algorithms

MDPI

Article

Stirling Numbers of Uniform Trees and Related Computational Experiments

Amir Barghi [1,*,†] and Daryl DeFord [2,†]

[1] Department of Mathematics and Statistics, Saint Michael's College, Colchester, VT 05439, USA
[2] Department of Mathematics and Statistics, Washington State University, Pullman, WA 99163, USA
* Correspondence: abarghi@smcvt.edu
† These authors contributed equally to this work.

Abstract: The Stirling numbers for graphs provide a combinatorial interpretation of the number of cycle covers in a given graph. The problem of generating all cycle covers or enumerating these quantities on general graphs is computationally intractable, but recent work has shown that there exist infinite families of sparse or structured graphs for which it is possible to derive efficient enumerative formulas. In this paper, we consider the case of trees and forests of a fixed size, proposing an efficient algorithm based on matrix algebra to approximate the distribution of Stirling numbers. We also present a model application of machine learning to enumeration problems in this setting, demonstrating that standard regression techniques can be applied to this type of combinatorial structure.

Keywords: Stirling numbers for graphs; cycle covers; random algorithms; classification problems

Citation: Barghi, A.; DeFord, D. Stirling Numbers of Uniform Trees and Related Computational Experiments. *Algorithms* **2023**, *16*, 223. https://doi.org/10.3390/a16050223

Academic Editor:Jesper Jansson

Received: 1 February 2023
Revised: 30 March 2023
Accepted: 13 April 2023
Published: 27 April 2023

1. Introduction

In a recent paper [1], we showed that several global graph statistics applied to trees of fixed order realize their extrema at paths and stars as the two opposite extremes. In addition to radius and diameter, these statistics include global degree centrality, global closeness centrality, and global betweenness centrality. Moreover, as we define below, graphical Stirling numbers of the first kind for trees and graphical factorials for trees also follow a similar pattern. We discuss algorithms for computing these values and generating the cycle covers, as well as introducing randomized methods for approximating these constructions.

In this paper, we also apply statistical learning methods to an enumerative combinatorial problem, using the algorithms mentioned above to generate the training sets. Applications of statistical learning and machine learning techniques in combinatorics is a recent development that can be very useful for problems such as the one we are considering in this paper where the underlying enumeration problem is #P complete [2,3]. One of our main objectives is to train models on trees of fixed order with global graph statistics as predictors and Stirling numbers of the first kind as the response variables. We then compare different models based on their performance on a test set. The other objective is to use these graph statistics to classify trees as "path-like" or "star-like". To this end, we will use a polynomial, introduced by Liu [4], that uniquely characterizes unrooted and unlabeled trees—we refer to these polynomials as distinguishing polynomials.

Outline

The organization of the paper is as follows: We discuss the mathematical background for the paper in Section 2. This is followed by computational algorithms for exact and probabilistic enumeration in Sections 3 and 4, respectively. In Section 5, we use the Stirling numbers computed by the previous algorithms as inputs for statistical learning methods, demonstrating that these measurements are effective predictors of structural properties of

trees in both regression and classification tasks. Finally, in Section 6 we discuss potential future applications of this work.

2. Mathematical Preliminaries

Let G be a graph on n vertices. The k-th Stirling number of the first kind of G, denoted by $\begin{bmatrix} G \\ k \end{bmatrix}$, is the number of partitions of G into exactly k disjoint cycles, where a single vertex is a 1-cycle, an edge is a 2-cycle, and cycles of order three or higher have two orientations. The graphical factorial of G, denoted by $G!$, is the total number of such decompositions without any restrictions on the number of cycles involved, i.e., $G! = \sum_k \begin{bmatrix} G \\ k \end{bmatrix}$. For results regarding the Stirling numbers of the first kind and graphical factorials for families including paths, cycles, complete bipartite graphs, and fans, among others, see articles by Barghi [5] and DeFord [6]. These results were initially motivated by a combinatorial model of seating rearrangements presented by Honsberger [7] and further analyzed by Kennedy and Cooper [8] and Otake, Kennedy, and Cooper [9].

We showed [1] that for any tree T on n vertices and any integer $n \geq k \geq \lceil \frac{n}{2} \rceil$,

$$\begin{bmatrix} S_n \\ k \end{bmatrix} \leq \begin{bmatrix} T \\ k \end{bmatrix} \leq \begin{bmatrix} P_n \\ k \end{bmatrix},$$

where S_n and P_n are the star and path of order n. Consequently, we have

$$S_n! \leq T! \leq P_n!.$$

We now define global closeness and between centralities. Let G be a connected graph. For a vertex v, closeness is defined as

$$\mathrm{cls}_G(v) = \frac{n-1}{\sum_{u \in V} d(u,v)},$$

where $d(u,v)$ is the distance between u and v. The global closeness centrality, which is a Freeman centrality measure [10,11], can be defined as

$$C_{\mathrm{cls}}(G) = \frac{\sum_{i=1}^{n} (\mathrm{cls}_G(v^*) - \mathrm{cls}_G(v_i))}{H},$$

where $n = |V(G)|$, v^* is a vertex in G such that $\mathrm{cls}(v^*) = \max_{v \in V} \mathrm{cls}(v)$, and H is the maximum value the nominator of $C_{\mathrm{cls}}(G)$ realizes for all connected graphs of order n.

For a vertex v in a graph G, the betweenness centrality is defined as

$$\mathrm{btw}(v) = \frac{1}{\binom{n-1}{2}} \sum_{u,w \in V - v} \frac{P(u,w;v)}{P(u,w)},$$

where $P(u,w)$ is the number of shortest paths from u to w and $P(u,w;v)$ are such paths that go through v. We divide by $\binom{n-1}{2}$ to normalize this centrality measure. The global betweenness centrality, which is a Freeman centralization [10,11], is defined as

$$C_{\mathrm{btw}}(G) = \frac{\sum_{i=1}^{n} (\mathrm{btw}_G(v^*) - \mathrm{btw}_G(v_i))}{H},$$

where $n = |V(G)|$, v^* is a vertex in G such that $\mathrm{btw}_G(v^*) = \max_{v \in V} \mathrm{btw}_G(v)$, and H is the maximum value the nominator of $C_{\mathrm{btw}}(G)$ realizes for all connected graphs of order n. In the case of $C_{\mathrm{btw}}(G)$ it does not matter whether we use a normalized or nonnormalized definition for $C_{\mathrm{btw}}(v)$.

We showed [1] that for any tree T of order n, where $n \geq 2$,

$$C_{\mathrm{cls}}(P_n) \leq C_{\mathrm{cls}}(T) \leq C_{\mathrm{cls}}(S_n)$$

and

$$C_{\text{btw}}(P_n) \leq C_{\text{btw}}(T) \leq C_{\text{btw}}(S_n).$$

For more information on different centrality measures, especially in the context of social networks, see a paper by Borgatti [12].

We now define distinguishing polynomials for trees. First, we need to define them for rooted unlabeled trees. For a rooted tree T_r, where r is the root, the primary subtree is a subtree S of T_r such that S has the same root as T_r and every leaf of T_r is either a leaf of S or is a descendant of a leaf of S. For a primary subtree S of T_r, we define $Q(S; x, y) = x^{\alpha} y^{\beta}$, where α is the number of leaves of S that are leaves in T_r and β is the number of leaves of S that are internal vertices in T_r. By convention, the root in a rooted tree is considered an internal vertex even though its degree might be one. The polynomial $P(T_r; x, y)$, which we call a distinguishing polynomial, is defined as $\sum_S Q(S; x, y)$, where the sum is over all primary trees of T_r. Liu [4] shows that $P(.; x, y)$ is a complete isomorphism invariant for rooted unlabeled trees.

For an unrooted and unlabeled tree T, the distinguishing polynomial $P(T; x, y)$ is the product of $P(T_1; x, y), \ldots, P(T_l; x, y)$, where l is the number of leaves in T and T_i is a rooted tree obtained from T by contracting an edge incident with a leaf and declaring the resulting vertex the root of T_i. Liu [4] also shows that $P(.; x, y)$ is an isomorphism invariant polynomial for unrooted and unlabeled trees.

One way to define a total ordering on trees of order n using $P(.; x, y)$ is to order them by evaluating $\log_{10}(P(T; x, y))$ at appropriate values of x and y. In this approach, we find $x = \mu_n$ for which $P(T; \mu_n, 1) \neq P(T'; \mu_n, 1)$ for any unrooted and unlabeled trees T, T' of order n such that $T \ncong T'$. We call this method of ordering trees of a fixed order, the evaluation-based total ordering and the two extremes of this total ordering are realized at the path and star of order n. In this paper, we use this approach to classify trees of order n by evenly dividing the associated evaluation-based total ordering into two classes and identifying the class containing P_n and S_n as "path-like" and "star-like", respectively. For a more detailed discussion of distinguishing polynomials, see papers by Liu [4] and Barghi and DeFord [1].

3. Exact Computations

In this section, we describe algorithms for the exact enumeration of Stirling numbers of the first kind on trees. We note that using the loop-erased random walk algorithm of Wilson [13], we can generate uniform spanning trees on n vertices beginning with K_n. This allows us to empirically estimate the distribution of Stirling numbers of the first kind for these trees as well as the expected distributions of several common metrics studied on graphs. We use these values to inform our parameter selection and classification of trees below. Additionally, as mentioned in Section 1, trees of a fixed size interpolate between being path-like and star-like with respect to many graph metrics. To sample more efficiently from these extremes, it is possible to implement a weighted version of the cycle basis walk on spanning trees, and the autocorrelation of this model is analyzed in the first section of the Supplementary Information. The Supplementary Information is posted on our corresponding GitHub repository for this paper; for a link to this repository, see our Data Availability Statement.

In addition to uniform trees, it is also possible to efficiently sample uniform cycle covers for bipartite, planar graphs, building on the method of Jerrum and Sinclair [14,15] for sampling uniform perfect matchings. This is described in Algorithm 1 below. Next, we observe that while computing the k-Stirling numbers of the first kind can be computationally intensive, computing the graph factorial, which is the sum of these numbers over k, can be done with a single matrix determinant for planar graphs using the FKT algorithm described in Section 1.2 and Theorem 1.4 in a book by Jerrum [16]. This algorithm, introduced in a paper by Kasteleyn [17], counts the number of perfect matchings in a planar graph by constructing a Pfaffian orientation in polynomial time. This is a simple example of the permanent-determinant method [18]. By modifying the Pfaffian with polynomial entries

we can compute the number of 2-cycles that appear in the cycle cover as the coefficient of x^k. We provide implementations of these algorithms in our corresponding GitHub repository and present examples in the section below motivating conjectures for uniform trees.

Algorithm 1: Uniform Cycle Cover

Input: A planar, bipartite graph $G = (A, B)$
Output: A uniform cycle cover of G
Select: two uniform perfect matchings M_1 and M_2 on G
Initialize: $M = \{\}$ **for** *edge (a,b) in* M_1 **do**
| Add (a,b) to M
end
for *edge (a,b) in* M_2 **do**
| Add (b,a) to M
end
Return: M

Theorem 1. *Algorithm 2 returns the number of k−matchings of T across all* $0 \leq k \leq |V(T)|$.

Proof. The permanent adjacency matrix of a tree T counts the cycle covers of T. These correspond to matchings since the only possible cycle lengths are 1 and 2. Let $A(T)$ be the adjacency matrix of T. We note that viewing $A(T)$ as a biadjacency matrix gives $T \square P_2$, which is planar since T is outerplanar. Thus, we can apply the FKT algorithm to $T \square P_2$ to obtain a Pfaffian orientation of $A(T \square P_2)$. This means that the number of perfect matchings in the product can be computed as the square root of this number, which is exactly the number of cycle covers that we were trying to compute. □

Algorithm 2: Tree Stirling

Input: A tree T
Output: The number of k-matchings of T for all k
Construct $H = T \square P_2$
Orient the adjacency matrix A of G with FKT
Multiply the non-diagonal elements of the signed matrix by x
Compute $P = \text{Det}(A)$
Return: $\sqrt{P}$

There is also an algebraic method for computing these values by representing the terms with a symbolic determinant.

Theorem 2. *The determinant of* $ixA(T) + I_n$ *returns the number of k−matchings of T across all* $0 \leq k \leq |V(T)|$ *as the coefficient of* x^{2k}.

Proof. A non-zero term in the determinant of $ixA(T) + I_n$ consists of a set of diagonal elements S counted with a weight of positive one and a perfect matching of $T \backslash S$ where each term collects a weight of $-x$. The product of these terms is then equal to x^{2k} where k is the number of edges of T that occur in the matching. Summing up over all permutations of the nodes in the determinant gives that the coefficient of x^{2k} is the desired number of k-matchings. We note that a similar argument shows that the determinant matrix of $xA(T) + I_n$ with the upper-triangular portion negative also returns the same values as coefficients and the determinant of the unsigned matrix $xA(T) + I_n$ returns the values with signs according to the parity of k. □

Although the above results show that the problem of enumerating the number of cycle covers of a given size can be completed in polynomial time in the number of nodes in the tree, actually generating the cycle covers themselves is a more difficult problem.

The reason that computing the complete list of cycle covers for graphs is computationally taxing is that even for trees and forests we need to rely on a modified version of the classic deletion-contraction algorithm described in Chapter 2 of [19] and Chapter 1 of [20], which we call the deletion–inclusion algorithm. In the deletion–inclusion algorithm, for an edge e in a forest F, we either delete e or we contract e and remove all the other edges incident with the endpoints of e. Since this is a binary branching algorithm, we continue with this process in each branch until there are no edges left to remove in the said branch. If the order of the empty graph at the end of a branch is k, it contributes to $\begin{bmatrix} F \\ k \end{bmatrix}$. The advantage of the deletion–inclusion algorithm to the uniform cycle cover is that it produces the set of all cycle covers of F while the running time of the two algorithms, at least for small enough trees, is comparable.

Algorithm 3: Deletion-Inclusion Step

Input: A forest F
Output: A contracted edge E and two sub-forests F_1, and F_2 with $|E(F_i)| < |E(F)|$
Select: A uniformly random edge $E = (i, j) \in E(F)$
Compute: $I(E) = \{e \in E(F) | i \in e \text{ or } j \in e\}$
Remove and Contract: $F_1 = G \backslash E$
Contract $F_2 = (G \backslash I(E))/E$ **Return:** E, F_1, F_2

Algorithm 4: Deletion-Inclusion

Input: A tree T
Output: Compatible edge subsets for cycle covers
Initialize: $A = \{T\}$ and $R = \{\}$
while $|A| > 0$ **do**
  for $G \in A$ **do**
    $e, F_1, F_2 =$ Deletion-InclusionStep(G)
    $A = A \backslash G$
    if $|F_1| = 0$ **then**
      $R = R \cup F_1$
    end
    else
      $A = A \cup F_1$
    end
    if $|F_2| = 0$ **then**
      $R = R \cup F_2$
    end
    else
      $A = A \cup F_2$
    end
  end
end
Return: R

Algorithm 5: Generate Cycle Covers

Input: A tree T
Output: All cycle covers of T
Initialize: $A = \{T\}$, $R = \{\}$, $C = \{\}$, and $M = \{\}$
while $|A| > 0$ **do**
  for $G \in A$ **do**
    $e, F_1, F_2 = \text{Deletion-InclusionStep}(G)$
    $A = A \backslash G$
    if $|F_1| = 0$ **then**
      $R = R \cup F_1$
    end
    else
      $A = A \cup F_1$
    end
    if $|F_2| = 0$ **then**
      $R = R \cup F_2$
    end
    else
      $A = A \cup F_2$
    end
  end
end
Return: R

We note that this generation process can be carried out in a polynomial time for fixed k and n by evaluating the $\binom{n}{n-k}$ many one-cycles for perfect matchings, which suggests that we should be mindful of letting n and k grow simultaneously, particularly when k is close to $\frac{n}{2}$.

To provide a complete implementation of the algorithms described in this section, we also included the *Python* scripts in our GitHub repository. In Figure 1, we use these methods to compute all k Stirling numbers across all isomorphism classes of trees on $n = 12$ nodes. These are representative of the values that we use in Section 5 as inputs to the statistical learning methods.

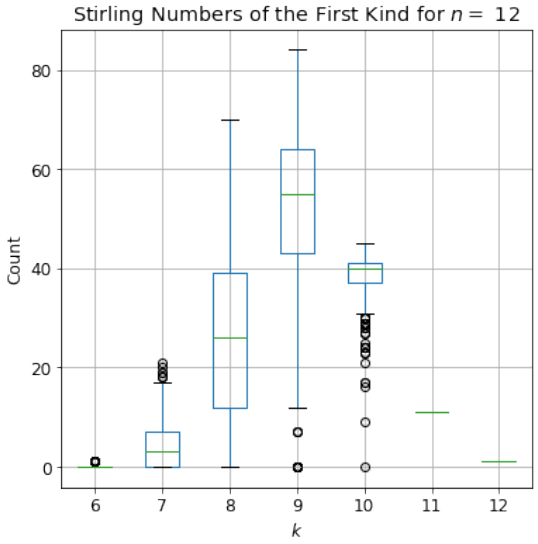


Figure 1. Full distribution of the k Stirling numbers of the first kind across all isomorphism classes of trees on 12 vertices. We focus on isomorphism classes since our intended application is to classify up to isomorphism.

4. Probabilistic Approach

Our next approach for computing Stirling numbers of the first kind is probabilistic, applying techniques from matrix algebra.

Let F be a forest and let V and E be its vertex and edge set, respectively, with c connected components. Let us assume that $|V| = n$; hence, $|E| = n - c$. Suppose $\mathcal{P}_k$ is a partition of F into k cycles. For $v \in V$, denote the set of incident edges with v by $I(v)$. For every $e_i \in E$, where $i \in \{1, \ldots, n-1\}$, define x_{e_i} as follows:

$$x_{e_i} = \begin{cases} 1, & \text{if } e_i \text{ is a 2-cycle in } \mathcal{P}_k; \\ 0, & \text{otherwise.} \end{cases}$$

With this definition in mind, $\begin{bmatrix} F \\ k \end{bmatrix} = |\{(x_{e_1}, \ldots, x_{e_{n-c}}) \mid \text{ some conditions are met}\}|$, where these conditions are as follows:

- For every $v \in V$,
$$\sum_{e_i \in I(v)} x_{e_i} = \begin{cases} 0, & \text{if } v \text{ is in a 1-cycle;} \\ 1, & \text{if } v \text{ is in a 2-cycle.} \end{cases}$$
- Subsequently, if $x_{e_i} = 1$ for some $e_i = uv \in E$, then $x_{e_j} = 0$ for all $e_j \in I(v) \cup I(u)$ and $j \neq i$.

If we denoted the set of vertices in 1-cycles and 2-cycles by A and B, respectively, then $k = |A| + |B|/2$. Please note that $A = \{v \mid \sum_{e_i \in I(v)} x_{e_i} = 0\}$, $B = \{v \mid \sum_{e_i \in I(v)} x_{e_i} = 1\}$ and $|A| + |B| = n$. It follows that $|A| = 2k - n$, $|B| = 2(n-k)$. As a result, $n \geq k \geq \frac{n}{2}$.

Let $\mathbf{A}$ be the incidence matrix for a forest F and $\mathbf{x}$ the column vector $(x_{e_1}, \ldots, x_{e_{n-c}})^T$, then $\mathbf{b} = \mathbf{Ax}$ is a $\{0,1\}$-column vector with exactly $|B| = 2(n-k)$ many 1's and $|A| = 2k - n$ many 0's. Reversing the previous discussion gives the following result that we can use to develop an approximation algorithm.

Lemma 1. *For any $\{0,1\}$-column vector $\mathbf{b}$ with $2(n-k)$ many 1's and $2k-n$ many 0's, the probability that the linear equation $\mathbf{Ax} = \mathbf{b}$ has a $\{0,1\}$-solution is:*

$$\frac{\begin{bmatrix} F \\ k \end{bmatrix}}{\binom{n}{2(n-k)}},$$

where F is the forest with incidence matrix $\mathbf{A}$.

The previous result suggests an algorithm for estimating $\begin{bmatrix} F \\ k \end{bmatrix}$ by sampling uniformly from the set of binary vectors with exactly $2(n-k)$ many 1's, and determining whether the solution to $\mathbf{Ax} = \mathbf{b}$ has binary entries.

For $1 \leq j \leq m$, let $\mathbf{b}_j$ be a $\{0,1\}$-column vector with $2(n-k)$ many 1's and $2k-n$ many 0's random entries and let $\mathbf{B} = [\mathbf{b}_j]$. Solving the equation $\mathbf{AX} = \mathbf{B}$, where $\mathbf{X} = [\mathbf{x}_j]$ is a $(n-c) \times m$ matrix of unknowns column vectors $\mathbf{x}_j$. The time complexity of applying Gaussian elimination to the augmented matrix $[\mathbf{A}|\mathbf{B}]$ is $O((n-c+m)n^2)$. Using back-substitution to find a solution $\mathbf{x_j}$ for $\mathbf{Ax}_j = \mathbf{b}_j$ is $O(n^2)$ and checking whether this solution consists of only 0s and 1s require $O(n)$ operations. With m random vectors $\mathbf{b}_j$, we need $O(mn^2)$ operations. Therefore, the overall time complexity of checking which random column vectors $\mathbf{b}_j$ yield an allowable solution $\mathbf{x}_j$ requires $O(n^3 + mn^2)$ operations, assuming c is fixed.

This naive approach requires fully solving the associated linear system and checking whether the resulting solution vectors are binary. We can circumvent this computational hurdle by instead interpreting the system as a collection of Diophantine equations, motivated by interpreting the product as occurring over the edges of the tree, rather than the vertices.

Note that

$$\sum_{e_i \in E} x_{e_i} + \sum_{e_i \in E} (1 - x_{e_i}) = \sum_{e_i \in E} 1 = n - c.$$

It follows that

$$|B|/2 + \sum_{e_i \in E} (1 - x_{e_i}) = n - c$$

and

$$n - k + \sum_{e_i \in E} (1 - x_{e_i}) = n - c.$$

Now define y_{e_i} as $1 - x_{e_i}$ for every $e_i \in E$. It follows that

$$\sum_{e_i \in E} y_{e_i} = k - c.$$

Additionally, the restrictions on x_{e_i}'s translate to the following restrictions for y_{e_i}:

- If $\sum_{e_i \in I(v)} x_{e_i} = 0$ for some $v \in V$, then $\sum_{e_i \in I(v)} y_{e_i} = \sum_{e_i \in I(v)} (1 - x_{e_i}) = d(v)$.
- If $\sum_{e_i \in I(v)} x_{e_i} = 1$ for some $v \in V$, then $\sum_{e_i \in I(v)} y_{e_i} = \sum_{e_i \in I(v)} (1 - x_{e_i}) = d(v) - 1$.

It follows that $A = \{v \mid \sum_{e_i \in I(v)} y_{e_i} = d(v)\}$ and $B = \{v \mid \sum_{e_i \in I(v)} y_{e_i} = d(v) - 1\}$, and

$$\begin{bmatrix} F \\ k \end{bmatrix} = |\{(y_{e_1}, \ldots, y_{e_{n-c}}) \mid |A| = 2k - n, |B| = 2(n - k)\}|. \tag{1}$$

In general, without any restrictions, the number of $\{0, 1\}$-solutions to the Diophantine equation

$$y_{e_1} + \cdots + y_{e_{n-c}} = k - c$$

is $\binom{n-c}{k-c}$. This implies the following result that allows for a more efficient randomized method.

Lemma 2. *Let $Y = (y_{e_1}, \ldots, y_{e_{n-1}})$ be a binary vector indexed by the edges of F. Then the probability that the positive entries of Y induce a partition of F into k cycles and hence correspond to edges that are not 2-cycles in $\mathcal{P}_k$ is*

$$\frac{\begin{bmatrix} F \\ k \end{bmatrix}}{\binom{n-c}{k-c}}.$$

As with the previous approximation result, Lemma 2 suggests an algorithm for estimating $\begin{bmatrix} F \\ k \end{bmatrix}$ by sampling uniformly from the set of binary vectors indexed by the edges of F, and determining whether they induced a partition with the proper number of cycles.

If $\mathbf{1}$ is the column vector whose entries are 1's and $\mathbf{A}$ is the incidence matrix of F, then $\mathbf{d} = \mathbf{A1}$ is column vector containing the degree sequence of F. On the other hand, if $\mathbf{y} = (y_{e_1}, \ldots, y_{e_{n-c}})^T$ satisfies the conditions in (1), then $\mathbf{d} - \mathbf{Ay} = \mathbf{A}(\mathbf{1} - \mathbf{y})$ should consist of $2(n - k)$ many 1's. If we have m many $\{0, 1\}$-column vectors $\mathbf{y}_j$ with random entries consisting of $k - c$ many 1's, then the time complexity to check whether each one satisfies the conditions in (1) is $O(n(n - c))$ and the overall time complexity is $O(mn(n - c))$, which may be more efficient than our earlier approach.

For an implementation of this approach in *Python*, on our GitHub repository, see the *Jupyter* notebook `TreeLearning-Testing_Probabilistic_Approach.ipynb`. In this notebook, a random tree T is generated, so the number of components c is one. The code generates m trials. At each trial, a $\{0, 1\}$-column vector with random entries is generated and a trial is considered a "success" if the condition that it contains exactly $k - 1$ many 1's is met. We use a Bernoulli random variable to represent the outcome of each trial. In other words, we denote the random variable representing the ith trial by

$$X_i = \begin{cases} 1 & \text{with probability } p; \\ 0 & \text{with probability } 1 - p, \end{cases}$$

where p is the probability of "success". The main objective is to estimate $\left[{T \atop k}\right]$ via simulation. Let $X = \sum_i X_i / m$. Since $p = \mathbb{E}[X] \approx \left[{T \atop k}\right] / \binom{n-1}{k-1}$, we calculate

$$\begin{bmatrix} T \\ k \end{bmatrix} - \binom{n-1}{k-1} X$$

in the code to see how well this probabilistic approach performs.

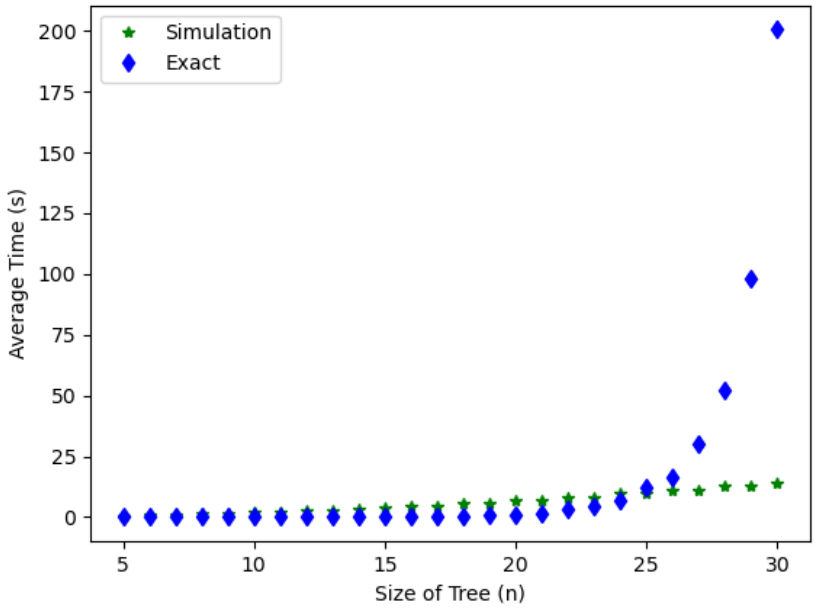
Results

To make our code reproducible, we are using a random seed. The results for n between 7 and 15 and $m = 10{,}000$ is included in Table 1. For each n, a random tree T of that order is generated and $\left[{T \atop k}\right]$ is estimated for $n - 2 \geq k \geq \lceil \frac{n}{2} \rceil$.

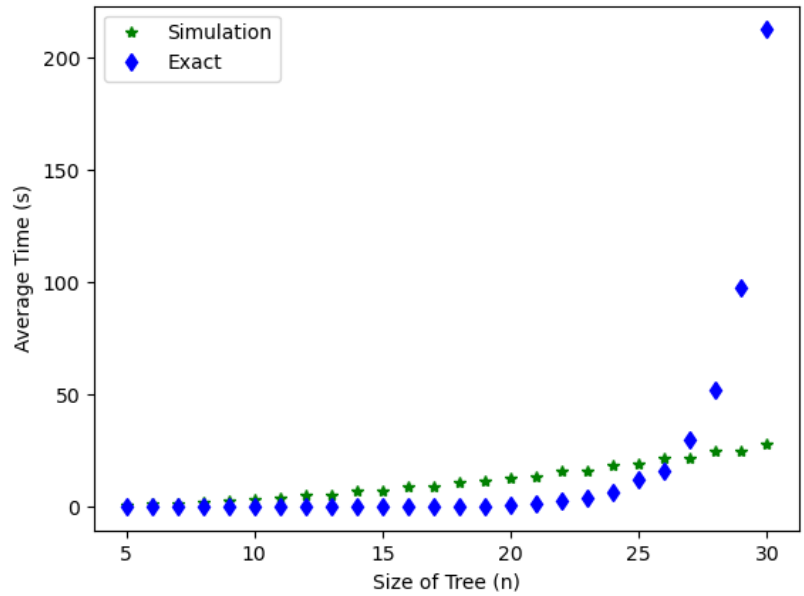
Table 1. The results of running `TreeLearning-Testing_Probabilistic_Approach.ipynb` for $7 \leq n \leq 15$ and $m = 10{,}000$. For each n, a random tree T of that order is generated and $\left[{T \atop k}\right]$ is estimated for $n - 2 \geq k \geq \lceil \frac{n}{2} \rceil$.

	k = 4	5	6	7	8	9	10	11	12	13
$n = 7$	0.1120	0.0560								
8	−0.0010	−0.3090	0.0203							
9		0.0000	0.1536	0.0300						
10		0.0000	0.1880	0.2880	0.1880					
11			0.0000	−0.7790	−0.3280	−0.1155				
12			0.0760	0.3412	−1.1470	0.9275	−0.1740			
13				0.1520	−1.9776	−1.2300	2.0180	−0.0444		
14				0.0000	−0.5208	−2.4607	2.1055	−0.5224	0.2676	
15					−0.4024	1.4715	3.8850	−5.2100	2.4296	−0.1539

One natural question is how the probabilistic approach performs compared to the exact computation of $\left[{T \atop k}\right]$. In Figure 2, the average running times for the probabilistic approach for estimating and the exact method for computing $\left[{T \atop k}\right]$ for $n - 2 \geq k \geq \lceil \frac{n}{2} \rceil$, for $5 \leq n \leq 30$, and a random tree T of order n, are visualized. As we see in Figure 2, the average running times for using the probabilistic approach grows linearly while the exact computation grows exponentially, and the average running times using exact computation take over those of the probabilistic approach when $n = 25$ for 10,000 iterations and $n = 27$ for 20,000 iterations. The code to reproduce these plots is in the following *Python* script: `probabilistic_approach-average_running_times.py`. Note that the running time for this *Python* script is long.



(**a**) 10,000 Iterations



(**b**) 20,000 Iterations

Figure 2. Average running times for the probabilistic approach for estimating and the exact method for computing $\left[\begin{smallmatrix} T \\ k \end{smallmatrix}\right]$, for $n-2 \geq k \geq \lceil \frac{n}{2} \rceil$, for $5 \leq n \leq 30$, and a random tree T of order n.

The *Python* script `probabilistic_approach-uniform_sampling.py` on the GitHub page computes the difference between the exact value of the k-th Stirling number and its approximation using the probabilistic approach discussed above for 100 trees of order n uniformly sampled using the Wilson algorithm for $n \in \{7, \ldots, 19\}$ and $n-2 \geq k \geq \lceil \frac{n}{2} \rceil$. This *Python* script returns separate `.csv` files for each n. On the other hand, the *Python* script `analysis-probabilistic_approach-uniform_sampling.py` computes the mean, standard deviation, and skewness of these differences for each n and each k. As shown in the table in the second section of the Supplementary Information, the mean of these differences stay relatively close to zero and, with the exception of a few extreme values of k, the skewness of these differences also stay relatively close to zero, indicating symmetric distributions. Since, in this experiment, we do not scale m along with n and we take uni-

form samples of size 100 for each value of n from the space of tree of order n, the standard deviations increase as n increases, as expected.

5. Statistical Learning and Enumerative Metrics on Trees

Many interactions between combinatorial analysis and modern machine and statistical learning techniques have focused on the field of combinatorial optimization [21–23]. These analyses have both applied learning techniques to generating heuristics or approximate solutions to difficult combinatorial problems [24–27], as well as motivating interesting new areas of combinatorial research [28]. Another recent area of interest is Graph Neural Networks [29,30] which use graph structures to better represent features in modern datasets. Other techniques that have motivated work between these fields include determinantal point processes [31–33] and submodular functions [34–36], which have the same property of providing solutions to difficult problems and providing interesting new avenues of study. In this paper, we do not consider an optimization approach but rather use the combinatorial structure reflected in the enumeration of Stirling numbers and other network statistics as input and training data for classification and regression.

As we discussed earlier, the k-th Stirling number of the first kind (for fixed k), global closeness centrality, and global betweenness centrality for trees exhibit a similar property in that their values varies between two extremes which are realized at paths and stars. In other words, for any tree T on n vertices and any integer $n \geq k \geq \lceil \frac{n}{2} \rceil$,

$$\begin{bmatrix} S_n \\ k \end{bmatrix} \leq \begin{bmatrix} T \\ k \end{bmatrix} \leq \begin{bmatrix} P_n \\ k \end{bmatrix},$$

$$C_{\text{cls}}(P_n) \leq C_{\text{cls}}(T) \leq C_{\text{cls}}(S_n),$$

and

$$C_{\text{btw}}(P_n) \leq C_{\text{btw}}(T) \leq C_{\text{btw}}(S_n).$$

Based on these observations, we use statistical learning tools and use Stirling numbers of the first kind for trees as predictors to make predictions about members of random sets of trees, both in the training and testing stages. It is assumed that trees in both the training and testing sets have a fixed number of vertices and there is the same number of trees in both training and testing sets. Moreover, we use both classification and regression algorithms to address this problem.

We will use three separate datasets while using statistical learning methods. The first dataset consists of all non-isomorphic trees of order 12, of which there are 551, using the `networkx.nonisomorphic_trees` function. First, we classify these trees by evenly dividing the associated evaluation-based total ordering into two classes and identifying the class containing P_n and S_n as "path-like" and "star-like", respectively. We then evenly split the trees into a train and test set at random. For reproducibility purposes, we use a random seed in our code.

In the second dataset, using a random seed, we generate 500 non-isomorphic trees of order 18 using the `networkx.nonisomorphic_trees` function. Again, we classify them as "path-like" and "star-like" by evenly dividing the associated evaluation-based total ordering. Note that a tree might be misclassified as opposed to its class if we used the complete list of non-isomorphic trees of order 18, but we expect the number of such misclassifications to be low. Again, we evenly divide these 500 trees into a train and test set at random. Because we can generate a separate test set, we will not be using cross-validation and out-of-bag error estimation in our code. Lastly, in the third dataset, we generate 500 trees of order 18 sampled uniformly from the space of such trees.

We use the following classifiers from scikit-learn library [37] in *Python*:

- DecisionTreeClassifier
- ExtraTreeClassifier
- BaggingClassifier

- RandomForestClassifier
- ExtraTreesClassifier
- SVC (Support Vector Classification)

For DecisionTreeClassifier, ExtraTreeClassifier, RandomForestClassifier, and ExtraTreesClassifier, we used both Gini and entropy criteria, which are measures of node impurity when fitting these models. We also used Minimal Cost-Complexity Pruning in these models. And to compare methods based on their performance on train and test sets, we use the following classification metrics: accuracy_score, confusion_matrix, matthews_corrcoef, and classification_report.

To estimate Stirling numbers of the first kind for trees, we will use the following regression models from scikit-learn library [37]:

- LinearRegression
- Ridge
- Lasso
- ElasticNet
- PolynomialFeatures (for degree 2 regression)
- SGDRegressor (Stochastic Gradient Descent)
- DecisionTreeRegressor
- ExtraTreeRegressor
- RandomForestRegressor
- ExtraTreesRegressor
- BaggingRegressor
- SVR (Support Vector Regression)

To measure how these methods perform on train and test sets, we use the following regression metrics: r2_score, explained_variance_score, and mean_squared_error.

5.1. Results

5.1.1. Classification

The results for tree-based and support vector classification methods with closeness, betweenness, and $\lceil \frac{T}{k} \rceil$ for $n-1 \geq k \geq \lceil \frac{n}{2} \rceil + 1$ as predictors for all non-isomorphic trees of order $n = 12$, for a sample of size 500 of non-isomorphic trees of order $n = 18$ generated randomly using `networkx.nonisomorphic_trees`, and for a sample of size 500 of trees of order $n = 18$ sampled uniformly from the space of all such trees using Wilson's algorithm are in the tables in the Supplementary Information. In this experiment, the response is class, i.e., whether a tree is "star-like" or "path-like". These results are computed by running the code in the *Jupyter* notebooks in our GitHub repository. As shown in the tables in the third and fourth sections of the Supplementary Information, tree-based classifiers (decision tree, extra tree, bagging, random forest, and extra trees) individually perform comparably on train and test sets so do support vector classifiers (linear and quadratic SVC). Additionally, across these methods, we also see comparable performances on test sets and train sets, separately, which suggests that trained models generalized nicely to unseen data, regardless of the method used. Here we are considering both accuracy scores and Matthews' correlation.

In Figure 3, we compare between train and test R^2 scores and explained variance scores (EVS) between different tree-based classification methods based on Gini and entropy criteria with and without pruning for a sample of size 500 of trees of order $n = 18$ sampled uniformly from the space of all such trees using Wilson's algorithm. As we see in Figure 3, pruning does not lead to a severe decrease in test R^2 scores and EVS and in some methods we see an increase in these scores. Of course, pruning leads to a decrease in train R^2 score and EVS, but the trade-off is that the resulting models are trees with less depth, fewer nodes, and less complexity and the same level of test performance. Moreover, using the two different sampling methods for trees of order $n = 18$, does not lead to a difference in results. In Figures A2 and A3, we do a similar comparison for all non-isomorphic trees of

order $n = 12$ and for a sample of size 500 of non-isomorphic trees of order $n = 18$ generated randomly using `networkx.nonisomorphic_trees`.

To compute the results for closeness, betweenness, and Stirling numbers of the first kind for trees as sole predictors, one may run the code in the *Jupyter* notebooks in our GitHub repository.

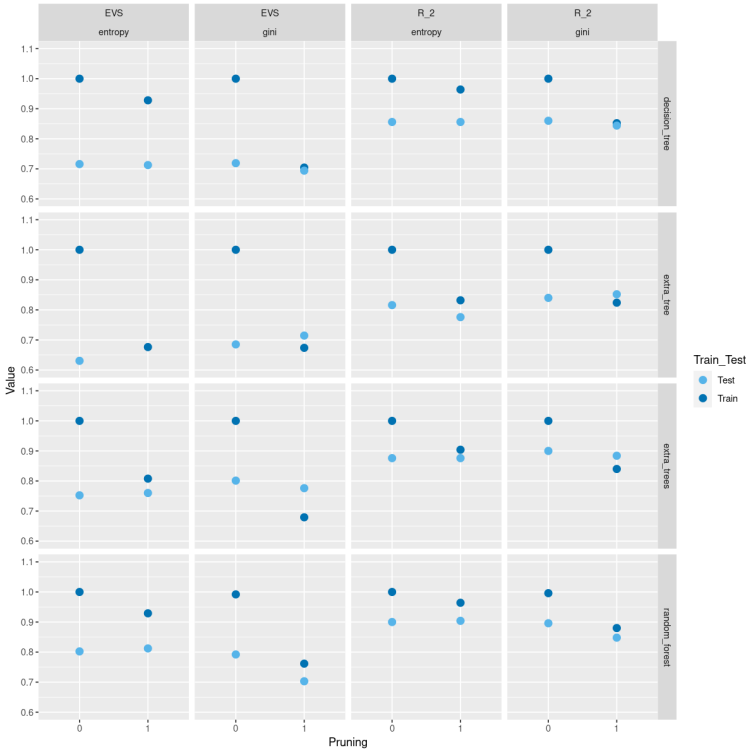
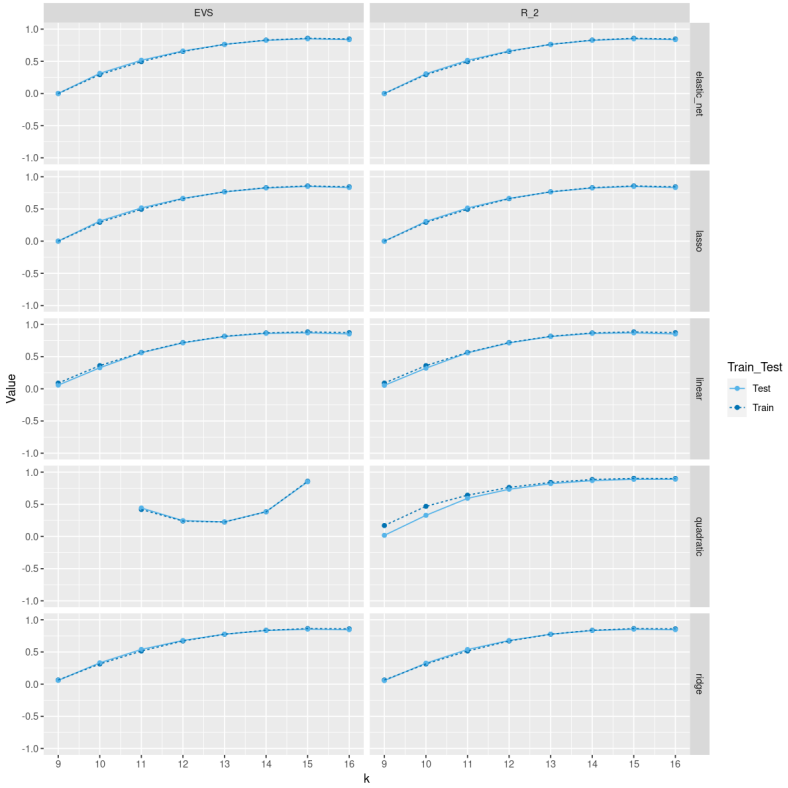


Figure 3. Train and test R^2 and explained variance scores between different tree-based classification methods based on Gini and entropy criteria with and without pruning using all predictors for 500 trees of order $n = 18$ sampled uniformly.

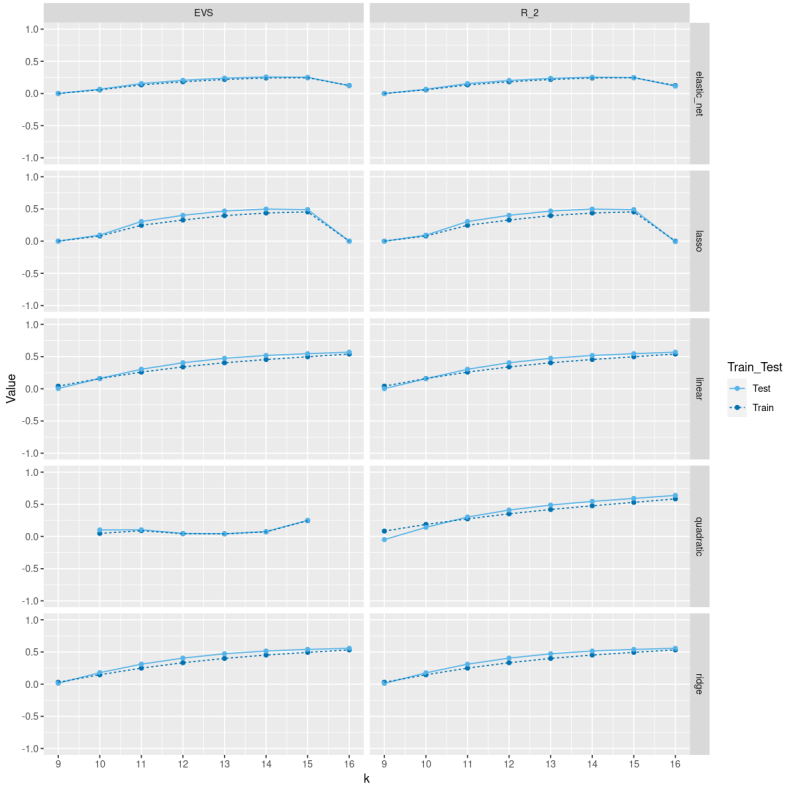
5.1.2. Regression

For regression and tree-based regression methods, we use $\log_{10}(P(T; 2, 1))$, closeness, betweenness, and class as predictors for all non-isomorphic trees of order $n = 12$, for a sample of size 500 of non-isomorphic trees of order $n = 18$ generated randomly using `networkx.nonisomorphic_trees`, and for a sample of size 500 of trees of order $n = 18$ sampled uniformly from the space of all such trees using Wilson's algorithm, respectively, to predict $\left[{T \atop k}\right]$ for $n - 1 \geq k \geq \lceil \frac{n}{2} \rceil + 1$. These results are computed by running the code in the *Jupyter* notebooks in our GitHub repository.

In Figures 4 and 5, we compare between train and test R^2 scores and explained variance scores (EVS) between different regression and tree-based regression methods (with and without pruning), respectively, for a sample of size 500 of trees of order $n = 18$ sampled uniformly from the space of all such trees using Wilson's algorithm. As we see in Figure 4, pruning does not lead to a severe decrease in test R^2 score and EVS and in some methods an increase in these scores. Of course, pruning leads to a decrease in train R^2 and EVS, but the trade-off is that the resulting models are trees with less depth, fewer nodes, and less complexity and the same level of test performance. Moreover, excluding $\log_{10}(P(T; 2, 1))$ as one of the predictors severely affects the performance of these models on train and test sets. Please note that the values in these plots are those between -0.5 and 1; specifically, values less than -0.5 are not included. In Figures A2, A3, A4, and A5, we do a similar comparison for all non-isomorphic trees of order $n = 12$ and for a sample of size 500 of non-isomorphic trees of order $n = 18$ generated randomly using `networkx.nonisomorphic_trees`. Using the two different sampling methods for trees of order $n = 18$, does not lead to a difference in results.

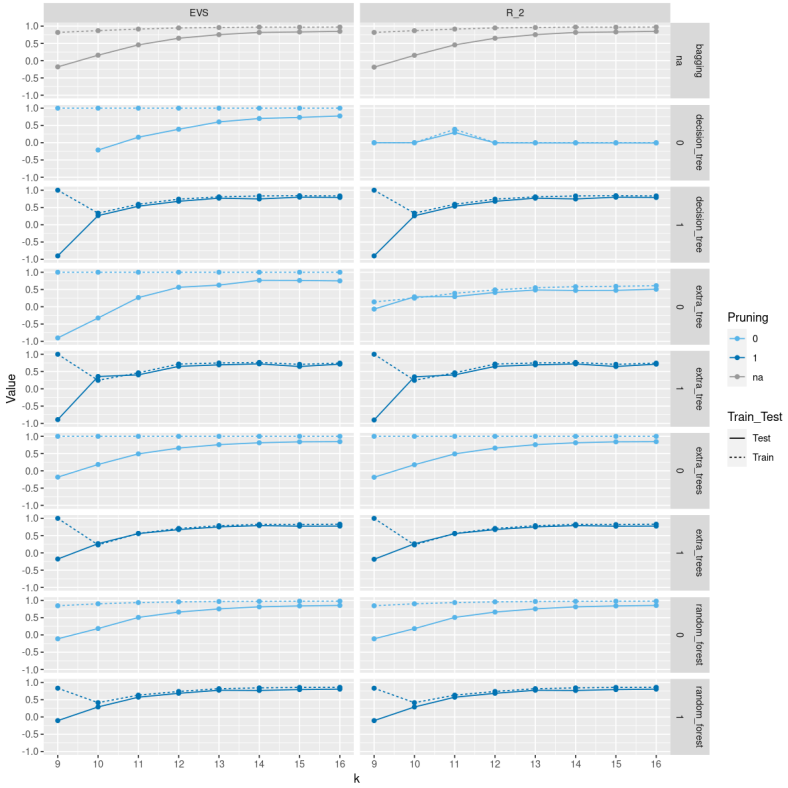


(**a**) All Predictors

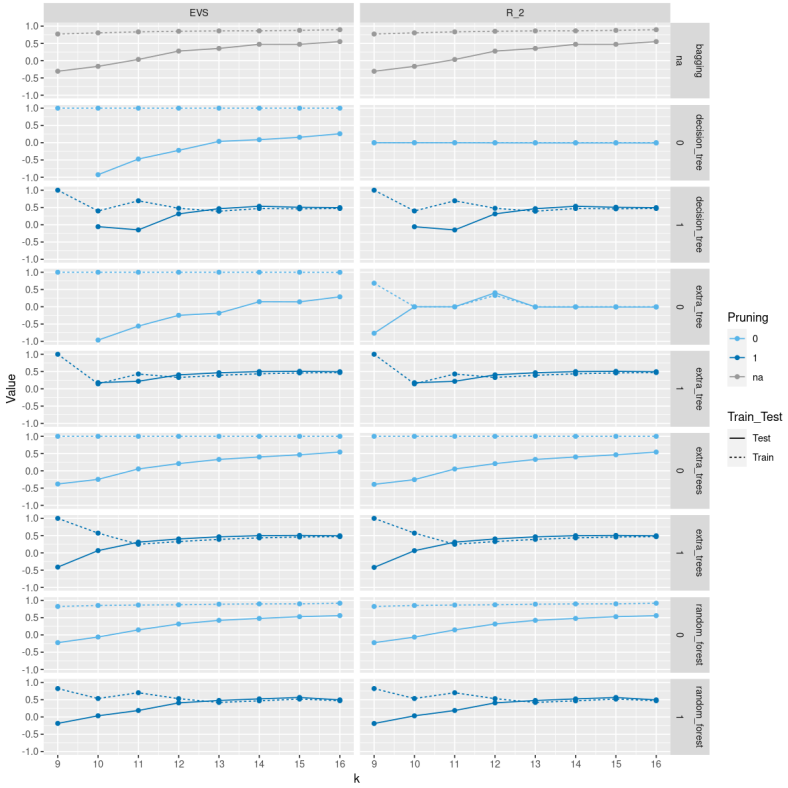


(**b**) Subsets of predictors

Figure 4. Train and test R^2 scores and explained variance scores (EVS) between different regression methods and with $\log_{10}(P(T;2,1))$, closeness, betweenness, and class as predictors for 500 trees of order $n = 18$ sampled uniformly.



(**a**) All predictors



(**b**) Subsets of predictors

Figure 5. Train and test R^2 scores and explained variance scores (EVS) between different tree-based regression methods based on Gini and entropy criteria with and without pruning and $\log_{10}(P(T;2,1))$, closeness, betweenness, and class as predictors for 500 trees of order $n = 18$ sampled uniformly.

In Figure 5, we see that pruning leads to an almost zero difference between R^2 score and EVS; for the exact values, see in the tables the fifth through seventh sections of the Supplementary Information. This indicates that the mean of residuals is almost zero. We also see that a difference between R^2 score and EVS only exists in decision tree and extra tree without pruning.

To compute the results for closeness, betweenness, and class together as predictors and $\begin{bmatrix} T \\ k \end{bmatrix}$ for $n-1 \geq k \geq \lceil \frac{n}{2} \rceil + 1$ as a response, one may run the code in the *Jupyter* notebooks in our GitHub repository.

6. Discussion and Conclusions

As computing Stirling numbers for arbitrary graphs is a difficult problem in general, we have focused here on developing and expanding methods in the case of trees of a fixed order, where it is tractable to design algorithms that can help inform the general case. These approaches can then be applied to approximation problems for denser graphs, estimating lower bounds by computing on sampled spanning trees. Our probabilistic methods for approximating Stirling numbers are also intrinsically interesting, and extending these approaches beyond the current setting is a promising avenue for future work. One potential weakness and limitation of the current formulation of these methods is the lack of a bound on the variance, which, as expected, does appear to grow with n, as shown in Table 1. Finally, our computational experiments with modern learning methods allow us to quantify the previously observed interpolation structure of trees and connect our combinatorial work to this broader class of methods. One avenue for future work is to use interpretable learning methods to directly observe which components of the graphs are best informing the statistical results as this is a limitation of the methods used in the current work. The success of these experiments suggests that statistical learning methods can be applied to a broad class of combinatorial problems in graph theory, whose solutions can be too difficult or computationally taxing to obtain using exact methods. It also serves to demonstrate that these types of combinatorial objects have detectable structures that allow us to exploit different statistical learning techniques, including others not considered here.

Author Contributions: Both authors have contributed equally to all aspects of this paper. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

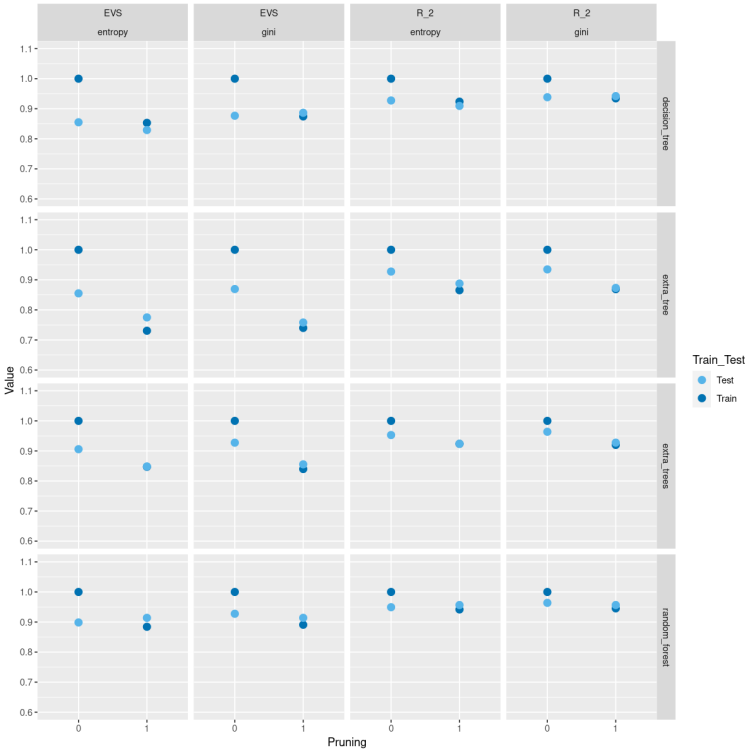
Informed Consent Statement: Not applicable.

Data Availability Statement: Replication code and data for this paper are available at https://github.com/drdeford/Computational_Experiments_on_Stirling_Numbers_of_Uniform_Trees (accessed on 12 April 2023).

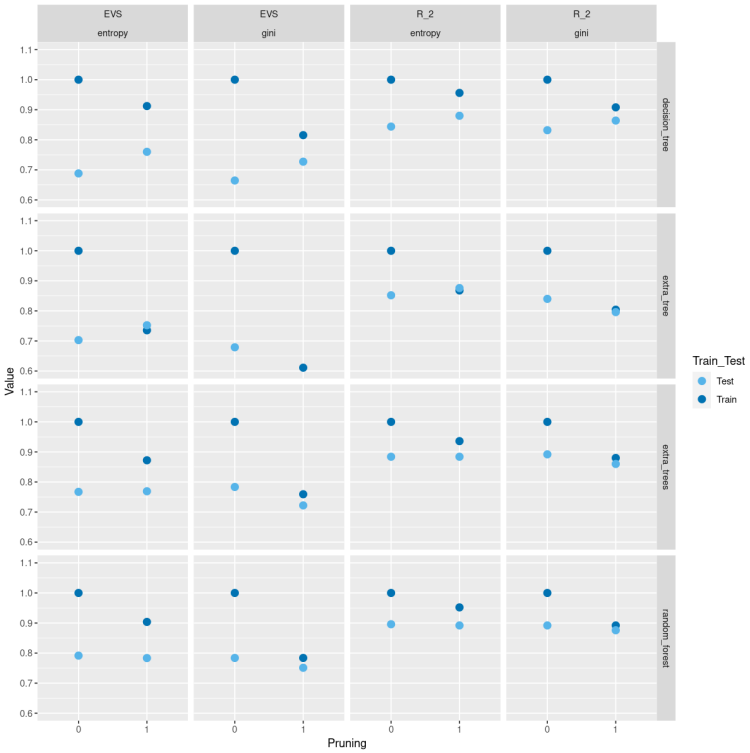
Acknowledgments: The second author would like to thank the Washington State University Department of Mathematics and Statistics and College of Arts and Sciences for their support.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Detailed Figures

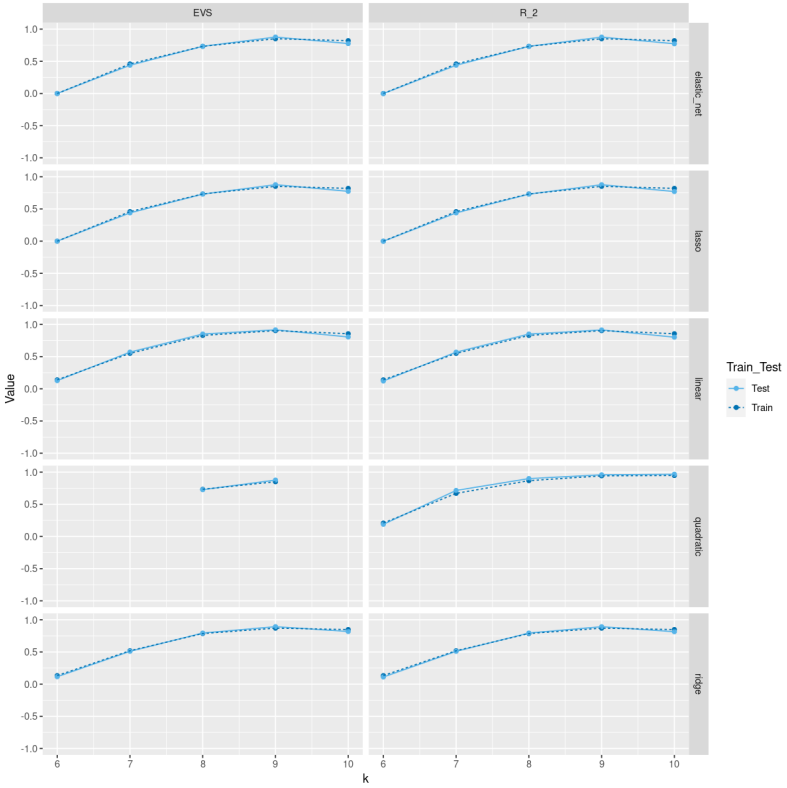


(**a**) All Predictors, $n = 12$

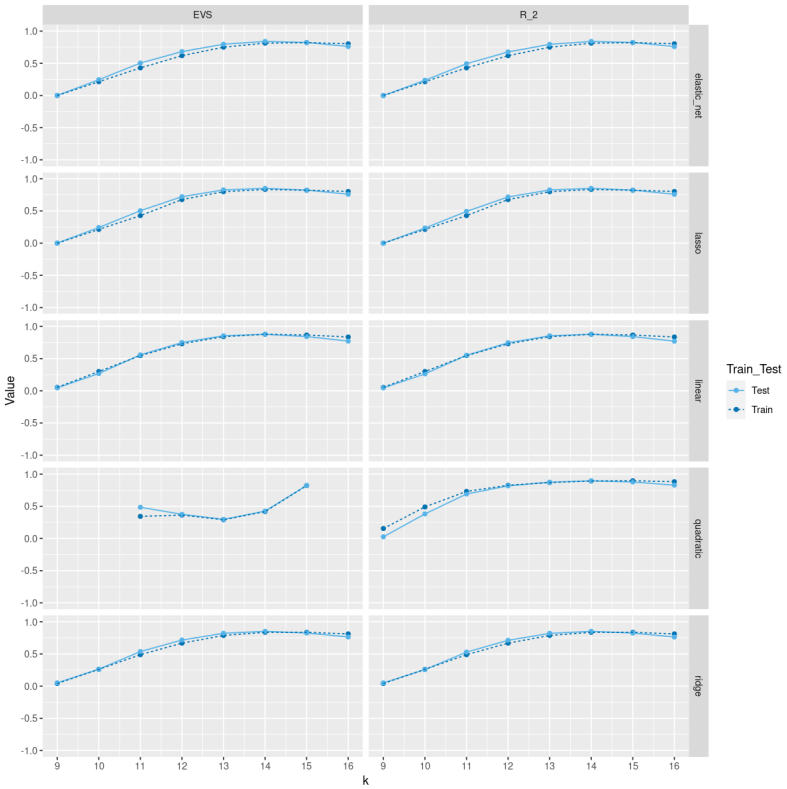


(**b**) All Predictors, $n = 18$

Figure A1. Train and test R^2 and explained variance scores between different tree-based classification methods based on Gini and entropy criteria with and without pruning.

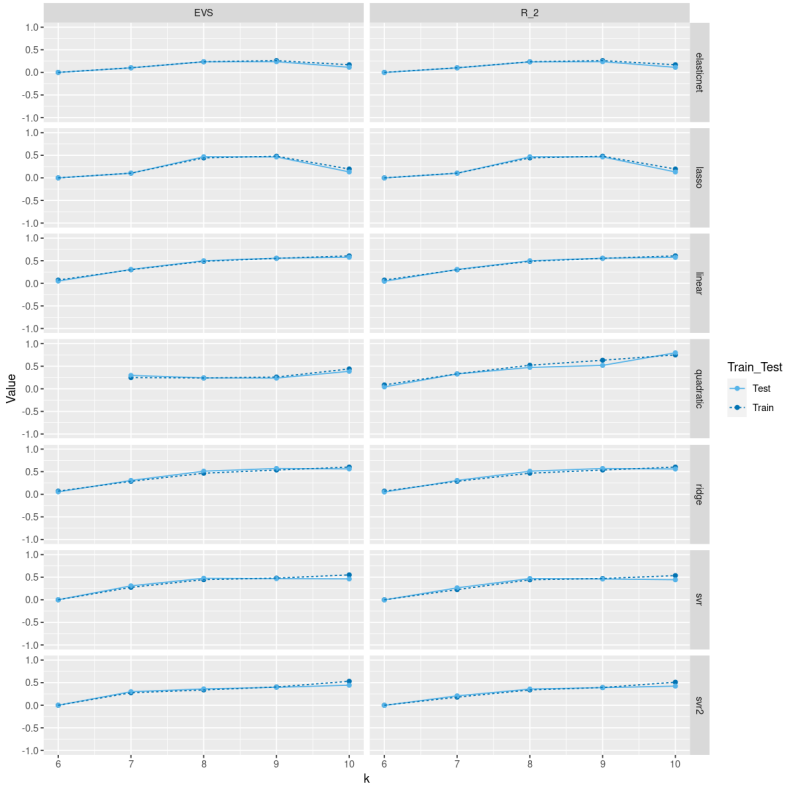


(**a**) All predictors, $n = 12$

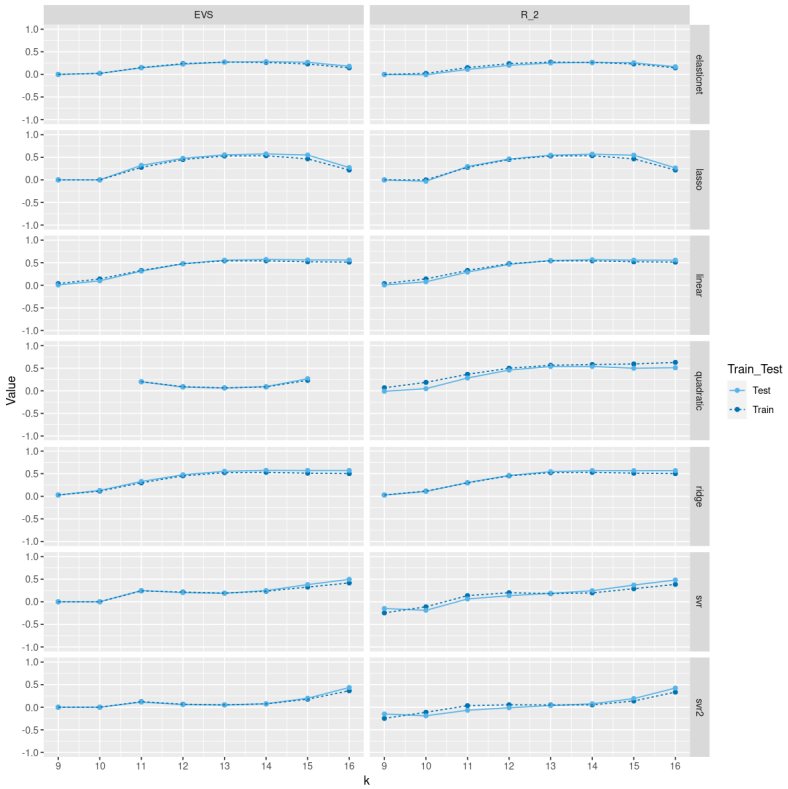


(**b**) All predictors, $n = 18$

Figure A2. Train and test R^2 scores and explained variance scores (EVS) between different regression methods and with $\log_{10}(P(T; 2, 1))$, closeness, betweenness, and class as predictors.

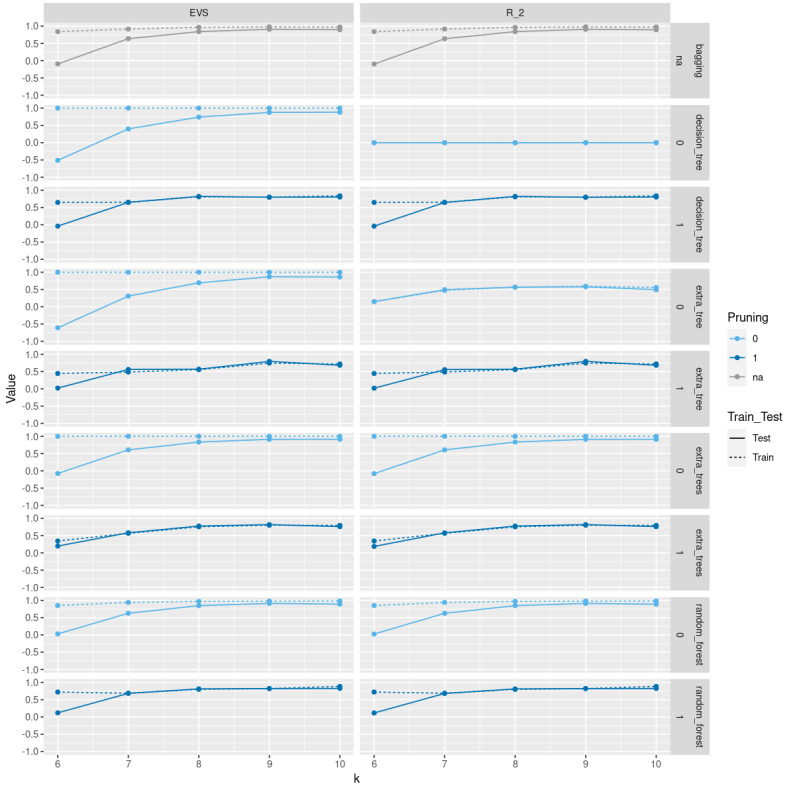


(**a**) Subsets of predictors, $n = 12$

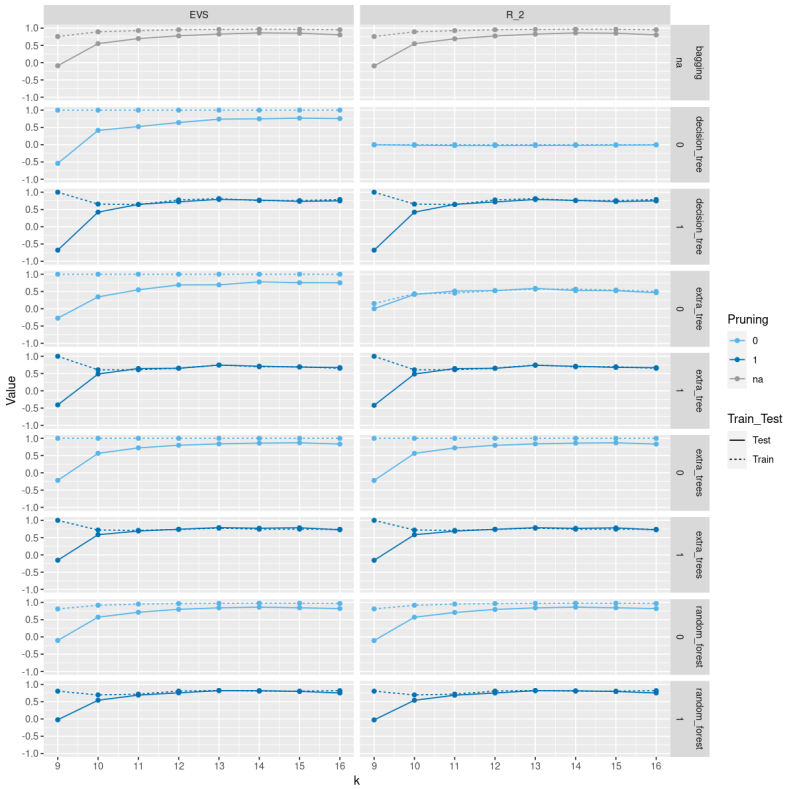


(**b**) Subsets of predictors, $n = 18$

Figure A3. Train and test R^2 scores and explained variance scores (EVS) between different regression methods and with $\log_{10}(P(T; 2, 1))$, closeness, betweenness, and class as predictors.

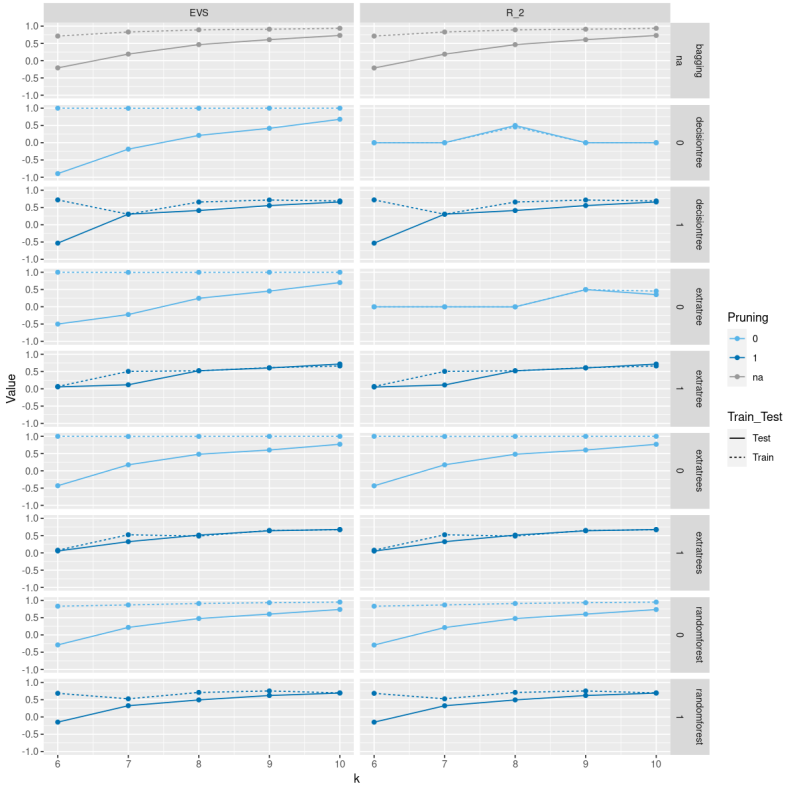


(**a**) All predictors, $n = 12$

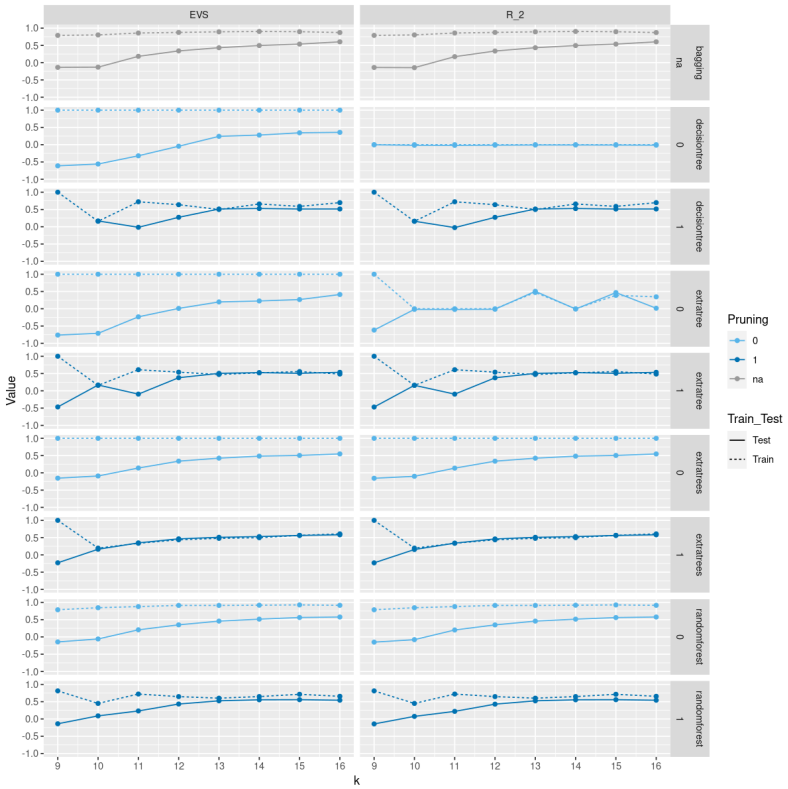


(**b**) All predictors, $n = 18$

Figure A4. Train and test R^2 scores and explained variance scores (EVS) between different tree-based regression methods based on Gini and entropy criteria with and without pruning and with $\log_{10}(P(T; 2, 1))$, closeness, betweenness, and class as predictors.



(**a**) Subsets of predictors, $n = 12$



(**b**) Subsets of predictors, $n = 18$

Figure A5. Train and test R^2 scores and explained variance scores (EVS) between different tree-based regression methods based on Gini and entropy criteria with and without pruning and with $\log_{10}(P(T;2,1))$, closeness, betweenness, and class as predictors.

References

1. Barghi, A.; DeFord, D.R. Ranking Trees Based on Global Centrality Measures. *Discret. Appl. Math.* 2022, *Submitted*.
2. Jerrum, M. Two-dimensional monomer-dimer systems are computationally intractable. *J. Stat. Phys.* **1987**, *48*, 121–134. [CrossRef]
3. Valiant, L.G. The complexity of computing the permanent. *Theor. Comput. Sci.* **1979**, *8*, 189–201. [CrossRef]
4. Liu, P. A tree distinguishing polynomial. *Discret. Appl. Math.* **2021**, *288*, 1–8. [CrossRef]
5. Barghi, A. Stirling numbers of the first kind for graphs. *Australas. J. Comb.* **2018**, *70*, 253–268.
6. DeFord, D.R. Seating rearrangements on arbitrary graphs. *Involv. A J. Math.* **2014**, *7*, 787–805. [CrossRef]
7. Honsberger, R. *In Pólya's Footsteps; Vol. 19, The Dolciani Mathematical Expositions*; Miscellaneous Problems and Essays; Mathematical Association of America: Washington, DC, USA, 1997; pp. xii+315.
8. Kennedy, R.; Cooper, C. Variations on a 5×5 seating rearrangement problem. *Math. Coll.* **1993**, *Fall–Winter*, 59–67.
9. Otake, T.; Kennedy, R.E.; Cooper, C. On a seating rearrangement problem. *Math. Inform. Q.* **1996**, 52, 63–71.
10. Freeman, L.C. Centrality in social networks conceptual clarification. *Soc. Netw.* **1978**, *1*, 215–239. [CrossRef]
11. Freeman, L.C. A set of measures of centrality based on betweenness. *Sociometry* **1977**, *40*, 35–41. [CrossRef]
12. Borgatti, S.P. Centrality and network flow. *Soc. Netw.* **2005**, *27*, 55–71. [CrossRef]
13. Wilson, D.B. Generating Random Spanning Trees More Quickly Than the Cover Time. In Proceedings of the Twenty-eighth Annual ACM Symposium on Theory of Computing, STOC '96, Philadelphia, PA, USA, 22–24 May 1996; pp. 296–303. [CrossRef]
14. Jerrum, M.; Sinclair, A.; Vigoda, E. A Polynomial-Time Approximation Algorithm for the Permanent of a Matrix with Nonnegative Entries. *J. ACM* **2004**, *51*, 671–697. [CrossRef]
15. Jerrum, M.R.; Valiant, L.G.; Vazirani, V.V. Random generation of combinatorial structures from a uniform distribution. *Theor. Comput. Sci.* **1986**, *43*, 169–188. [CrossRef]
16. Jerrum, M. *Counting, Sampling and Integrating: Algorithms and Complexity*; Birkhauser: Basel, Switzerland, 2003.
17. Kasteleyn, P. Graph theory and crystal physics. *In Graph Theory and Theoretical Physics*; Harary, F., Ed.; Academic Press: New York, NY, USA, 1967; pp. 43–110.
18. Kuperberg, G. An exploration of the permanent-determinant method. *Electron. J. Combin.* **1998**, *5*, 46, 34. [CrossRef] [PubMed]
19. Ellis-Monaghan, J.; Moffatt, I. *Handbook of the Tutte Polynomial and Related Topics*; Chapman & Hall: London, UK, 2022.
20. Dong, F.M.; Teo, K.L. *Chromatic Polynomials and Chromaticity of Graphs*; World Scientific: Singapore, 2005.
21. Yang, X.; Wang, Z.; Zhang, H.; Ma, N.; Yang, N.; Liu, H.; Zhang, H.; Yang, L. A Review: Machine Learning for Combinatorial Optimization Problems in Energy Areas. *Algorithms* **2022**, *15*, 205. [CrossRef]
22. Mazyavkina, N.; Sviridov, S.; Ivanov, S.; Burnaev, E. Reinforcement learning for combinatorial optimization: A survey. *Comput. Oper. Res.* **2021**, *134*, 105400. [CrossRef]
23. Bengio, Y.; Lodi, A.; Prouvost, A. Machine learning for combinatorial optimization: A methodological tour d'horizon. *Eur. J. Oper. Res.* **2021**, *290*, 405–421. [CrossRef]
24. Karimi-Mamaghan, M.; Mohammadi, M.; Meyer, P.; Karimi-Mamaghan, A.M.; Talbi, E.G. Machine learning at the service of meta-heuristics for solving combinatorial optimization problems: A state-of-the-art. *Eur. J. Oper. Res.* **2022**, *296*, 393–422. [CrossRef]
25. Castaneda, J.; Neroni, M.; Ammouriova, M.; Panadero, J.; Juan, A.A. Biased-Randomized Discrete-Event Heuristics for Dynamic Optimization with Time Dependencies and Synchronization. *Algorithms* **2022**, *15*, 289. [CrossRef]
26. Caro, G.A.D.; Maniezzo, V.; Montemanni, R.; Salani, M. Machine learning and combinatorial optimization, editorial. *OR Spectr.* **2021**, *43*, 603–605. [CrossRef]
27. Barrett, T.D.; Parsonson, C.W.F.; Laterre, A. Learning to Solve Combinatorial Graph Partitioning Problems via Efficient Exploration. *arXiv* **2022**, arXiv:cs.LG/2205.14105.
28. Wagner, A.Z. Constructions in combinatorics via neural networks. *arXiv* **2021**, arXiv:math.CO/2104.14516.
29. Wu, Z.; Pan, S.; Chen, F.; Long, G.; Zhang, C.; Yu, P.S. A Comprehensive Survey on Graph Neural Networks. *IEEE Trans. Neural Netw. Learn. Syst.* **2021**, *32*, 4–24. [CrossRef] [PubMed]
30. Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; Sun, M. Graph neural networks: A review of methods and applications. *AI Open* **2020**, *1*, 57–81. [CrossRef]
31. Ghosh, S.; Rigollet, P. Gaussian determinantal processes: A new model for directionality in data. *Proc. Natl. Acad. Sci. USA* **2020**, *117*, 13207–13213. [CrossRef]
32. Borodin, A. Determinantal point processes. *arXiv* **2009**, arXiv:math.PR/0911.1153.
33. Kulesza, A.; Taskar, B. *Determinantal Point Processes for Machine Learning*; Now Publishers Inc.: Hanover, MA, USA, 2012.
34. Dughmi, S. Submodular Functions: Extensions, Distributions, and Algorithms. A Survey. *arXiv* **2011**, arXiv:cs.DS/0912.0322.
35. Bilmes, J. Submodularity In Machine Learning and Artificial Intelligence. *arXiv* **2022**, arXiv:cs.LG/2202.00132.
36. Bach, F. Learning with Submodular Functions: A Convex Optimization Perspective. *Found. Trends Mach. Learn.* **2013**, *6*, 145–373. [CrossRef]
37. Pedregosa, F.; Varoquaux, G.; Gramfort, A.; Michel, V.; Thirion, B.; Grisel, O.; Blondel, M.; Prettenhofer, P.; Weiss, R.; Dubourg, V.; et al. Scikit-learn: Machine Learning in Python. *J. Mach. Learn. Res.* **2011**, *12*, 2825–2830.