MPC's, $N(T)$, occur for a given tree. This question has been raised [4], but has not yet been fully answered.

Our purpose here is to give a procedure for the enumeration of $N(T)$ that works for any tree by reduction to simple trees for which $N(T)$ is obvious. There are two parts to our approach to the enumeration. One is to subdivide the tree through identifying special vertices or special overlapping positions of the tree for which the count is multiplicative. The other is to partition the inclusion of certain special edges and divide the set of all MPC's of the tree into two subsets for which the count is additive. The former is why the growth of $N(T)$ can be rapid.

In the next section, some definitions and useful technical background are given. Then the reduction procedure, by type of situation is given in the next 3 sections, with a

final section for an overview of the complete procedure and additional examples and observations.

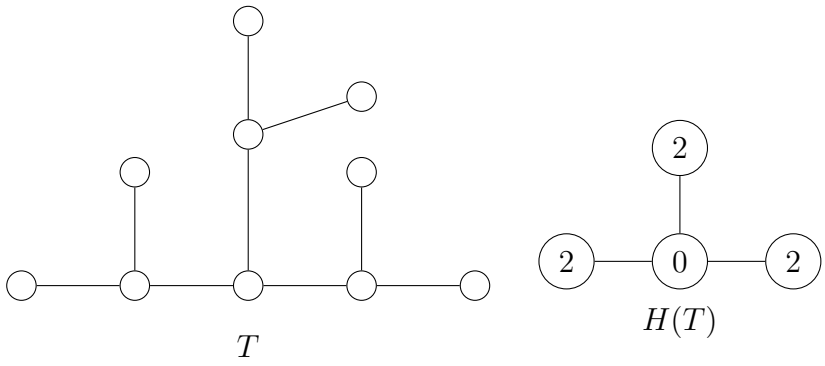
2 Background

Let $A = (a_{ij})$ be an n-by-n real symmetric matrix. The graph of A, denoted $G(A)$, is the simple undirected graph on n vertices with an edge $\{i, j\}$ if and only if $i \neq j$ and $a_{ij} \neq 0$. We use $\mathcal{S}(G)$ to denote the set of all real symmetric matrices whose graph is G.

Given a tree T, let $\deg_T(v)$ denote the **degree** of a vertex v in T, which is the number of neighbors of v. $M(T)$ denotes the largest possible multiplicity that occurs for eigenvalues of matrices in $\mathcal{S}(T)$.

In a tree T, a **high degree vertex** (HDV) is one of degree at least 3. Otherwise, the vertex is of **low degree**. A **leaf** is one of degree 1. The **Hi-graph**, $H(T)$, of T is the subgraph induced by its HDV's. $H(T)$ is a forest with one or more components (each of which is a tree). The **incremental degree** of a vertex v, $\delta(v)$, is the difference between its degrees in T and in $H(T)$. A **high-incremental degree** (HID) vertex in $H(T)$ is one of incremental degree at least 2; otherwise it is of **low-incremental degree** (LID).

Example 2.1 Given a tree T, its Hi-graph is shown on the right with all vertices in $H(T)$ labeled with their respective incremental degree.



T $\quad$ $H(T)$

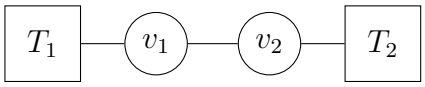
It is also helpful to characterize how edges are used in different MPC's of T when calculating $N(T)$. For a tree T with $N(T) = 1$, each of its edges is either included in the MPC or not. In Appendix A, a brief discussion of trees with a unique MPC is included. For a general tree T with possibly multiple MPC's, the MPC's differ from one another by including different sets of edges. We distinguish 3 statuses for edges in T.

Definition 2.2 *An edge is **absent** if it is used in no MPC of T. An edge is **required** if it is used in all MPC's. An edge is **discretionary** if it occurs in some but not all MPC's.*

We consider how to identify the status of an edge and how an edge contributes to our knowledge of $N(T)$, both here and in later sections.

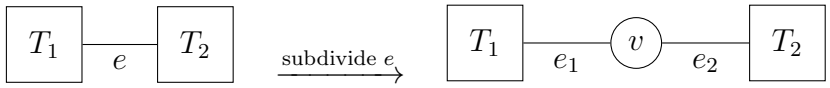
Lemma 2.3 *Any edge between two low degree vertices is required.*

Proof. Shown below is the general structure of two adjacent low degree vertices v_1 and v_2 in a tree T. T_1 and T_2 represent the subtrees connected respectively to v_1 and v_2 in T. One or both of them can be empty.



We will complete the proof by contradiction. Let $\mathcal{C}_1$ be an MPC of T. Assume that the edge (v_1, v_2) is not used in $\mathcal{C}_1$. Then, there are two paths in $\mathcal{C}_1$ that terminate at v_1 and v_2. Through merging these two paths into one by including (v_1, v_2) and thus connecting v_1 and v_2, we construct a new path cover for T, $\mathcal{C}_2$. All the other paths in $\mathcal{C}_2$ are the same as in $\mathcal{C}_1$. The new path cover, $\mathcal{C}_2$, covers all the vertices in T and contains one fewer path than $\mathcal{C}_1$. Since we assumed $\mathcal{C}_1$ to be an MPC, a contradiction is reached. Therefore, (v_1, v_2) must be included in all MPC's of T. □

Edge subdivision is the introduction of a new degree-2 vertex, positioned along an existing edge.



Lemma 2.4 *If T' is obtained by subdividing a required edge in T, then $N(T') = N(T)$.*

Proof. Given that the edge e in the illustration above is required in T, we will first show that after subdividing e, the resulting edges e_1 and e_2 are both required in the new tree T'.

Since we can always construct a path cover for T' from an MPC of T by including e_1, v, and e_2 in the path that originally includes e and keeping other paths unchanged, we have $P(T') \leq P(T)$. Without loss of generality, assume that e_1 is not required in T'. Then, for a *minimum* path cover of T', $\mathcal{C}_{T'}$, that does not use e_1, either e_2 is used and v is included in a path that goes into T_2 or e_2 is not used and v_2 is included in $\mathcal{C}_{T'}$ as a singleton.

If e_2 is used in $\mathcal{C}_{T'}$, we can construct a corresponding path cover $\mathcal{C}_T$ for T that preserves the paths in $\mathcal{C}_{T'}$ except that the path that originally terminates at v now terminates at the vertex in T_2 that is connected to e. If e_2 is not used in $\mathcal{C}_{T'}$, we can also construct a corresponding path cover $\mathcal{C}_T$ for T that preserves all the paths in $\mathcal{C}_{T'}$ excluding v as a singleton. For both cases, $|\mathcal{C}_T| \leq |\mathcal{C}_{T'}| \leq P(T)$, meaning $\mathcal{C}_T$, in which the edge e is not used, is a *minimum* path cover of T. However, we assumed e to be a required edge, and thus a contradiction is reached.

Therefore, e_1 and e_2 are both required in T, which means that e_1, v, and e_2 must be included in the same path in every MPC of T', making them equivalent to e, a single edge, in T. Therefore, we have $N(T') = N(T)$. □

Corollary 2.5 *For a tree T, the value of $N(T)$ is independent of the lengths of the paths induced by the low degree vertices in T.*

Remark 2.6 For a tree T, an LID vertex cannot be a leaf of $H(T)$. Furthermore, a vertex of incremental degree 0 in $H(T)$ must have at least 3 pendent branches. Otherwise, they would not have been included in the Hi-graph.

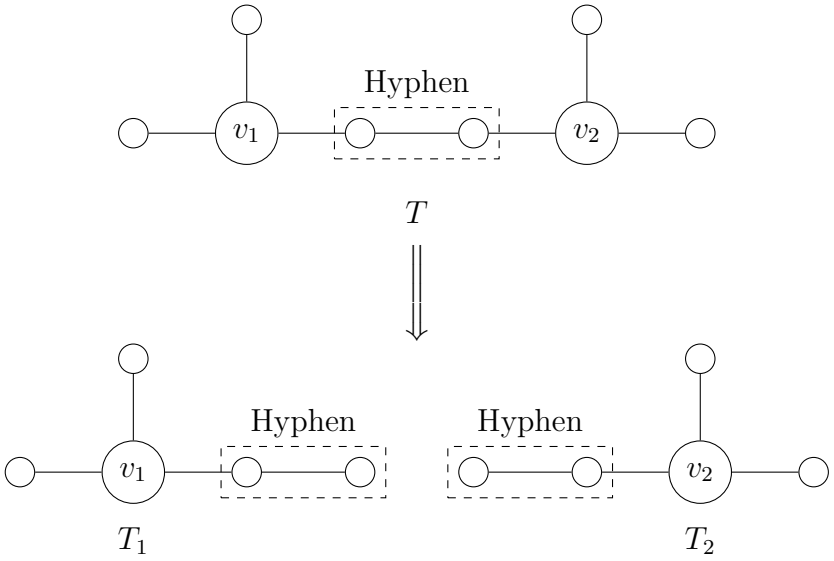
3 Trees with Multiple-Component Hi-graphs

The Hi-graph of a tree T has two or more components when there are one or more low-degree vertices on a single path between two of the HDV's. We will call such a path, induced by the low-degree vertex or vertices between two HDV's in the original tree, a **hyphen**.

Remark 3.1 By Lemma 2.3 and Corollary 2.5, the edges in a hyphen are required and the value of $N(T)$ is independent of the lengths of the hyphens.

Given a tree with a Hi-graph of k components, where $k \geq 2$, it has $k-1$ hyphens. We select a hyphen l_h and consider it a shared boundary between the two neighboring components. The two HDV's connected by the hyphen are denoted v_1 and v_2, respectively. The process of **hyphen decomposition** is defined as follows. We first separate the two components by removing the edge between l_h and v_2, with the resulting tree that contains v_1 denoted T_1. Similarly, T_2 is obtained by removing the edge between l_h and v_1 and selecting the part that includes v_2. Note that after the separation, both T_1 and T_2 include l_h.

Below is a tree with a two-component Hi-graph with each component being a singleton of incremental degree 3. The hyphen is labeled, and the process of hyphen-decomposing T into T_1 and T_2 is displayed.



It is easier to enumerate $N(T_1)$ and $N(T_2)$ than to directly calculate $N(T)$. In fact, we will show that $N(T)$ is the product of the two.

Lemma 3.2 *Suppose that the Hi-graph of a tree T has two or more components. T is then hyphen-decomposed into T_1 and T_2 at one of the hyphens. Then, the selected hyphen is always included in a single path in every MPC of T_1 and T_2.*

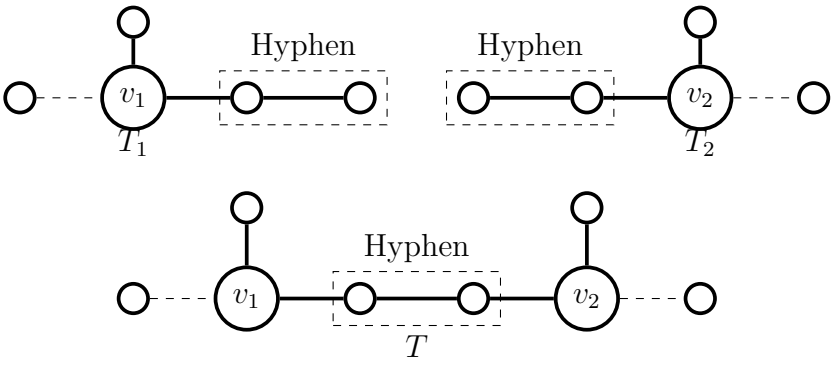
Proof. Since T_1 and T_2 are named arbitrarily, we only need to present the proof for T_1. When the hyphen is a degenerate path, the statement is trivially true. When the hyphen is composed of two or more vertices, every vertex in it is of low degree. Therefore, by Lemma 2.3, every edge in the hyphen is required, meaning the hyphen must be included in a single path in every MPC of T_1. The same argument applies to T_2. □

Lemma 3.3 *Suppose that the Hi-graph of a tree T has two or more components. T is then hyphen-decomposed into T_1 and T_2 at one of the hyphens. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two MPC's of T_1 and T_2, respectively. Then $\mathcal{C}_1$ and $\mathcal{C}_2$ can be merged into a path cover of T by taking the two paths in $\mathcal{C}_1$ and $\mathcal{C}_2$ that both contain the hyphen and merging them into one path. Note that two such paths must exist by Lemma 3.2. The other paths in $\mathcal{C}_1$ and $\mathcal{C}_2$ are unchanged by the merge. The resulting path cover is an MPC of T.*

Proof. An example of merging MPC's of T_1 and T_2 is shown below. We will complete the proof by contradiction. Suppose that the resulting path cover of T is not an MPC. Thus, the number of paths in it, k, is greater than $P(T)$. We have

$$k = |\mathcal{C}_1| + |\mathcal{C}_2| - 1 > P(T).$$

Because the edges in the hyphen are required in T, the reverse of the merging process can be applied on an MPC of T to obtain two path covers for T_1 and T_2. We apply the reverse-merging on an MPC of T, during which the count of paths will increase by 1, and thus the total number of paths in these two path covers is $P(T) + 1$. But we have $|\mathcal{C}_1| + |\mathcal{C}_2| > P(T) + 1$, suggesting that at least one of $\mathcal{C}_1$ and $\mathcal{C}_2$ is not minimum. A contradiction is reached. Therefore, the resulting path cover of T from merging $\mathcal{C}_1$ and $\mathcal{C}_2$ is an MPC of T. □



We now consider the relationship between $\boldsymbol{\mathcal{P}}(T_1)$, $\boldsymbol{\mathcal{P}}(T_2)$, and $\boldsymbol{\mathcal{P}}(T)$, the sets of MPC's of T_1, T_2, and T.

Lemma 3.4 *Suppose that the Hi-graph of a tree T has two or more components. T is then hyphen-decomposed into T_1 and T_2 at one of the hyphens. There exists a bijection between $\boldsymbol{\mathcal{P}}(T_1) \times \boldsymbol{\mathcal{P}}(T_2)$ and $\boldsymbol{\mathcal{P}}(T)$.*

Proof. Let $\mathcal{C}_1$ and $\mathcal{C}_2$ be two MPC's of T_1 and T_2, respectively. We will first construct a function f that maps from $\boldsymbol{\mathcal{P}}(T_1) \times \boldsymbol{\mathcal{P}}(T_2)$ to $\boldsymbol{\mathcal{P}}(T)$.

Define $f : \boldsymbol{\mathcal{P}}(T_1) \times \boldsymbol{\mathcal{P}}(T_2) \to \boldsymbol{\mathcal{P}}(T)$. The function f merges $\mathcal{C}_1$ and $\mathcal{C}_2$ in the same way as described in Lemma 3.3. Let $\mathcal{C} \in \boldsymbol{\mathcal{P}}(T)$ be the resulting MPC. We then have $f(\mathcal{C}_1, \mathcal{C}_2) = \mathcal{C}$ for $\mathcal{C}_1 \in \boldsymbol{\mathcal{P}}(T_1), \mathcal{C}_2 \in \boldsymbol{\mathcal{P}}(T_2), \mathcal{C} \in \boldsymbol{\mathcal{P}}(T)$.

For $\mathcal{C}_1, \mathcal{C}_1' \in \boldsymbol{\mathcal{P}}(T_1)$ and $\mathcal{C}_2, \mathcal{C}_2' \in \boldsymbol{\mathcal{P}}(T_2)$, if $(\mathcal{C}_1, \mathcal{C}_2) = (\mathcal{C}_1', \mathcal{C}_2')$, we show that $\mathcal{C} = f(\mathcal{C}_1, \mathcal{C}_2) = f(\mathcal{C}_1', \mathcal{C}_2') = \mathcal{C}'$. Given the merging procedure described above, it is impossible for two sets of identical MPC's of T_1 and T_2 to become distinct MPC's of T after the merge. Therefore, f is well-defined.

Now show that f is a bijection. We first prove that f is injective by showing that if $\mathcal{C}_1 \neq \mathcal{C}_1'$ or $\mathcal{C}_2 \neq \mathcal{C}_2'$, then $\mathcal{C} \neq \mathcal{C}'$. This is true by the merging process described above. Let f^{-1} be defined as the reverse merging process. An MPC of T gets split up into MPC's of T_1 and T_2. T_1 is the component that contains v_1 after the removal of the edge between the hyphen and v_2; Similarly, T_2 is the component that contains v_2 after the removal of the edge between the hyphen and v_1. Then, for each $\mathcal{C}^{(i)} \in \boldsymbol{\mathcal{P}}(T)$, we get $f^{-1}(\mathcal{C}^{(i)}) = (\mathcal{C}_1^{(i)}, \mathcal{C}_2^{(i)})$, with $(\mathcal{C}_1^{(i)}, \mathcal{C}_2^{(i)}) \in \boldsymbol{\mathcal{P}}(T_1) \times \boldsymbol{\mathcal{P}}(T_2)$. Therefore, f is surjective.

Because f is both injective and surjective, it is a bijection. □

Theorem 3.5 *Let T be a tree with a multiple-component Hi-graph, and let T_1 and T_2 be the results of hyphen-decomposing T at any selected hyphen. Then, $N(T) = N(T_1)N(T_2)$.*

Proof. As is shown in Theorem 3.4, there exists a bijection between $\boldsymbol{\mathcal{P}}(T_1) \times \boldsymbol{\mathcal{P}}(T_2)$ and $\boldsymbol{\mathcal{P}}(T)$. The two sets have the same cardinality. Hence, $N(T_1) \cdot N(T_2) = |\boldsymbol{\mathcal{P}}(T_1)||\boldsymbol{\mathcal{P}}(T_2)| = |\boldsymbol{\mathcal{P}}(T_1) \times \boldsymbol{\mathcal{P}}(T_2)| = |\boldsymbol{\mathcal{P}}(T)| = N(T)$. □

Given a tree T of k components and $k-1$ hyphens, hyphen-decomposition can be applied at each of the $k-1$ hyphens in a sequential order to decompose T into k parts. Each of the resulting parts has a single-component Hi-graph.

Algorithm 3.6 *Given a tree T with a k-component Hi-graph and $k-1$ hyphens,*

1. *Identify all the hyphens and assign indices $1, 2, \cdots, k-1$ to them. The order of the assignment does not matter as long as each hyphen is assigned exactly once.*
2. *Starting from $i = 1$, while $i \leq k-1$,*
   (a) *decompose the component that includes the i-th hyphen into two subparts through applying hyphen-decomposition at the i-th hyphen, and*
   (b) *increment i by 1.*
3. *When Step 2 is completed, we have decomposed T into k subparts.*

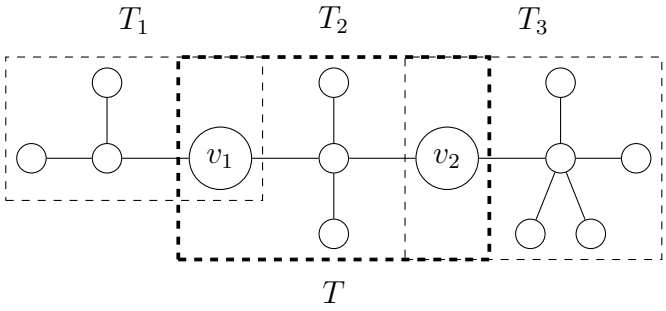
Corollary 3.7 *Suppose that the Hi-graph of a tree T consists of k disjoint components. If Algorithm 3.6 is applied and T is hyphen-decomposed into k parts, $T_1, T_2, \cdots, T_k$, then*

$$N(T) = \prod_{i=1}^{k} N(T_i).$$

Proof. This is a direct result of Theorem 3.5 and the construction of Algorithm 3.6. □

Example 3.8 For the tree T shown below that has a three-component Hi-graph, v_1 and v_2 are identified as the two hyphens. We then apply hyphen decomposition to T and get T_1, T_2, and T_3 as the resulting components. By Corollary 3.7,

$$N(T) = N(T_1)N(T_2)N(T_3) = \binom{3}{2} \times \binom{4}{2} \times \binom{5}{2} = 180.$$



In this example, the resulting components are all stars. We will describe the calculation of $N(T)$ for trees with a connected Hi-graph in the next section.

4 Trees with Single-Component Hi-graphs

We are now able to calculate $N(T)$ for a tree T with a Hi-graph of multiple components when given $N(T_i)$'s for all its components resulting from hyphen decomposition. In this section, we will look only at trees with a connected Hi-graph and show how to enumerate $N(T)$.

First, there are certain simple trees of which we can calculate $N(T)$ without having to decompose further. An example is **generalized stars**.

Definition 4.1 *A **generalized star** is a tree with at most one HDV.*

Proposition 4.2 *Let T be a generalized star with d branches. Then,*

$$P(T) = d - 1 \quad \text{and} \quad N(T) = \binom{d}{2}.$$

Proof. In order to minimize the number of paths used in a path cover for T, one of the paths includes two of the branches as well as the central vertex. The other $(d-2)$ branches are included as disjoint paths in the MPC. Therefore, $P(T) = d - 1$. Because there are $\binom{d}{2}$ ways to select two branches to lie on the same path in an MPC, $N(T) = \binom{d}{2}$. □

For a tree T with a single-component Hi-graph that is not a generalized star, there are ways to decompose it into smaller pieces, $T_1, T_2, \cdots, T_k$, enumerate $N(T_i)$ for each piece, and then calculate $N(T)$ given rules for recombination. **Absent-edge decomposition**, a process defined in Lemma 4.3, plays an important role when decomposing such trees.

Lemma 4.3 *Absent-edge decomposition is the process where a tree T is decomposed into smaller components $T_1, T_2, \cdots, T_k$ through the removal of all of its absent edges. Then,*

$$P(T) = \sum_{i=1}^{k} P(T_i) \quad \textit{and} \quad N(T) = \prod_{i=1}^{k} N(T_i).$$

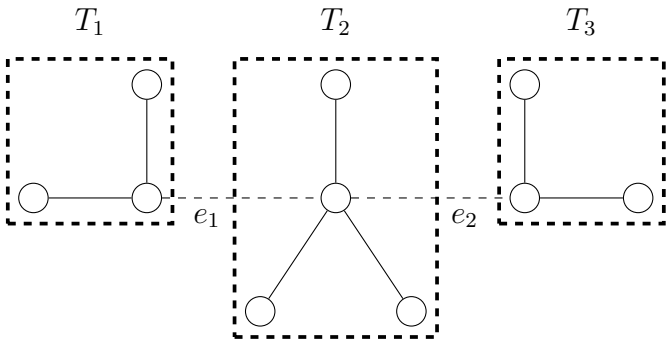
Proof. The absent edges of T are, by definition, not used in any MPC of T. Through removing all of them, an MPC of T can be viewed as a union of the MPC's of $T_1, T_2, \cdots, T_k$. Therefore, $P(T) = \sum_{i=1}^{k} P(T_i)$. The choice of which MPC of $\boldsymbol{\mathcal{P}}(T_i)$ to use for constructing an MPC for T is independent across different T_i's. Thus, we can write $\boldsymbol{\mathcal{P}}(T)$ as the product of $\boldsymbol{\mathcal{P}}(T_1), \boldsymbol{\mathcal{P}}(T_2), \cdots, \boldsymbol{\mathcal{P}}(T_k)$,

$$N(T) = |\boldsymbol{\mathcal{P}}(T)| = |\boldsymbol{\mathcal{P}}(T_1)||\boldsymbol{\mathcal{P}}(T_2)| \cdots |\boldsymbol{\mathcal{P}}(T_k)| = \prod_{i=1}^{k} N(T_i).$$

□

Example 4.4 For the tree below, e_1 and e_2 are absent. Therefore, we apply absent-edge decomposition and decompose T into T_1, T_2, and T_3. By Theorem 2.3,

$$P(T) = \sum_{i=1}^{3} P(T_i) = 1 + 2 + 1 = 4 \quad \text{and} \quad N(T) = \prod_{i=1}^{3} N(T_i) = 1 \cdot \binom{3}{2} \cdot 1 = 3.$$



Our goal now is to identify all absent edges in a given tree.

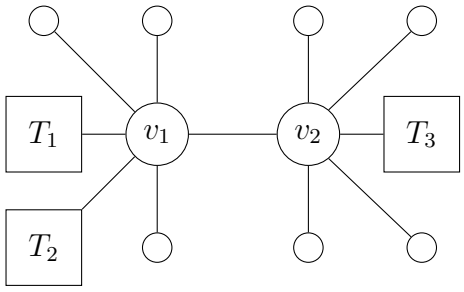
4.1 Identifying Absent Edges

We first use $H(T)$ and incremental degrees to directly identify some absent edges for a tree T.

Proposition 4.5 *In a tree T, an edge between two HID vertices is an absent edge.*

Proof. Suppose that in a tree T with a connected Hi-graph, there is one or more pairs of adjacent HID vertices. We complete the proof by contradiction.

Assume that in T, the edge between two adjacent HID vertices, v_1 and v_2, is used in an MPC $\mathcal{C}$. We use (v_1, v_2) to denote the edge and l to denote the path in $\mathcal{C}$ that contains v_1, v_2, and (v_1, v_2). Suppose that $\delta(v_1) = d_1$ and $\delta(v_2) = d_2$, with $d_1, d_2 \geq 2$, and that $P(T) = k$. Because $H(T)$ is connected, v_1 and v_2 have d_1 and d_2 pendent paths in T.



The tree shown above demonstrates a sample structure of T. The nodes denoted as T_i's represent the HDV's (besides v_1 and v_2) adjacent to either v_1 or v_2 and the subtrees connected to them. In order to minimize the number of paths used, l must include one neighboring edge at v_1 and one at v_2 besides (v_1, v_2). These two edges at v_1 and v_2 can either connect them with a pendent path or an adjacent HDV depending on the overall structure of T. Either way, there is at least one pendent path left at each vertex that is included in $\mathcal{C}$ separately. A new MPC, $\mathcal{C}'$, can be constructed from $\mathcal{C}$. The process is as follows.

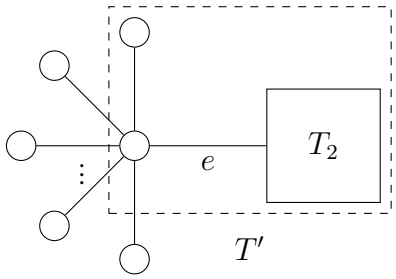
The path, l, first gets split into two by removing (v_1, v_2) from $\mathcal{C}$. Then, we connect the path that contains v_1 to one of its "available" pendent paths and apply the same to v_2. During this process, the total number of paths increases by 1 when (v_1, v_2) is removed and decreases by 2 when v_1 and v_2 are respectively connected to the two pendent paths that were originally separate paths in $\mathcal{C}$. Therefore, $|\mathcal{C}'| = k + 1 - 2 = k - 1 < P(T)$. We have now reached a contradiction. Therefore, (v_1, v_2) is absent in T. □

Not all absent edges are between two HID vertices. In order to identify the rest of them, we reduce a tree to a smaller one without changing the statuses of the edges with the help of **pendent generalized stars**. A pendent generalized star in a tree T is a generalized star induced by a **peripheral HDV** and its pendent paths. An HDV v is peripheral if and only if there is exactly one branch of T at v that contains all the other HDVs in T.

Lemma 4.6 *In a tree T, an edge connecting a pendent generalized star to the rest of T is never required.*

Proof. Let e be an edge connecting a pendent generalized star of incremental degree d to the rest of T, T_2. Assume that e is a required edge. Then, in every MPC of T, there is always a path that includes e, the central HDV of the pendent generalized star, as well as one of its pendent paths in order to minimize the number of paths used. Suppose that in an MPC of T, other than the path that uses e, there are k paths that cover the rest of T_2. Then, $P(T) = k + 1 + (d - 1) = k + d$.

We now construct a new path cover for T, where we use the same set of paths to cover T_2, except that the path that originally included e now terminates at the vertex in T_2 neighboring e. This means that there are still $(k + 1)$ paths covering T_2. We then need $(d - 1)$ paths to cover the pendent generalized star by including two of its pendent paths and the central vertex in the same path and the rest of the pendent paths as separate ones. This new path cover of T uses $(k+1+d-1) = k+d$ paths, which is equal to $P(T)$. We have reached a contradiction, meaning that e must not be a required edge. □



Proposition 4.7 *Removing a pendent generalized star from a tree T does not change the statuses of the rest of the edges in T.*

Proof. A pendent generalized star is arbitrarily selected to be removed from T along with e, the edge that connects it with the rest of T. The resulting tree is denoted T_2. v denotes the vertex in T' that is the immediate neighbor of e. By Lemma 4.6, e can only be absent or discretionary.

If e is an absent edge, none of the original MPC's uses e. It is obvious that removing it does not change the statuses of edges in T_2.

Now suppose that e is discretionary. We can then partition $\boldsymbol{\mathcal{P}}(T)$, the set of all MPC's of T, into two subsets, one with all the MPC's that use e and the other with MPC's that do not use e. Suppose that the selected pendent generalized star has incremental degree k. For the subset that does not include e, removing e does not affect the structures of the MPC's, and thus the edges in T_2 have the same status as in T. In this case, we write $P(T) = P(T_2) + (k - 1)$.

For the subset of $\boldsymbol{\mathcal{P}}(T)$ where e is always used, the path that includes e in each of the MPC's also goes through a pendent path of the pendent generalized star to minimize the number of paths used. Let T' denote the subtree of T induced by e, T_2, and the pendent path. Then, each MPC in $\boldsymbol{P}(T)$ is consisted of $k - 1$ paths as well as an MPC in $\boldsymbol{P}(T')$. Thus, $P(T) = (k - 1) + P(T') = P(T_2) + (k - 1)$. We then get $P(T') = P(T_2)$.

If there is an MPC in $\boldsymbol{P}(T')$ where e is not used, the aforementioned pendent path must be included as a separate path. We get $P(T') = P(T_2) + 1$ as a result, which is

contradictory to the result in the previous paragraph. This suggests that e must be a required edge in T'. Therefore, when the pendent generalized star and e are removed from T, the statuses of the edges left remain unchanged. □

The **internal tree**, $I(T)$, of a tree T is the subtree induced by the vertices and edges in T when all of its pendent generalized stars as well as the edges connecting them to the rest of T are removed.

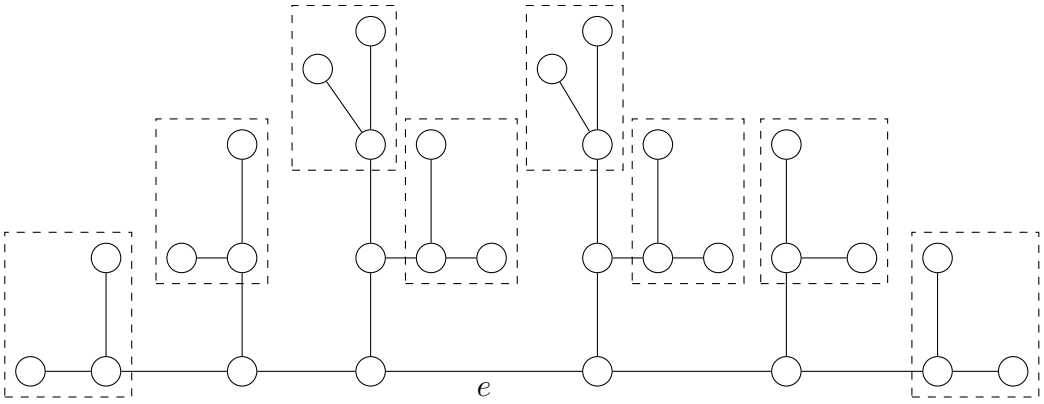
Corollary 4.8 *For a tree T and its internal tree $I(T)$, the edges that are included in both trees share the same statuses.*

For a tree T where no absent edge can be immediately identified, we can now reduce it to a smaller tree by removing a pendent generalized star and check if Proposition 4.5 is applicable. The following algorithm results in identifying all absent edges for a tree T.

Algorithm 4.9 *A tree T is given.*

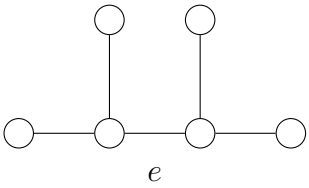
1. *Let $E_{absent} = \{\}$ denote the set of absent edges in T and set $F = T$.*
2. *Let $T_1, \cdots, T_m$ denote the disjoint components in F. While there exists a connected component in F that has more than one HDV:*
   (a) *For $T_i \in F$, if T_i has only one HDV, we remove T_i from F since it does not include an absent edge. Otherwise, T_i has two or more HDV's. We update F and proceed to the next step.*
   (b) *Let E denote the set of all edges in F. Iterate over all edges in E. If an edge e_i is between two HID vertices in the connected tree T_i that it belongs to, set $F = F - e_i$ and add e_i to E_{absent}.*
   (c) *Apply the following steps to each of the $T_i's$. Initially, T_i is at the 0th iteration and is represented using $T_i^{(0)}$.*
      i. *During the t-th iteration, every edge in $T_i^{(t)}$ is checked for whether it is absent. In order to determine the status for an edge e_{ij} of $T_i^{(t)}$, remove all pendent generalized stars of $T_i^{(t)}$ as well as the edges that connect the pendent generalized stars to the rest of $T_i^{(t)}$. However, if e_{ij} itself is in a pendent generalized star, that generalized star does not get removed.*
      ii. *$T(i)^{(t+1)}$ is obtained after removing all the pendent generalized stars of $T_i^{(t)}$. F is updated accordingly when newly identified absent edges are removed. Repeat the entire process in Step 2 on $T(i)^{(t+1)}$ until the initial condition becomes unsatisfied.*
3. *At the completion of the previous steps, all absent edges in T are included in E_{absent}.*

Example 4.10 Determine whether e is an absent edge for the following tree T.



We first attempt to identify any edge that is between two HID vertices. Since there is no such edge in T, we proceed to Step 2(c) of Algorithm 4.9.

Since our goal is to identify whether e is absent, all pendent generalized stars of T as well as the edges that connect them to the rest of T are removed. The resulting tree is shown below. Since it still includes more than one HDV, we repeat Step 2 of Algorithm 4.9 on the updated tree and find that e is now between two HID vertices. We conclude that e is an absent edge in the original tree T. In fact, e is the only absent edge in T, which can be verified by applying the algorithm thoroughly on T.



5 Prime Trees

In this section, we will present an algorithm to enumerate $N(T)$ for trees that cannot be further decomposed using hyphen decomposition or absent-edge decomposition.

Definition 5.1 *A tree T is **prime** if $H(T)$ is connected and there are no absent edges in T.*

Lemma 5.2 *For a prime tree T, an edge e connecting a pendent generalized star to the rest of T is discretionary.*

Proof. Since T is a prime tree, e must not be absent. By Lemma 4.6, e must not be required. Therefore, e is a discretionary edge. □

Lemma 5.3 *Let T be a prime tree with one or more pendent generalized stars. If an edge e connects a pendent generalized star of incremental degree k to the rest of T, and if T' is the resulting subtree after removing any $k-1$ pendent paths of the pendent generalized star from T, then e is a required edge in T'.*

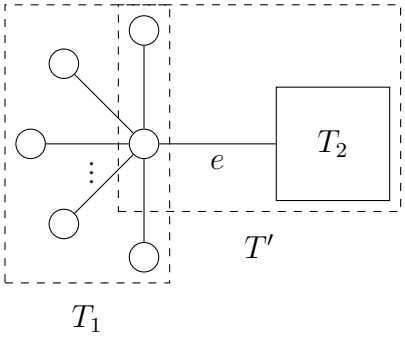
Proof. Included in the proof for Proposition 4.7. □

Algorithm 5.4 *In order to enumerate MPC's for a prime tree T, we first identify an edge e connecting a pendent generalized star, T_1, with the rest of T. By Lemma 5.2, e is discretionary. We then consider partitioning $\boldsymbol{\mathcal{P}}(T)$ into two subsets where e is either always used or never used. For the subset $\boldsymbol{\mathcal{P}}_N(T)$, where e is not used, consider the two trees T_1 and T_2 as the result of removing e from T. We have $|\boldsymbol{\mathcal{P}}_N| = N(T_1) \times N(T_2)$.*

Now consider the subset $\boldsymbol{\mathcal{P}}_U(T)$, where e is always used. In this case, in order to minimize the number of paths used, for every MPC in $\boldsymbol{\mathcal{P}}_U(T)$, the path that includes e must also go through the central vertex of T_1 as well as one of its pendent paths. We construct a subtree T' through removing $(k-1)$ pendent paths of the pendent generalized star from T. There are $\binom{k}{k-1} = k$ ways of doing so. For each of the k ways, we only count the number of MPC's of T' that use e to be consistent with our setup. By Lemma 3.3, e is required in T', so $N(T')$ is exactly the number of MPC's of T' that use e. The resulting trees, regardless of which $(k-1)$ paths are removed, are all automorphic to one another and thus have the same number of MPC's, $N(T')$. Therefore, $|\boldsymbol{\mathcal{P}}_U| = k \cdot N(T')$.

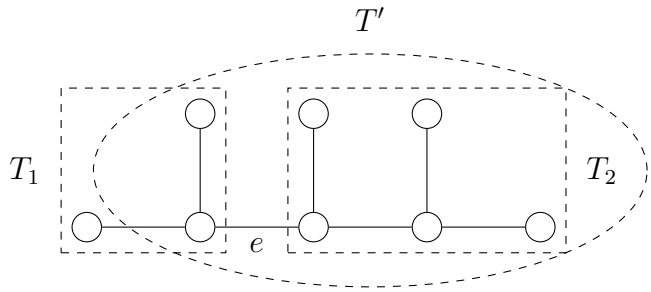
Finally, we have

$$N(T) = |\boldsymbol{\mathcal{P}}(T)| = |\boldsymbol{\mathcal{P}}_N(T)| + |\boldsymbol{\mathcal{P}}_U(T)| = N(T_1) \cdot N(T_2) + k \cdot N(T').$$



Since T_1, T_2, and T' are all on fewer vertices than T, the values $N(T_1), N(T_2)$, and $N(T')$ can also be known inductively with the same algorithm.

Example 5.5 Calculate $N(T)$ for the following prime tree T.



$|\boldsymbol{\mathcal{P}}_N(T)| = N(T_1) \cdot N(T_2) = 1 \cdot 3 = 3$; $|\boldsymbol{\mathcal{P}}_U(T)| = k \cdot N(T') = 2 \cdot 1 = 2$;
$N(T) = |\boldsymbol{\mathcal{P}}(T)| = |\boldsymbol{\mathcal{P}}_N(T)| + |\boldsymbol{\mathcal{P}}_U(T)| = 3 + 2 = 5$.

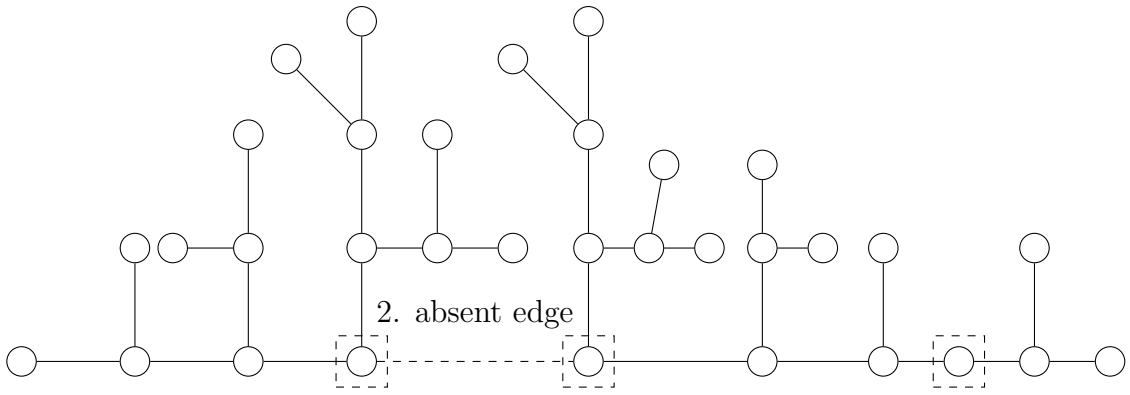
6 A General Algorithm

Algorithm 6.1 *Given a tree T,*

1. *Apply hyphen decomposition using Algorithm 3.6 to decompose T into components with connected Hi-graphs.*
2. *For each of the resulting components, identify all absent edges and apply absent-edge decomposition.*
3. *For each of the resulting components, repeat Step 1 and 2 since new hyphens and absent edges may arise after the decomposition processes. Keep a record of every decomposition applied and the components involved for Step 5. When all the resulting components become prime, go to Step 4.*
4. *Use Algorithm 5.4 to calculate the number of MPC's for prime trees inductively.*
5. *Use the values obtained from Step 4 and recombine them one step at a time using Corollary 3.7 and Lemma 4.3 to eventually obtain $N(T)$.*

The time complexity of Algorithm 6.1 is, in the worst case, quadratic. However, the algorithm tends to perform significantly better than quadratic when applied to trees with relatively simple structures. Further analysis is needed to understand the algorithm's average runtime across various tree structures. The code that implements Algorithm 6.1 is available by contacting the authors.

Example 6.2 Solve $N(T)$ for the following tree T using Algorithm 6.1.



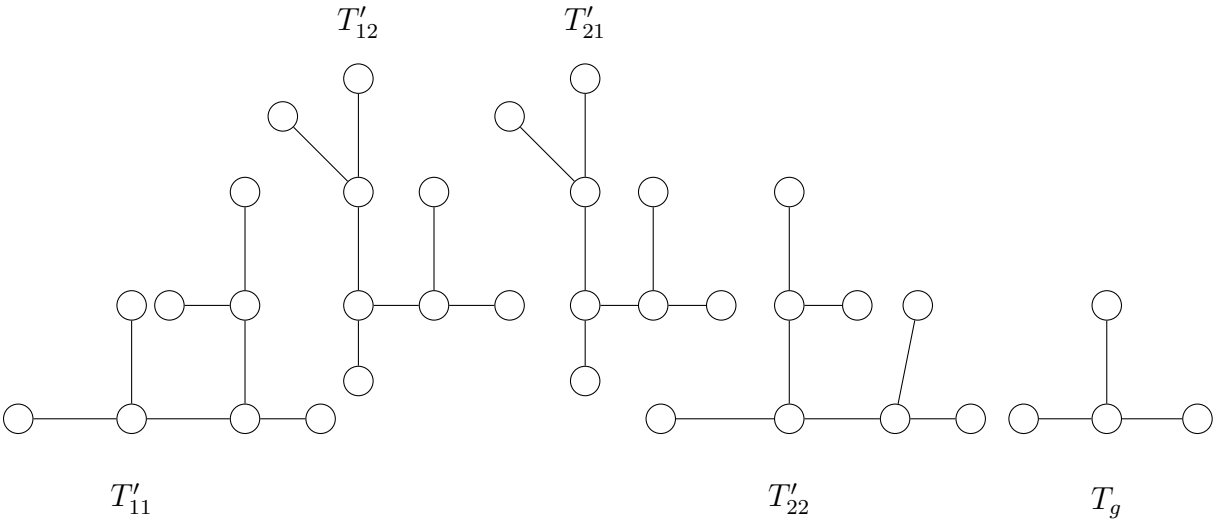
1. Apply hyphen decomposition on T and obtain two subparts. One of them is a generalized star, and we use Proposition 4.2 to get $N(T_g) = \binom{3}{2} = 3$. Let T' denote the other resulting part. By Corollary 3.7,

$$N(T) = 3N(T').$$

2. Apply absent-edge decomposition on T' and obtain T'_1 and T'_2. By Lemma 4.3, $N(T') = N(T'_1)N(T'_2)$. We now have

$$N(T) = 3N(T'_1)N(T'_2).$$

3. Repeat steps 1 and 2 in Algorithm 6.1 for each of the resulting components. Two newly arisen hyphens are identified, for which hyphen decomposition is applied. No new absent edge is found. The resulting parts after steps 1 through 3 are shown below.



Eventually, we have

$$N(T) = 3N(T'_{11})N(T'_{12})N(T'_{21})N(T'_{22}).$$

Notice that T_{11}, T_{12}, T_{21}, and T_{22} are all isomorphic to the tree in Example 5.5. All the components are now prime trees. Thus,

$$N(T_{11}) = N(T_{12}) = N(T_{21}) = N(T_{22}) = 5.$$

Finally, we have

$$N(T) = 3N(T_{11})N(T_{12})N(T_{21})N(T_{22}) = 3 \times 5^4 = 1875.$$

In the following two figures, we plot the average and maximum values of $N(T)$ for all trees on n vertices over the value of n, based on a small sample of values for n.

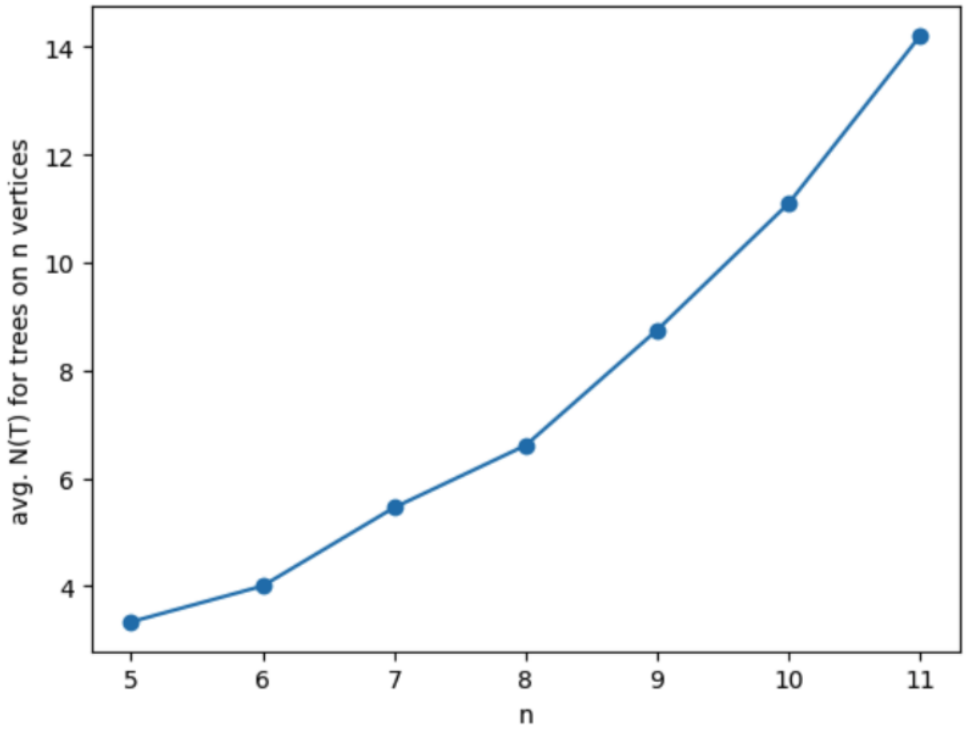


Figure 1: Growth of average $N(T)$

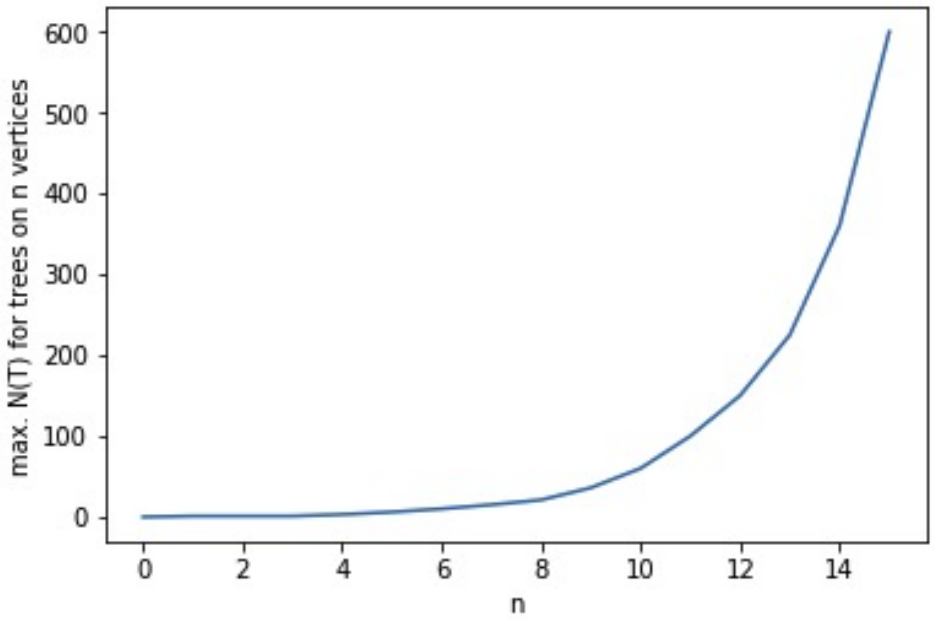
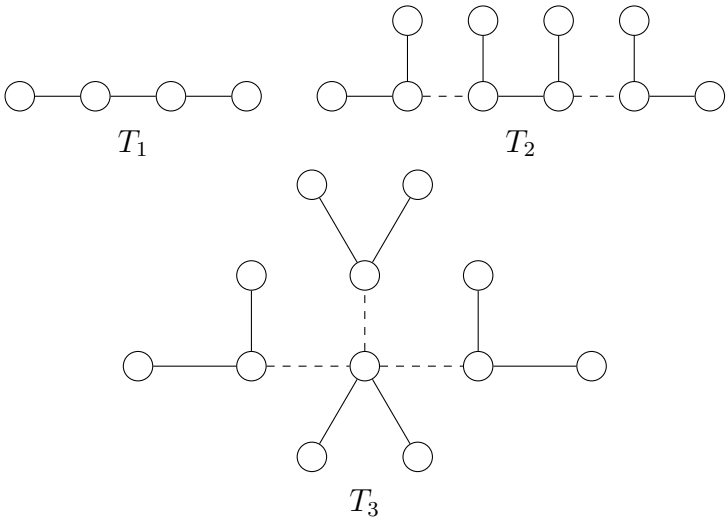


Figure 2: Growth of max $N(T)$

Appendix A Trees with a Unique MPC

One problem that has been of interest is characterizing all trees that have a unique MPC. However, among these trees, there still exist a variety of structures. Using the techniques we develop, it can be determined whether a given tree T has a unique MPC. However, we have not been able to directly describe all trees with a unique MPC using overt combinatorial features of T.

Here, we present some examples of trees with unique MPC. For trees T_1, T_2, and T_3, $P(T_1) = 1, P(T_2) = 3$, and $P(T_3) = 4$. The dashed edges are absent, and all the other edges are required.



Acknowledgments

This work was supported in part by the National Science Foundation under Grant DMS #1757603.

References

[1] C.R. Johnson, C.M. Saiago, *Eigenvalues, Multiplicities, and Graphs*, Cambridge University Press, 2018.

[2] C.R. Johnson, A. Leal-Duarte, The maximum multiplicity of an eigenvalue in a matrix whose graph is a tree, *Linear Multilinear Algebra*, **46** (1999), 139–144.

[3] I.J. Kim, B.L. Shader, Smith normal form and acyclic matrices, *J. Algebraic Combin.*, **29** (2009), 63–80.

[4] L. Hogben, C.R. Johnson, Path Covers of Trees, unpublished note.

[5] C.R. Johnson, C.M. Saiago, Estimation of the maximum multiplicity of an eigenvalue in terms of the vertex degrees of the graph of a matrix, *Electron. J. Linear Algebra*, **9** (2002), 27–31.

[6] C.R. Johnson, C.M. Saiago, The trees for which maximum multiplicity implies the simplicity of other eigenvalues, *Discrete Math.*, **306** (2006), 3130–3135.

Charles R. Johnson
Department of Mathematics, William & Mary
P.O. Box 8795
Williamsburg, VA 23187
E-mail: crjmatrix@gmail.com

Merielyn Sher
Department of Mathematics, William & Mary
P.O. Box 8795
Williamsburg, VA 23187
E-mail: merielyn.sher@gmail.com

Received: March 23, 2024 **Accepted**: October 6, 2024
Communicated by Maria Isabel Bueno Cachadina