University of Greifswald

Faculty of Mathematics and Computer Science

Genetic programming as a means for generating improved tree balance indices

MASTER THESIS

submitted in partial fulfillment of the requirements for the degree of Master of Science (M.Sc.) in Biomathematics by

Sophie Johanna Kersting

First supervisor: Prof. Dr. Mareike Fischer

Second supervisor: Prof. Dr. Volkmar Liebscher

Greifswald, August 13, 2020

Abstract

Balance indices are measurements of the degree of symmetry in (rooted binary) trees and are objects of current research. They can be used as statistical tests to decide if a tree is likely to have arisen by chance or if its topology was generated under a different tree development model. This can give insights amongst others on evolutionary influences in the history of species based on an analysis of their reconstructed phylogeny.

The already existing wide range of balance indices raised questions: How can we decide for as few as possible indices to avoid the consequences of multiple testing? Is there a most powerful balance index (for a specific task)? Several studies were performed to compare the statistical power of some of indices. However, there are no formulated guidelines or recommendations on when to use which balance index, yet.

In this master thesis we will perform a substantially larger comparison with a longer list of balance indices and on a large variety of alternative tree models. We will see that there are indeed groups of balance indices better suited to different alternative tree models, showing that the decision for a certain balance index could be improved with prior knowledge.

To further advance the search for an optimized balance index we will use genetic programming to travel across the field of possible balance indices in order to find a better performing individual. For this we will use two approaches with different building blocks as well as two different optimization tasks. Thereby we will discover a newly constructed balance index that has the advantage of being an "allrounder", performing well on all of the alternative models, a feature that none of the other tested established indices showed to this extent. By this we can also show that genetic programming as an optimization algorithm can be a viable tool for this particular topic in the area of phylogenetics.

Table of Contents

1 Introduction

> "Cause trees grow that way. They're not all perfect; they're like us."
>
> — Bob Ross, The Joy of Painting

For decades balance indices have been discussed as a tool to measure the degree of asymmetry in trees like genealogies or phylogenies. This can give insights into diversification rate variation [48], influences like fertility inheritance and selection [5, 30] or effects of different tree reconstruction methods [22, 24, 44]. Up to this day, new innovative approaches or recently proven properties of already established indices are frequently published.

With all their advantages as a quick measuring tool that summarizes the symmetry of a tree in just a real number, their insufficiencies have also been reported. Apart from not being derived from some model of diversification, Moore criticizes the following aspect [39, p. 6]:

> "[T]hese indices appear to capture different but poorly characterized aspects of tree shape [...]. Consequently, any attempt to test for significant diversification rate variation with these tree-balance indices must grapple with the "agony of choice" between myriad alternatives or opt to use all (or some subset of) the indices and endure issues of multiple-test correction."

This is precisely the problem that will be dealt with in this master thesis: Is there a way to reduce the amount of indices that should be tested by finding an optimized index (for a certain task)?

The first idea is to compare indices that already exist. There have already been smaller experiments (see Section 3.1) that each compared a handful of indices, but here we will generate an overall comparison on a significantly larger number, including new and well-known indices, in addition to a wider range of alternative balance influencing tree models.

The main question, however, will be if a better performing index can be constructed using genetic programming (GP). GP, an optimization strategy from the field of evolutionary algorithms, was chosen because its strategy seems to reasonably fit the structure of the problem. A balance index is seen as an individual that consists of different building blocks that are linked with different functions which can be evolved using mutation and selection. Mutation will cause a variation within a population of such individuals and selection will cause well performing individuals to produce more offspring (balance indices that inherit substructures of their parent(s)). Thus, GP offers a sensible way to travel across the space of all possible indices with the aim to find the best performing. How well GP actually performs on this problem will be evaluated in the end.

All in all, we will go along with the following steps and questions. The two main parts will both have a theoretical part in which the theoretical and mathematical basics are described.

- Comparison of established balance indices.
  - How similar are they? Are there groups of indices that perform similarly well?
  - Can the list of balance indices be shortened?

- Construction of optimized balance indices using genetic programming.

  Stage 1: Using established balance indices as building blocks

  Stage 2: Phylogenetic auxiliary functions as building blocks

  - How much better are the newly constructed indices? Is a huge optimization even possible?
  - Are the resulting indices readable and understandable or are they only an arbitrary combination of operations? Can they be simplified by hand without dropping in performance?
  - Do the new indices contain novel ideas on measuring asymmetry?
  - How well does genetic programming perform overall as a means for generating optimized balance indices?

Remark

The algorithms used in this master thesis are implemented in the free programming language `R (version 4.0.0)` [42]. The package `rgp (version 0.4-1)` [15, 16] with `emoa (version 0.5-0.1.)` [33] was used as the foundational structure for genetic programming. Other packages used are `ape (version 5.3)` [40], `phytools (version 0.7-20)` [43] as well as `data.table (version 1.12.8)` [14] for handling the tree data and `microbenchmark (version 1.4-7)` [32] to test the run time. To parallelize calculations the `doParallel (version 1.0.15)` [12] as well as the `foreach (version 1.5.0)` [34] package were used.

Due to the fact that the implementation of several algorithms, tree models and balance indices is a significant part of this thesis, some pseudo code and an overview of the scripts will be given and explained for the main topics. The complete R-scripts can be found on the attached CD.

The figures were made with R, GeoGebra and ClickCharts. The latter unfortunately does not support LaTeX-code for mathematical symbols and was therefore used for overviews that presented normal text with only a few mathematical terms.

2 Trees, tree models and balance indices

In this thesis balance indices will be applied on *rooted binary trees* $T(V, E)$ with a finite set of vertices or nodes V and edges E. A tree is an acyclic connected graph. The *degree* of a node is defined as the number of connected or *incident* edges. Connected nodes are called *adjacent*. Nodes with a degree ≤ 1 are *leaves* V^1, the remaining vertices are called the *inner nodes* $\mathring{V}$. Similarly $\mathring{E}$ denotes the subset of E that contains all *inner edges* which are not incident to a leaf. A *path* is a sequence of distinct nodes $(a_1, ..., a_m)$ with a_i and a_{i+1} being adjacent for $i = 1, ..., m-1$. In a tree there is a unique path that connects each pair of nodes.

A tree is *binary* if all inner nodes have degree 3. Furthermore, a tree is *rooted binary* if there additionally is exactly one distinct inner node ρ with only degree 2, which is the *root* of the tree. As the introduction of a root induces a natural direction of the edges, trees are normally depicted with the root at the top and all leaves at the bottom as shown in Figure 1 [49, p. 3-6]. A node v is an *ancestor* of node w (*descendant*) if $v \neq w$ and v lies on the unique path of w to the root. If v and w are additionally adjacent they are also called *(direct) parent* and *(direct) child*. A *pending subtree* rooted in v is the subtree induced by v and all of its descendant nodes.

A *phylogeny* or *phylogenetic X-tree* $\mathcal{T} = (T, \phi)$ consists of a tree T that is called the *topology* or *tree shape* of $\mathcal{T}$ and a bijection ϕ from the set of labels or species X to V^1, the set of leaves of T [46, p. 19].

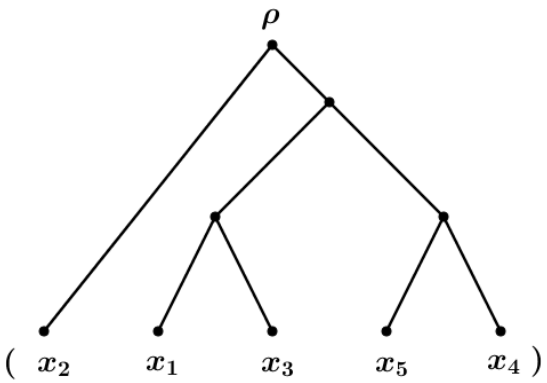


Figure 1: Example of a rooted binary tree with $n = 5$ leaves. A possible phylogeny with an assignment of the species $x_1, ..., x_5$ to the leaves is depicted as well.

Let $T = (V, E)$ be a rooted binary tree with n leaves, then we have [49, p. 10]:

$$\begin{aligned} |V| &= 2n-1, \;\; |\mathring{V}| = n-1 \\ |E| &= 2n-2, \;\; |\mathring{E}| = n-2 \end{aligned} \tag{2.1}$$

Throughout this master thesis we will group tree shapes by their number of leaves $n = |V^1| \in \mathbb{N}$, which we will also refer to as *size* of the tree. A tree is called larger than another tree, if it has the larger size. Let RB_n with $n \in \mathbb{N}$ denote the set of rooted binary trees (or trees shapes) with n leaves.

Further definitions and functions on trees especially for the established balance indices will be given in the following list for a quick overview.

$\delta(x)$ The depth of a vertex x is defined as the number of edges on the unique path from x to the root [46, p. 22].

$\kappa(v)$ The number of descendant leaves of an interior vertex v. A leaf is sometimes seen as its own descendant, thus for a leaf v, $\kappa(v)$ is normally set to 1.

$lca_T(v, w)$ The last common ancestor of two vertices v, w is defined as the vertex with the highest depth that is on both unique paths from v and from w to the root.

$s(T)$ The number of symmetry nodes of T. An interior vertex v is a *symmetry node* if the two pending subtrees rooted in the direct children of v have the same tree shape.

$c(T)$ The number of cherries of T. A *cherry* is a pair of leaves that have the same direct parent.

There are two special tree shapes that should be shortly mentioned. Let T_n^{cat} denote the so-called *caterpillar tree* with n leaves that is defined as the unique rooted binary tree shape with only one cherry. Second, the *fully balanced tree* T_k^{bal}, which is only defined for $n = 2^k$ with $k \in \mathbb{N}$, is characterized through the inner nodes that are all symmetry nodes and split their number of descendant leaves in half. Thus, this tree shape is often considered the most symmetrical and every leaf x has depth $\delta(x) = log_2(n) = k$.

2.1 The Yule model

Tree models define a system of rules on how to generate a (binary) tree usually for a given number of leaves n. There are different approaches like gradual clustering of the leaves, splitting the descendant leaves into two groups for every interior node starting with the root or building a tree from a single node by splitting leaves successively into two new leaves. In this master thesis we will use the latter approach, as we can use it effectively to create basic intuitive models for evolutionary influences (further discussion in the corresponding paragraph in Section 3.1).

Remark. *A huge field of application are phylogenies that represent the evolutionary history of species. Therefore, we will also sometimes use the corresponding terms (e.g. "speciation event" for the event of a leaf splitting into two new leaves). It goes without saying that all tree models and explanations can also be applied to other topics like genealogies, family trees or search trees.*

One important model which describes the easiest form of evolutionary development is the Yule model. It models the idea that at any point in time there is a finite number of species, with one primordial species at time zero, and from time to time there are speciation events in which a species splits into two different species. The probability that a species is affected by such an event is equal for all currently existing species.

Definition 2.1 (Yule model). The *Yule model* – also known as the *equal-rates-Markov-model (ERM)* or *random branching* – is based on a population development model that describes a pure birth process with no deaths. The size of a population in this context is the number of currently living species. Each species has birth rate λ (evolutionary rate) that illustrates how likely the species will split up (see Figure 2 on the left).

One of various algorithms to generate a topology under the Yule model is the following: Start with a single node (the root). Continuously choose a leaf from the currently existing leaves and

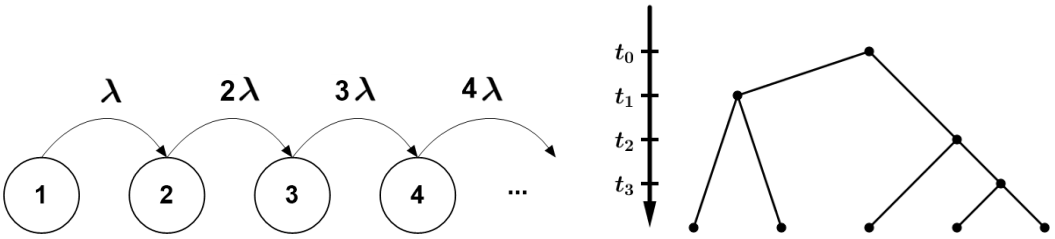


Figure 2: Visualization of the Yule model as a population model with transition rates as well as an example of tree construction under the Yule model.

split it into two new leaves until a determined number of leaves is achieved (see Figure 2 on the right). Note that we will ignore the time until the first species splits and therefore the resulting trees have a root of degree 2 without a so-called root edge.

The leaves represent the current individuals or species and every splitting event is seen as a speciation event in which a species is split into two new daughter species [20, 25, 50].

(The implementation `rtree()` of the `ape` package is used [41, p. 313].)

Although we are only interested in tree topologies we can not fully ignore all aspects of tree models regarding the waiting time until the next event. In order to create alternative models that simulate an influence that causes a differing degree of asymmetry and a variation in the evolutionary rates, we will use the following mathematical property [13, p. 55]:

Let $s_1, ..., s_n$ denote the currently living species with evolutionary rates $\lambda_1, ..., \lambda_n$. The waiting time until the next speciation event is the minimum of the waiting times for a speciation event for every single species ($\exp(\lambda_i)$-distributed for s_i). It is also exponentially distributed with rate $\sum_{1 \leq j \leq n} \lambda_j$. A species i is affected by this speciation event with probability

$$p_i = \frac{\lambda_i}{\sum_{1 \leq j \leq n} \lambda_j}.$$

Thus, the evolutionary rate of a species (or its lineage) directly corresponds to the probability to be affected by a speciation event. This will allow us to modify the evolutionary rates very intuitively as they directly express the ratio of the probabilities. To characterize further tree models we will use the vector $\lambda = (\lambda_1, ..., \lambda_n)$ containing the evolutionary rates and set the default evolutionary rate to 1. Using the Yule model the evolutionary rates would always be described with $\lambda = (1, 1, ..., 1)$.

Example: After the first two splitting steps we have a tree shape with 3 leaves. For the third step let the evolutionary rates be $\lambda = (0.2, 1, 4)$ for these three leaves. This results in the following probabilities of splitting p_1, p_2 and p_3 for each leaf. We have $p_1 = \frac{1}{5} \cdot p_2$ and $4 \cdot p_2 = p_3$; the probabilities are in the proportions $0.2 : 1 : 4$.

$$p_1 = \frac{0.2}{0.2 + 1 + 4} = \frac{0.2}{5.2} \approx 0.038 \qquad p_2 = \frac{1}{5.2} \approx 0.192 \qquad p_3 = \frac{4}{5.2} \approx 0.769$$

2.2 The ζ-models

We can now introduce the *ζ-models*, a collection of various models simulating different ways how the evolutionary rates can be influenced. ζ denotes a positive (> 0) factor by which new leaves or already existing leaves change their evolutionary rates and thereby their probability to speciate. There are several possible ways how ζ can influence tree generation. These are the concrete models that will be used as alternative models in this thesis. A visualization of these different models is shown in Figure 3. Furthermore, we will also have an example that explains the whole process for one model step by step.

direct-children-only (DCO): In every step when a parent leaf was chosen, the direct children get rate ζ, all other leaves are reset to rate 1.

This model resembles a very short timed influence as only the direct children are affected. As later confirmed it can already be expected that the DCO-model requires comparably high ζ values in order to create an effect in tree asymmetry. Especially for $0 < \zeta < 1$ there will be hardly any effect as new leaves are only excluded from the next step, but are allowed to speciate in the following steps (see Section 3.2). Thus, the next models are introduced to simulate a long term effect.

inherited-fertility (IF): The direct children inherit the rate of their parent modified by ζ. All other rates stay untouched. There are different possibilities for these modifications, we will use the following two.

- **both-children (IF-both):** In every step when a parent leaf was chosen, both direct children get rate ζ multiplied with their parent's rate. Thus, a leaf will have rate ζ to the power of its depth.
- **children-different (IF-diff):** Let the parent have rate λ, one direct child gets rate $\frac{2\zeta}{\zeta+1}\lambda$ and $\frac{2}{\zeta+1}\lambda$ with $\zeta \geq 1$. The leaf rates are in the proportion $1{:}\zeta$ with one rate being higher and the other rate being lower than their parents rate. As both new leaves are structurally equal it would be equivalent to take $\zeta \in]0, 1]$ resulting in proportions like $1 : \frac{1}{2}$ or $\zeta \in [1, \infty[$ with proportions like $1 : 2$, thus we decide to use ζ values ≥ 1.

It is obvious that both ζ-IF-models are more prone to produce imbalance than DCO as rates are expected to grow ($\zeta{>}1$) or decrease ($\zeta{<}1$) exponentially over time. The next model describes rates that change over time with every age or splitting step.

age-step-based (ASB): New direct children get rate 1. Every step ($\rightarrow$ age+1) each other rate is updated by multiplication with ζ resulting in a gradual increase ($\zeta{>}1$) or decrease ($\zeta{<}1$) of rates over time.

Inspired by other literature [1, p. 867] we will also use a model called *speciational Brownian evolution*. It differs from the ζ-models as ζ does not illustrate the rate proportions anymore but the variance of normally distributed summands that change the rates. It is also known as a trait-based model because the idea is that a trait changes over time and affects the fitness and therefore the evolutionary rate of a species.

Speciational Brownian evolution (Brownian): Let the parent have rate λ, then new direct children get rates $\lambda_{1,2} = \lambda + c_{1,2}$ with $c_{1,2} \sim N(0, \zeta)$ normally distributed with variance ζ. As a default value the root gets rate 100. All other rates remain untouched. If rates would fall beneath or on zero, they are set to 10^{-10} to ensure that the rates stay positive and that the tree development does not stop due to the case of all rates being zero.

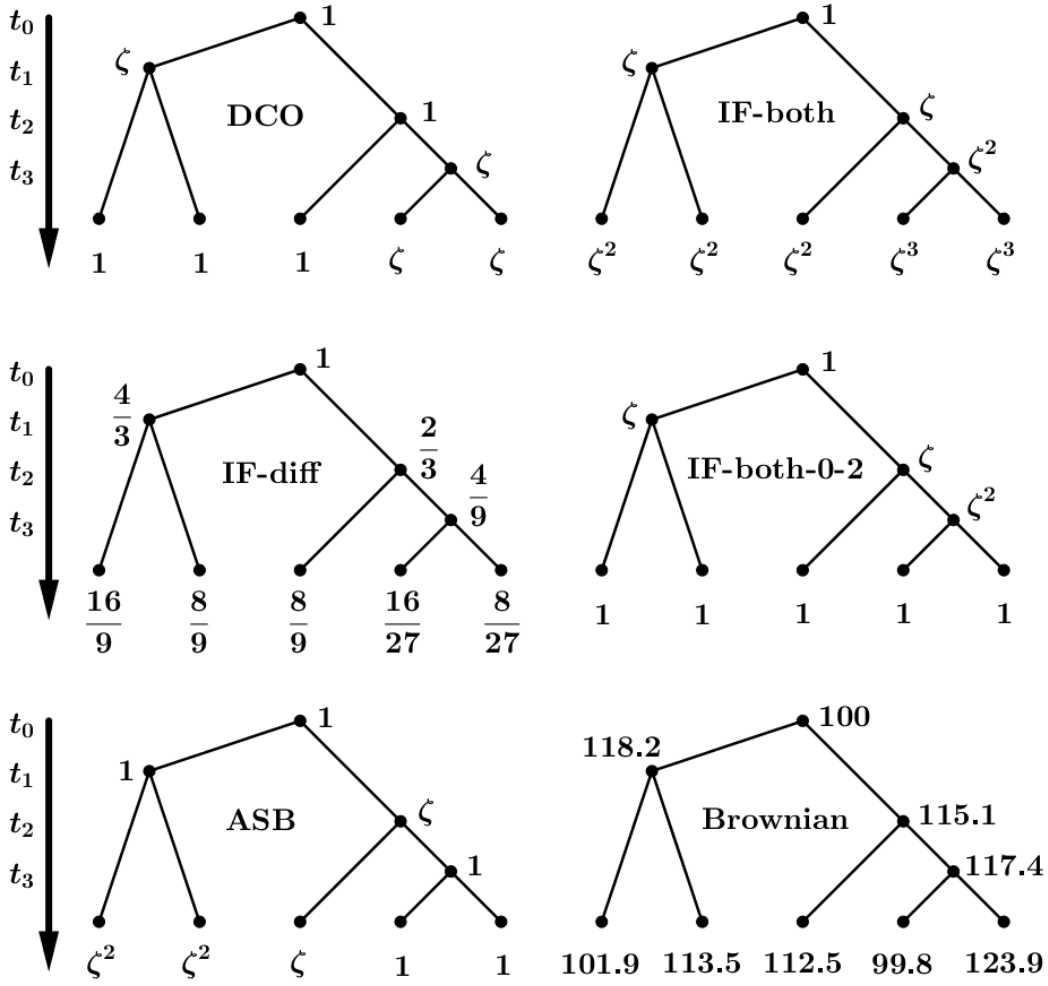


Figure 3: Visualization of the ζ-models described above as well as the Brownian model. For the IF-diff-model ζ has been set to 2 and for the Brownian model to 10 (rates were rounded up to one decimal place). The inner nodes are marked with their rates that were active when they were chosen for the splitting event. The five leaves are marked with their rates after the first four splitting steps. The process could continue from there by choosing a new leaf to split using rate-depending probabilities as explained above.

Another idea would be to limit the alternative model to a subset of time steps. Define (ζ-model)-t_{start}-t_{end} as a model in which only in steps in $[t_{\text{start}}, t_{\text{end}}]$ the tree generation is affected by ζ in one of the above mentioned ways. At the time $t_{\text{end}+1}$ the parent is chosen using those modified rates, but after that all rates are reset to 1 and therefore follow the Yule model from then on. If t_{start} and t_{end} are not mentioned, the ζ-model is assumed to apply to the whole process.

The idea is only mentioned here for the sake of completeness and to present an interesting future research idea, but it is beyond the scope of this thesis. The fundamentals, however, are already implemented in the R-script `GenerateTrees.R` with three variations: all, the first half of or the second half of all time steps under ζ influence.

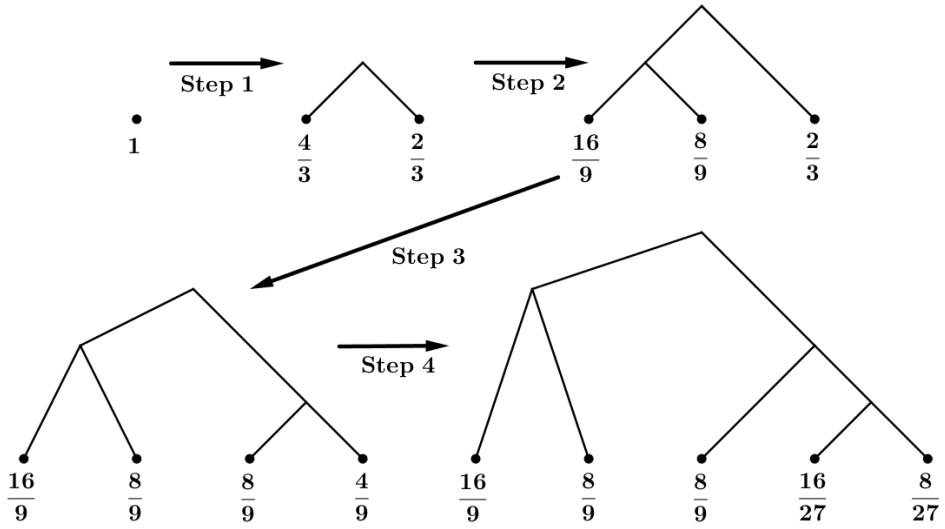


Figure 4: Example of tree generation under the ζ-IF-diff-model. All leaves are marked with their respective evolutionary rates.

Example: To further clarify the process we will use the tree for the IF-diff-model in Figure 3 and explain its generation process step by step (see Figure 4). ζ is set to 2 and let λ^i denote the vector of evolutionary rates that was created in step $i-1$ and is valid at the beginning of step i. Let p^i denote the vector containing the corresponding probabilities of the leaves to be chosen for the speciation event.

Initialization: We start with one leaf with rate 1, the root of the tree.

$\lambda^1 = (1)$, $p^1 = (1)$

Step 1: As there are no other possibilities this single leaf is chosen and split into two new leaves. Following the formula $\frac{2\zeta}{\zeta+1}\lambda_{\text{parent}}$ and $\frac{2}{\zeta+1}\lambda_{\text{parent}}$ with $\lambda_{\text{parent}} = 1$ in this case, we get rate $\frac{4}{3}$ and $\frac{2}{3}$ for them. To keep it consistent the left leaf in the depiction of the tree will always get the higher value, but note that the tree itself is still not ordered (no leaf order). It can be seen that $\frac{4}{3}$ is twice as high as $\frac{2}{3}$ as desired with $\zeta = 2$. There are now two leaves available for selection. The first will be selected with probability $\frac{4}{3} \cdot \frac{1}{2} = \frac{2}{3}$ and the latter with probability $\frac{2}{3} \cdot \frac{1}{2} = \frac{1}{3}$. Again we can see that the first probability is twice as high as the second.

$\lambda^2 = (\frac{4}{3}, \frac{2}{3})$, $p^2 = (\frac{2}{3}, \frac{1}{3})$

Step 2: Let us assume that the first more probable leaf is chosen and is split up into two new leaves. These will have the rates $\frac{16}{9}$ and $\frac{8}{9}$ following the above mentioned formula with

$\lambda_{\text{parent}} = \frac{4}{3}$ in this case. The sum over all rates in λ^3 is $\frac{30}{9}$. Thus, the probabilities p^3 for the next step will be $\frac{16}{9} \cdot \frac{9}{30} = \frac{16}{30}$, $\frac{8}{9} \cdot \frac{9}{30} = \frac{8}{30}$ as well as $\frac{6}{9} \cdot \frac{9}{30} = \frac{6}{30}$.
$\lambda^3 = (\frac{16}{9}, \frac{8}{9}, \frac{2}{3} = \frac{6}{9})$, $p^3 = (\frac{16}{30}, \frac{8}{30}, \frac{6}{30})$

Step 3: Let us assume that the least probable case occurs: The third leaf is chosen for a splitting event. Its children get rates $\frac{8}{9}$ and $\frac{4}{9}$ with $\lambda_{\text{parent}} = \frac{2}{3}$. The sum over all entries in λ^4 is $\frac{36}{9}$.
$\lambda^4 = (\frac{16}{9}, \frac{8}{9}, \frac{8}{9}, \frac{4}{9})$, $p^4 = (\frac{16}{36}, \frac{8}{36}, \frac{8}{36}, \frac{4}{36})$

Step 4: Let us again assume that the least probable case occurs: The fourth leaf is chosen for a splitting event. Its children get rates $\frac{16}{27}$ and $\frac{8}{27}$ with $\lambda_{\text{parent}} = \frac{4}{9}$. Here the sum over all rates of λ^5 is $\frac{120}{27}$.
$\lambda^5 = (\frac{16}{9} = \frac{48}{27}, \frac{8}{9} = \frac{24}{27}, \frac{8}{9} = \frac{24}{27}, \frac{16}{27}, \frac{8}{27})$, $p^4 = (\frac{48}{120}, \frac{24}{120}, \frac{24}{120}, \frac{16}{120}, \frac{8}{120})$

Step 5: This process can be continued as shown in the steps before until the desired number of leaves is reached. If a tree shape with i leaves is required we can stop after step $i - 1$.

The generalized process of the ζ-models also can be retraced using the following pseudo code (see Algorithm 1). For that it should be shortly explained how for a rooted tree the tree format `phylo` in `R` is structured: A tree consists of at least three elements: its number of inner nodes, a vector that holds the leaf labels (its length gives the number of leaves) and an edge matrix, which has a row with parent and child for each of the ($(2n-2)$ if binary) edges of the tree. Thus it is a $(2n-2) \times 2$-matrix for a binary tree.

Algorithm 1: Generating a tree using the ζ-model.

Data: $n \leftarrow$ NUMBER_OF_LEAVES, $\zeta \leftarrow$ DEGREE_OF_ASYMMETRY
Result: tree generated under ζ-model
initialize edge matrix $M \leftarrow$ empty $(2 \cdot n - 2) \times 2$ -matrix ;
initialize vector of current leaves *leaves* $\leftarrow$ *root*;
initialize vector of current evol. rates $\lambda \leftarrow 1$;
for *step* $\leftarrow$ *1* **to** $(n-1)$ **do**
  $i_{\text{parent}} \leftarrow$ `chooseNextParentIndex` (λ) ; `// λ gives ratio of probabilities`
  parent $\leftarrow$ *leaves*$[i_{\text{parent}}]$;
  Fill next 2 rows of M with edges (*parent*,*child*$_1$) and (*parent*,*child*$_2$);
  Remove *parent* from *leaves* and add *child*$_1$, *child*$_2$;
  `updateEvolRates` (λ, i_{parent}, ζ, method);
  `// removes value of parent, adds values for children and modifies values depending on the method`
tree $\leftarrow$ (M,n);

2.3 Balance indices

A balance index for rooted binary trees can be defined as a function $\phi : M_{RB} \to [0, \infty)$ that measures some aspect of symmetry in tree shapes. Commonly the fully balanced tree and the caterpillar are seen as the most extreme tree shapes regarding symmetry and should have extremal index values on the opposite sides of the spectrum. A lot of indices increase with a higher degree of asymmetry, but there are exceptions. For our analysis this does not matter as we use acceptance intervals that cover both low and high values as well as limits that will be dynamically set by the evolutionary algorithm.

Overall, there are different characteristics in which balance indices differ that can sometimes be important for their application [22, p. 1819]. One of these is the question if the index can deal with polytomies or only with completely resolved binary trees. Unresolved nodes are normally considered missing information rather than true multifurcative evolutionary development. Due to that using a balance index for only binary trees will not always be disadvantageous as non-binary trees will often be generally sorted out in advance. Furthermore, even indices that can handle these trees normally skip unresolved nodes in their calculation. Nevertheless, there are topics where it is rare to find resolved trees and one has to fall back on balance indices that do not require a binary tree [5].

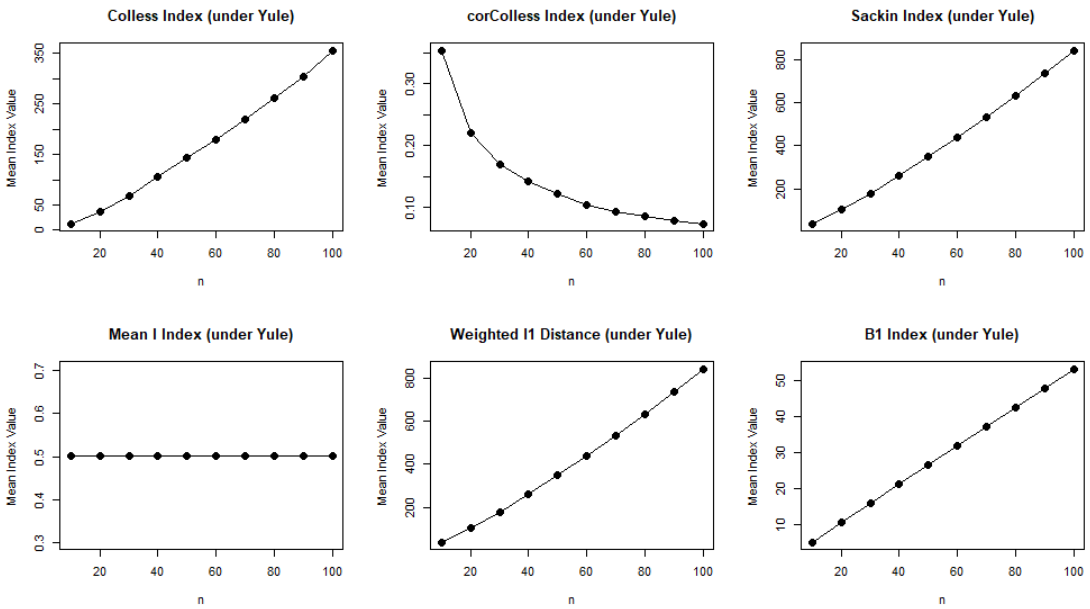


Figure 5: Some indices and their mean index values for 1000 trees per n generated under the Yule model with n leaves. The corrected Colless index is an example for a normalized index, that is still dependent on n, and the $meanI'$ index is an example of an n-independent index. One can see that the range of values is very different for each index.

Furthermore, it should be noted that the comparison of trees with different numbers of leaves can be problematic. For many balance indices, even if they are normalized to $[0, 1]$, an increase in the number of leaves will lead to increasing or decreasing average index values (see Figure 5). These should be therefore used with caution [22, p. 1819], however there are indices using

a value I' (see below) that have a constant expected balance index value over all tree sizes and could thus be viable for such a comparison [5].

The following list shows a (after thorough research hopefully) complete list of currently existing balance indices for binary rooted trees. Due to the fact that a list of this extent cannot be found in literature and some indices and their implementation will be commented regarding their features or implementation this list will not be attached to the appendix but featured here. Furthermore, it gives insights in the various approaches on how to measure balance or imbalance in trees. All indices are described with their function names used in the `R` scripts. An example can be seen in Figure 6 for the total cophenetic index.

Average leaf depth $\bar{N}$ is defined as the average number of interior nodes between a leaf and the root (including the root). This is equivalent to the average depth of the leaves [45].

$$AvgLeafDepI(T) = \bar{N}(T) \coloneqq \frac{1}{n} \sum_{x \in V^1} \delta(x)$$

Variance of leaf depth σ_N^2 is defined as the variance of the number of interior nodes between a leaf and the root (including the root). Again this is equivalent to the variance of the depth of the leaves [45].

$$VarLeafDepI(T) = \sigma_N^2(T) \coloneqq \frac{1}{n} \sum_{x \in V^1} \left(\delta(x) - \bar{N}(T)\right)^2$$

Sackin index The idea of the Sackin index is to look at the number of descendant leaves $\kappa(y)$ of an inner vertex y or in an equivalent definition at the depth $\delta(x)$ of leaf x. It therefore is closely related to the average leaf depth [45].

$$SackinI(T) \coloneqq \sum_{y \in \mathring{V}(T)} \kappa(y) = \sum_{x \in V^1(T)} \delta(x)$$

Colless index The Colless index describes a very intuitive approach as it compares for every inner vertex v how many leaves are descendants of either v_1 or v_2, the direct children of v [10].

$$CollessI(T) \coloneqq \sum_{v \in \mathring{V}(T)} |\kappa(v_1) - \kappa(v_2)|$$

There also is a corrected version of the Colless index with values in the interval $[0, 1]$ [22, p. 1819].

$$corCollessI(T) \coloneqq \frac{2 \cdot CollessI(T)}{(n-1)(n-2)}$$

Total cophenetic index Given a tree T with n leaves this index calculates the sum of the depths of the lowest common ancestor $lca_T(v, w)$ over all pairs of different leaves v and w [35].

$$TotalCophI(T) \coloneqq \sum_{1 \leq v < w \leq n} \delta(lca_T(v, w))$$

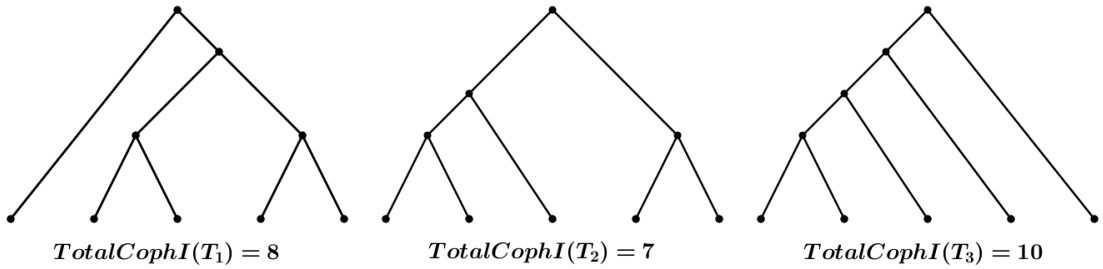


Figure 6: Total cophenetic index of three trees with $n = 5$ leaves. The middle tree is the most balanced of them according to this index.

Quartet index This index is also applicable to non-binary trees. It calculates the sum over the value $f(v_1, v_2, v_3, v_4)$ for all quartets (4-tupels) (v_1, v_2, v_3, v_4) of leaves of a tree T with n leaves. The value $f(v_1, v_2, v_3, v_4)$ quantifies the level of symmetry of the subtree obtained by restricting T to the leaves of the quartet. f is required to increase with the number of automorphisms on this subtree as they indicate symmetry. The subtrees can have two (in the case of a binary tree) or five (non-binary tree) different tree shapes [11, p. 15].

$$QuartetI(T) \coloneqq \sum_{1 \leq v_1 < v_2 < v_3 < v_4 \leq n} f(v_1, v_2, v_3, v_4)$$

In case of a binary tree they assign value 0 to the caterpillar tree and q_3 to the fully balanced tree (other q_i-values for non-binary shapes). The authors suggest two possible assignments: $q_i = i$ or $q_i = 2^i$, but as we are only differentiating between caterpillar and fully balanced, we can set $q_3 = 1$ which results in counting the number of quartet induced subtrees that are fully balanced.

The implementation uses a comparison of the last common ancestor of each pair (i, j), (i, h) and (i, k) in a quartet (i, j, h, k) with i being randomly chosen but fixed. In a fully balanced tree shape, the oldest common ancestor will appear exactly twice, but in a caterpillar either once or three times depending on the position of i.

R index (left-light rooted ranking) This measurement was originally not intended to be a balance index. To every distinct tree shape with n leaves it assigns a value ranging from 1 to $|RB_n|$. More symmetrical trees get higher values [18, p. 213]. Let T_R and T_L be the two pending subtrees rooted in the children of the root with size r and s, respectively. Choose T_R and T_L such that $r > s$ or in the case of equality that $R(T_R) \geq R(T_L)$. $Z(i)$ denotes the number of unlabeled rooted trees with i leaves. $Z(1) = Z(2) = 1$ and $Z(x) = 0$ for non-integer values x.

$$LLRRI(T) \coloneqq R(T) = \begin{cases} \sum_{i=1}^{s-1} Z(i)Z(n-i) \quad + Z(r)\left(R(T_L) - 1\right) + R(T_R), & \text{if } r > s \\ \sum_{i=1}^{s-1} Z(i)Z(n-i) \quad - Z(s) & \\ \qquad + R(T_L)\left(Z(s) - \frac{1}{2}R(T_L) + \frac{1}{2}\right) + R(T_R), & \text{if } r = s \end{cases}$$

$$Z(i) \coloneqq \sum_{j=1}^{\lceil i/2-1 \rceil} Z(j)Z(i-j) \quad + \frac{1}{2}Z(i/2)(Z(i/2)+1)$$

B_1 **index** This index is based on the interior nodes except the root (here node number $n-1$). For every other interior node i we look a the corresponding subtree rooted in i and calculate the maximal number of interior nodes M_i between i and the leaves of the subtree (including i). Again we can use the depth of the leaves in order to define M_i [47].

$$B_1I(T) \coloneqq \sum_{i=1}^{n-2} \frac{1}{M_i} = \sum_{i=1}^{n-2} \frac{1}{\max_{x \in V^1(T_i)} \delta(x)}$$

B_2 **index** For B_2 we measure again the number of interior nodes on a path from every leaf to the root including the root (equivalent to the depth of the leaf) and use the following formula [47].

$$B_2I(T) \coloneqq \sum_{i=1}^{n} \frac{\delta(i)}{2^{\delta(i)}}$$

I' **index** The idea of using I' has been developed by Fusco and Cronk (modified by Purvis et al.) [19] and has later been transformed into new indices [1]. I'_v is a value that can be calculated for a fully resolved (i.e. it has two outgoing edges) vertex v with at least 4 descendant leaves. For this index we calculate the value only for the root of the tree $I' = I'_{root}$.

$$I'_v = \begin{cases} I_v & \textit{if } n \textit{ is even} \\ \frac{n-1}{n} \cdot I_v & \textit{else} \end{cases} \qquad \textit{with} \quad I_v = \frac{B-m}{M-m}$$

With n being the size of the subtree rooted in v, B being the the size of its larger daughter clade (pending subtree rooted in a child of v), $M = n-1$ being the maximum value for B and $m = \lceil \frac{n}{2} \rceil$ being the minimum value for B. It measures how equally the leaves are split, with 0 indicating the most equal split possible and 1 indicating the most uneven split.

$\sum I'$ **index** For every interior node $v \in \mathring{V}$ that is fully resolved and that has more than 3 descendant leaves we calculate I'_v. Then we take the sum of these values $\sum I'_v$. This index is not suitable to compare trees of the same size that are resolved to varying degrees.

mean I' **index** For every interior node $v \in \mathring{V}$ that is fully resolved and that has more than 3 descendant leaves we calculate I'_v. Then we take the mean of these values *mean* I'_v

mean I'_{10} **index** Only for the 10 oldest nodes I'_v is calculated. Then we take the mean of these values.

Cherry index This index uses the number of leaves as an aspect to measure symmetry. There are two different versions, one that calculates the number of cherries $c(T)$ [31] and the modified version that counts all leaves that are not in a cherry.

$$CherryI(T) \coloneqq c(T) \qquad modifCherryI(T) \coloneqq n - 2 \cdot c(T)$$

Symmetry nodes index This index uses the number of symmetry nodes in the tree. Again we can state two versions with the second being the number of interior nodes, that are not symmetry nodes.

$$SymNodesI(T) \coloneqq s(T) \qquad modifSymNodesI(T) \coloneqq (n-1) - s(T)$$

Weighted l_1 distance This index is based on the empirical vs. the theoretical distribution of subtrees of certain sizes under the Yule model. $f_n(z)$ denotes the observed and $p_n(z)$ the expected frequency of subtrees with size z in the tree (percentage of inner nodes with z descendant leaves). For $n \geq 2$ we have $p_n(z) = \frac{n}{n-1}\frac{2}{z(z+1)}$ if $z = 1, 2, ..., n-1$ and $p_n(n) = \frac{1}{n-1}$ if $z = n$ [6, p. 145].

$$D(T) \coloneqq \sum_{z=2}^{n} z|f_n(z) - p_n(z)|$$

The frequencies in itself can also be considered balance indices with the cherry index as the best known example. They will later be used as building blocks in one version of the genetic programming runs (see Section 7).

Metric based (Colijn) Recently there has been a metric introduced for tree shapes instead of phylogenies. It follows a similar idea like Furnas' left-light rooted ranking and assigns every topology a different integer value. Whereas Furnas ranks trees in RB_n separately for each $n \in \mathbb{N}$, Colijn's apporach enumerates **all** possible tree shapes with different numbers [9]. Let single leaves have enumeration 1 and let the pending subtrees of the root have enumeration k and j with $k \geq j$. Then the complete tree gets enumeration

$$\phi(k, j) = \frac{1}{2}k(k-1) + j + 1.$$

Thus, the enumeration can be recursively calculated. A numeric metric like the euclidean distance can then be used on these enumerations or also on the vector of the enumerations of all subtrees. This concept could potentially be used as a balance index if for example the distance to the corresponding caterpillar with the same number of leaves was calculated.

After doing a run time comparison with the `R` package `microbenchmark` [32] for different numbers of leaves it was decided to exclude the quartet index (run time in $\Omega(n^4)$) and the left-light rooted ranking (run time in $O(n^2)$ without precomputations) from the comparison and especially from the genetic programming runs. Even for very small numbers of leaves (less than 30, the smallest n that will be used in these analyses), their run time was more than 1000 times as high as the run time of the other indices (see Table 1). As there have to be several thousands of trees measured for our comparisons and the indices' run time will further drastically increase with higher n they had to be excluded.

Another index had to be omitted as well due to a different reason. Although being fast to calculate the Colijn-metric-based index has the disadvantage that even for small tree sizes e.g. the caterpillar for 25 leaves the index values go beyond the standard range of numerical values in R. Therefore it would not have been possible to use the distance from the caterpillar as any

meaningful measurement. In future balance index comparisons it could be possible to include this index if a new range for numerical values was introduced.

Table 1: Run time comparison of the balance indices for varying numbers of leaves n. The approximate run time was averaged over 10 to 100 trees with `microbenchmark` using `<balance_index>(genPhylo(n))`. All times are given in milliseconds.

n	20	25	30	100	200
other indices	-	-	6 ms	30-50 ms	70-150 ms
mean I'_{10}	-	-	12 ms	50 ms	125 ms
sum I', *mean* I'	-	-	12 ms	100 ms	300 ms
TotCophI	-	-	77 ms	1660 ms	9,500 ms
QuartetI	4,000 ms	11,000 ms	39,000 ms	-	-
LLRRI	865 ms	28,000 ms	-	-	-

3 Comparison of established balance indices

3.1 Method of comparing the power of balance indices

Remark. *In published literature balance indices have been used in two different ways for phylogenetic applications:*

The index values of different groups of trees can be used as data and after overcoming some problems with test requirements regarding distribution, an analysis of balance differences between these groups is possible for example with an ANOVA [22, p. 1821].

The second usage common for the comparison of balance indices is to view a balance index as a test statistic and therefore the basis for a statistical hypothesis test. This will be more thoroughly explained in the next paragraphs, but it should be noted that in published articles the term "confidence interval" is often vaguely used to describe the "region of acceptance" [26, p. 1174]. The confidence interval gives information on how precisely the location of a parameter was estimated, but the term is sometimes used instead of "region of acceptance" referring to the interval in which the null hypothesis is not rejected (the complement of the critical region).

In this thesis we will use balance indices as non-parametric statistical tests. A hypothesis test is a method to make the decision if observed data is consistent with a null hypothesis H_0 or if it differs too greatly such that the null hypothesis should be rejected because it is very improbable that the results could have arisen by chance. In a non-parametric test these hypotheses cannot be described using parameters. In general we calculate the value of a test statistic with a known distribution for our observations and decide to reject H_0 if the calculated value lies within a critical region.

Such a test can make two kinds of errors: The *first type of error* is the false positive rate. It occurs when the null hypothesis is wrongly rejected. The false negative rate, the *second type of error*, occurs when the null hypothesis is wrongly not rejected. Normally there is a trade-off between these types of errors, such that a decrease in one will increase the other. Therefore, the developers of a test have to find a region of acceptance for the test values that satisfy the requirements which are set by their field of application.

In general the following steps are used for the application of a statistical test [7, p. 97-108] [29, p. 147-152].

1.) Define the null hypothesis H_0 as an initial research position and the alternative hypotheses H_1.

2.) Choose the appropriate *test statistic* $\mathcal{T}$.

3.) Derive the distribution of $\mathcal{T}$ under the null hypothesis.

4.) Select a level of significance, a probability threshold below which the null hypothesis will be rejected. This gives an upper limit for the false positive rate.

5.) Choose a partition of the possible values of $\mathcal{T}$ into a *region of acceptance* (H_0 not rejected) and a *critical region* (H_0 is rejected), such that the probability for a value in the critical region under the null hypothesis is smaller or equal to the level of significance.

6.) Calculate the value of $\mathcal{T}$ based on the sample data.

7.) Reject H_0 if the value lies in the critical region, otherwise we fail to reject H_0.

The *power* is the true positive rate, e.g. how well the test can recognize the deviation from the null hypothesis. With a limit for the first type error through the level of significance we will use the power as a measurement for the quality of a balance index.

Now, we want to have a look on the actual execution of these steps. Alongside this section provides an overview and a summary of the methods and results of four comparisons of balance indices that can be found in literature. Some ideas are commented to discuss their suitability to be incorporated in the analyses in this master thesis. The already published results regarding balance indices are summarized in Table 3. For sample sizes and further values of the previous comparisons see Table 2.

3.1.1 Null hypothesis (Yule model)

With regard to tree imbalance the Yule model is normally used as the null hypothesis and will also be used here.

Other models have been discussed as null hypotheses like two variations of the *uniform model*, the proportional-to-distinguishable-arrangements-model (all phylogenies are equally probable) and the equiprobable-types-model (all topologies are equally probable). Because both do not model evolution sensibly and do not arise from a plausible population development, they are mostly rejected and ignored [38, p. 36].

There have been efforts to find a family of tree models (depending on one or two parameters) that include the Yule and the uniform model for a certain parameter setting. For example the beta-splitting model [2], the alpha-model [17] as well as the alpha-gamma-model [8]. For the first two models the average parameters for reconstructed trees have been estimated and were shown to lie between the parameters for the Yule and uniform model [4].

These families of tree models will also not be used for a parametric test in this master thesis, because their processes are not directly linked to evolutionary development and the parameters do not give an intuitive understanding of the nature of the model. The alpha model for example uses a probability that a new leaf can arise from an interior edge, which has no real meaning in evolutionary development as only current species are able split up.

Another null hypothesis used once was the "biogeographic null" model. With this null model one can test if a local subphylogeny is significantly more symmetric or asymmetric than expected if a random phylogeny of the same size is picked from the overall supertree. This model yields the advantage to be applicable to non-monophyletic trees (in contrast to the Yule model), as we can choose a monophyletic[1] subtree to be analyzed. Furthermore, geographical or climate information can be taken into account [23, p. 108]. However, this model was not used for the comparison of balance indices with simulated trees, but for the evaluation of real data. In this model balance indices do not serve as statistical tests, but only as data producing measurements that can be evaluated with further tests. Additionally this model does not apply to single trees that are to be analyzed here. Thus, this model will not be used.

[1] A monophyletic subtree contains all taxa/nodes that share a common ancestry.

3.1.2 Test statistics and their distributions (BI and distribution by sampling)

The test statistics that will be used and evaluated are of course the balance indices. Calculating variances and distributions of balance indices analytically is too complex (or impossible for some indices), therefore it is the common decision to obtain the distribution via sampling of trees (all 5 papers mentioned in the table below). The comparisons will be done for $n = 30,\ 100$ and 200 in order to be able to compare results with the latest comparison [6] and because it is to be expected that reconstructed phylogenies with larger numbers of leaves will be more common as technologies and methods advance.

We will simulate $N_d = 10^4$ for $n = 30, 100$ and due to run time issues $N_d = 2 \cdot 10^3$ for $n = 200$ trees under the Yule model and calculate their index values to estimate the distribution.

3.1.3 Level of significance and critical region (quantile-based)

Both a two-tailed (H_1=deviation from Yule model, more balanced or asymmetric as expected by chance) as well as a one-tailed test (H_1=more imbalanced) have been used in previous comparisons. The former for a general analysis, the latter because observed trees are often more imbalanced. As we want to explore the balance indices' power to recognize both more balanced and more imbalanced trees, we will stick with a two-tailed test.

To stay comparable with other analyses we will set the level of significance to 5%. Thus, $\leq 5\%$ of trees constructed under the Yule model will be incorrectly accessed (first type error).

As in literature where they used the corresponding quantiles as critical values for a one-tailed test, we will use the 0.025- as well as the 0.975-quantiles to create an acceptance interval I. This method is shown in Figure 7 as well as 8 on the left and will now be clearly defined.

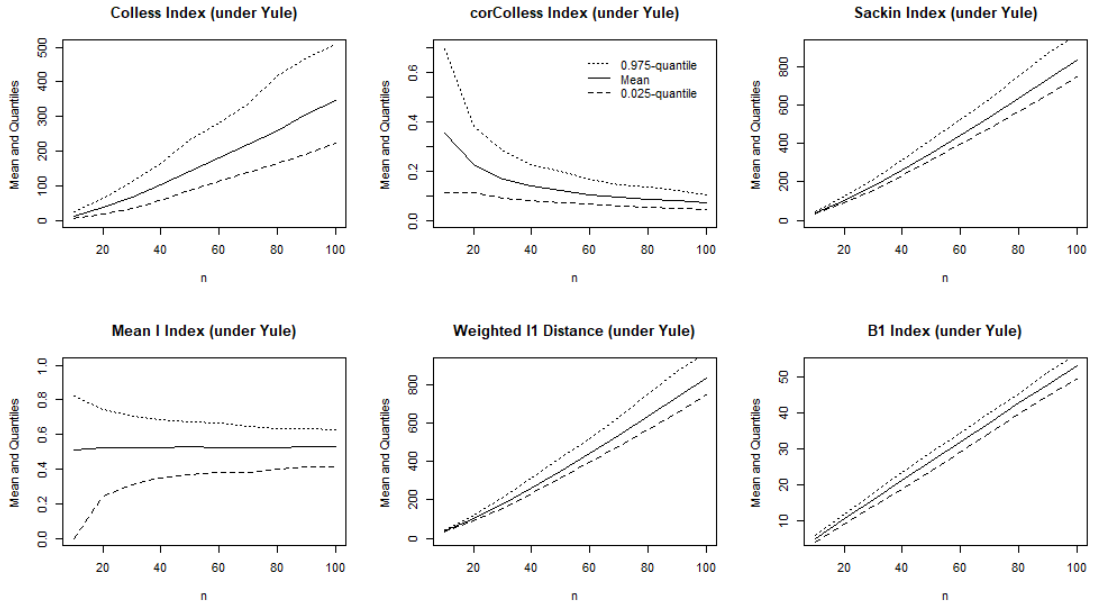


Figure 7: Visualization of mean and quantiles for some balance indices estimated based on 1000 Yule trees per number of leaves n.

An empirical p-quantile with $p \in [0,1]$ for a sample $X = \{x_1, ..., x_n\}$ sorted in ascending order can be defined as [29, p. 70]

$$x_p^* = \begin{cases} \frac{1}{2}(x_{np} + x_{np+1}), & \text{if } np \in \mathbb{N} \\ x_{\lfloor np+1 \rfloor}, & \text{else.} \end{cases}$$

Thus, at least $p \cdot |X|$ values are smaller or equal to x_p^*. x_p^* divides the values approximately in a ratio $p : (1-p)$ if the distribution is sufficiently wide spread. We require I to contain at least 95% of the values because the false positive rate would otherwise exceed 5%. Therefore, we have to exclude the quantiles resulting in $]-\infty, x_{0.025}^*[\cup]x_{0.975}^*, \infty[$ as the critical region for the 0.025-0.975-quantile approach. For examples see Figure 7.

We will not use a different critical region because with this we can do replicable two-sided tests and any optimization of the critical region can only be based on a certain alternative model or real data for every context that we want to explore. Furthermore, this should also give information on the quality of the balance indices in a one-tailed test.

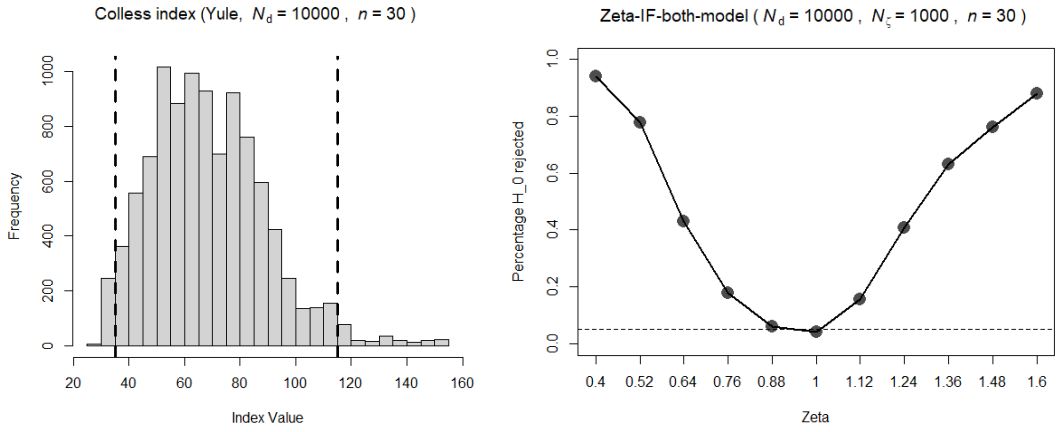


Figure 8: Visualization of the distribution and acceptance interval (0.025- and 0.975-quantile marked with bold vertical lines) of the Colless index as well as the power for several values of ζ under the ζ-IF-both-model (30 leaves, 10000 trees for the estimation of the distribution, 1000 per ζ for the calculation of the power).

3.1.4 Alternative hypotheses (ζ-models)

Different examples for the alternative hypotheses H_1 have been given. All ideas were integrated in some form into the ζ-models that have been defined in Section 2.2.

- A model named biased speciation was directly adopted under the name IF-diff [26, p. 1177].
- A second version of the biased speciation model has been used as well. Let $p \in [0,1]$ be a fixed value. Let a species with rate r split, then its direct children will have the rates $p \cdot r$ and $(1-p) \cdot r$. The other rates will not be reset. $p = 0$ or $= 1$ will lead to asymmetric and $p = 0.5$ to more balanced trees [6, p. 149]. This model will not be used, as we have already

integrated a version of inherited fertility with unequal influence of the daughter species and it is not suitable to generate more asymmetric trees than under the Yule model, because both children have rates lower or equal than their parent.

- Second, speciational Brownian evolution with speciation rates depending on the value of evolved traits. One trait was expressed with a parameter X. For the root X was set to 100 and X was changed only at speciation events by drawing from a normal distribution for both new lineages [1, p. 867] [37, p. 6]. This model will be used in this master thesis as it provides a completely different approach with "evolutionary noise".

- Furthermore, a model was used with speciation rate λ of a lineage decreasing over time since its last speciation event (age). As a formula $\lambda = A \cdot t^{-0.5} + 0.5 = \frac{A}{\sqrt{t}} + 0.5$ with t being the age of a lineage and A being a parameter for the level of influence. λ was updated frequently as a continuous process was approximated instead of a discrete process [1, p. 867] [37, p. 7].

  In order to save computing time we will not directly adapt this model. However, we will try to mimic the model using the ζ-ASB-model which is significantly faster to calculate, but lacks the frequent update of the rates.

After the overall comparison we will select interesting models as a basis for the genetic programming runs.

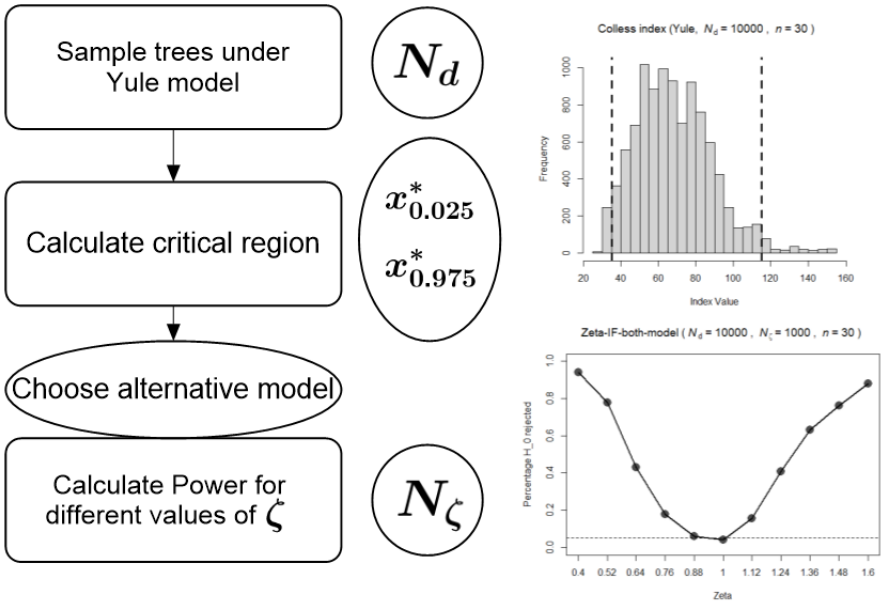


Figure 9: Summary of the procedure to evaluate a balance index.

3.1.5 Calculating power and comparison

The power of a balance index can only be estimated for a given non-Yule model. For that we will simulate $N_\zeta = 10^3$ (sample size per ζ) trees under the non-Yule model and calculate the percentage of trees that have index values outside of the acceptance interval and are therefore

considered not to be generated under the Yule model. The value of ζ influences the power strongly. For $\zeta \to 0$ and $\zeta \to \infty$ the power increases under the DCO-, IF-both- and ASB-model, resulting in a bathtub like curve (see Figure 8 on the right and 9). For IF-diff- and the speciational Brownian model we will only observe increasing power with increasing ζ values.

Now, the approach has been clearly defined. The next section briefly summarizes the already published results and presents the two Tables 2 and 3 that contain the results concerning each balance index as well as the methods of the previous comparisons. Then, the actual comparison will follow.

3.1.6 General results of previous studies

- In general we can not assume that balance indices are normally distributed (shown for $i) - vi)$ see Table 3) [26, p. 1176].
- Balance indices $i) - vi)$ see Table 3) are not able to work as statistical tests for $n < 8$ because their critical region is trivial in these cases [26, p. 1174].
- General observation: The ratio of speciation rates/leaf probabilities has to be large to be noticed by balance indices. Therefore one can assume that an imbalance effect that is noticed by these tests is likely to be large.
- The balance indices perform differently depending on the chosen alternative model. Thus, the goal to construct an index that performs well on a mixture of various alternative models should be taken into consideration as well. In reality there can be situations in which an index that has an overall good power is useful because there is no assumption possible based on other prior information. However, it is not clear how this mixture of alternative models should be defined.

Table 2: Null hypothesis, examples from H_1 used for calculating power, method to construct the critical region, values for number of leaves n and and sample size N_d and N_ζ in published comparisons.[3]

Source	[22], 1992	[26], 1993	[1], 2002	[6], 2005	[23], 2007
H_0	Yule	Yule	Yule	Yule	Yule & "bio-geographic null"
H_1-ex	-	biased speciation	trait value-based & age-based rates	biased speciation v2	-
test	-	two-tailed	one-tailed (H_1= imbalanced)	one- & two-tailed	two-tailed
crit. values	-	(not mentioned)	empirical 0.95-quantile	empirical 0.95-quantile	(not clear)
n	1,...,14 [4]	$\in [8, 50]$ & 10,20,40	8,16,32,64	30,100,200	-
N_d	10^4	10^5	$(5\cdot)10^3$	10^4	10^4
N_ζ	-	10^4	10^3	10^4	-

[2] For the sake of completeness it should mentioned that there has been an additional recent study that compared balance indices as well. However, they used a different method to assess the power, such that the results can not be compared easily [21, p. 56].

[4] The authors tested the balance indices with reconstructed phylogenies based on real data. The number of leaves n did not exceed 14 because the amount of such reconstructed phylogenies with that many leaves was too small at that time to produce more insights.

Table 3: Balance indices (BI) used in published analyses with short description of results. "n↑" means "for higher numbers of leaves", a-b and t-b refer to the age- and trait-based alternative models.

Source	[22], 1992	[26], 1993	[1], 2002	[6], 2005	[23], 2007
i) $corCollessI$	✓	good	best (trait-based)	-	✓
ii) $\bar{N}$	-	weak	best (t-b)	-	-
iii) σ^2_N	-	good	ok	-	-
iv) R	-	good	-	-	-
v) B_1	-	different, best	good (n↑, age-based), weak (t-b)	-	-
vi) B_2	-	weak	weak	-	-
vii) $\sum I'$	-	-	good (n↑, a-b), weak (t-b)	-	-
vii) $meanI'$	-	-	good (n↑, a-b), weak (t-b)	-	-
ix) I'_{10}	-	-	(weak)	-	-
x) $SackinI$	-	-	-	good	-
xi) D	-	-	-	good	-
xii) $CherryI$	-	-	-	very weak	-

3.2 Comparison

As discussed in Section 3.1 we will use the following parameters with $N_d = 20,000$ for $n = 30$, $N_d = 10,000$ for $n = 100$ as well as $N_d = 2,000$ for $n = 200$:

$$n = 30, 100, 200 \qquad N_d = 2 \cdot 10^4,\ 10^4,\ 2 \cdot 10^3$$
$$A_{crit} =]-\infty, x^*_{0.025}[\cup]x^*_{0.975}, \infty[\qquad N_\zeta = 10^3$$

At first it had to be explored which values for ζ are suitable for each model. The goal was to have the curves reach $\approx$80% on the sides for the best indices and lower values for others. However, especially for the DCO-model, the values < 1 could not always be chosen to fulfill this criterion. The range of these ζ-values should be divided by 5-7 equidistant ζ values in total, such that the development of balance index values depending on ζ can be retraced. Important to add is that all balance indices are accessed on the same sample of trees. Thus, the confounding influence of different sample bases can be decreased.

3.2.1 Implementation (R)

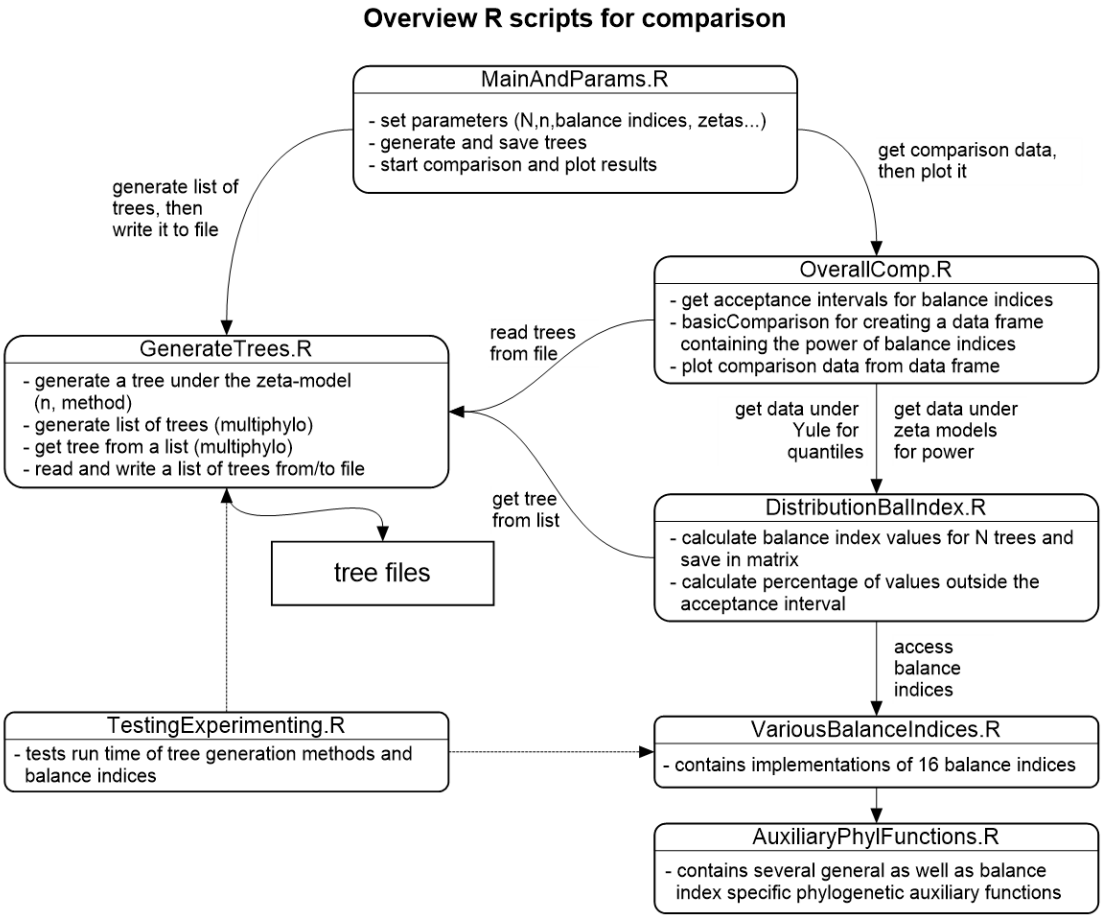


Figure 10: Overview of the general process of the balance index comparison.

See Figure 10 for an overview of the R-scripts used to generate the comparison data (see `scripts_comp.zip` on the attached CD). All balance index functions are written to apply to trees of class *phylo*, the basic tree format of the `ape` package. As said before in Section 2.2 a tree consists of at least three elements: its number of inner nodes, a vector that holds the leaf labels (its length gives the number of leaves) and an edge matrix, which has a row with parent and child for each of the ($(2n-2)$ if binary) edges of the tree.

Unfortunately, the methods to generate a tree under a ζ-model produce a different node enumeration than trees generated with `ape` methods. This had dramatic effects as trees would change and sometimes not even be binary after saving it in a text file and reloading it using the standard `ape` functions *write.tree* and *read.tree*. An auxiliary function which assigned the standard `ape` node enumeration was tested but discarded because it increased the run time by the factor ≥ 80. To avoid this considerable elongation of the run time a new set of writing and reading functions was implemented in the scipt `GenerateTrees.R`. For we are only dealing with binary trees the number of leaves or inner nodes can be deducted from the number of rows of the edge matrix. Thus, these functions only save the edge matrix. For multiple trees the writing function combines the edge matrices. The reading function can than retrieve the single matrices (two columns of the grand matrix each).

3.2.2 Results of the first comparison

In the following Figures 11, 12, 13, 14 and 15 the results of the first comparison are visualized. The figures for $n = 100$ have been excluded because they do not provide further insights. However, the complete set of figures as well as RData-files of the data sets can be found on the attached CD (`firstcomp_results.zip`).

As expected the average leaf depth and the Sackin index as well as the corrected version of the Colless index and the Colless index itself delivered exactly the same results because they are only scaled by a factor $\frac{1}{n}$ or $\frac{2}{(n-1)(n-2)}$,respectively. The scaling only compresses the range of index values, but does not effect their function as a statistical test. Therefore, one index of each pair – the average leaf depth and the corrected Colless in this case – was omitted to keep the figures more clear. The I'_{root} index was excluded as well because it was extremely weak (power $<5\%$ regardless of ζ for n up to 60 leaves) and there still were three other indices included that use the I' approach.

In accordance with our speculations the DCO-model required high ζ values to produce asymmetric trees and even for considerably low ζ values the model has only little influence on tree symmetry (see Figure 11).

How similar are the indices? Are there groups of indices that perform similarly well? For the DCO-, IF-both-, IF-diff- and Brownian model there are mostly clear groups of balance indices regarding their performance: The Sackin index, the Colless index and the variance of leaf depths always have the highest power. Slightly below we have the total cophenetic index. The cherry and the symmetry nodes index performed worst, the remaining indices were in the middle range. For the ASB-model the ranking differed. The B_1 index, the $\sum I'$ as well as the $meanI'$ index performed best (consistent for higher and lower ζ values). The cherry and the symmetry nodes index performed significantly better for higher numbers of leaves (near top tier for 200 leaves). The best group mentioned above for the other models (Sackin index, Colless

index and the variance of leaf depth) are at best middle tier and falling off for higher numbers of leaves.

This drastic shift of groups from best to worst for different models makes it clear that the right choice of a balance index can have a serious impact. With prior knowledge on fertility inheritance, trait-based fertility or age-based fertility different groups of balance indices should be recommended. This leads to the question if we can find a balance index that performs well on both types of models (see Section 5.1).

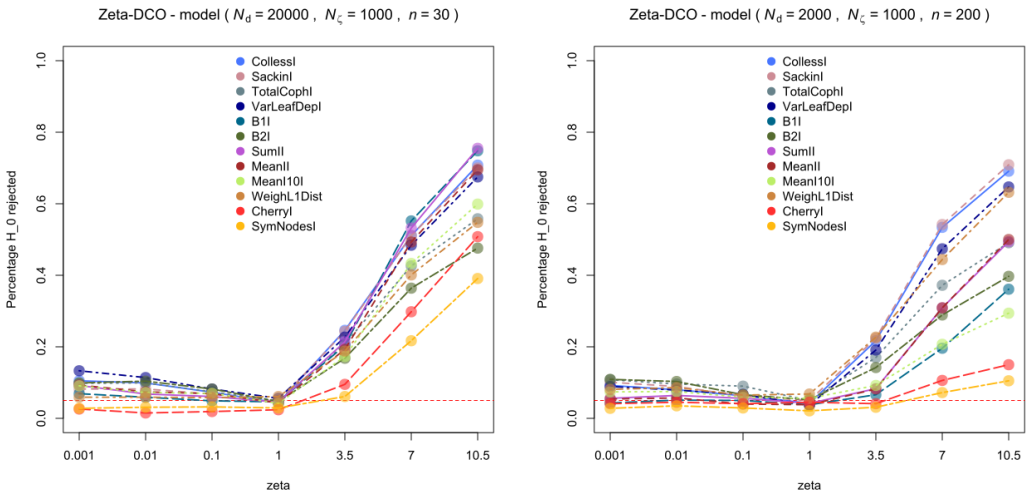


Figure 11: Comparison of the power of balance indices regarding the ζ-DCO-model.

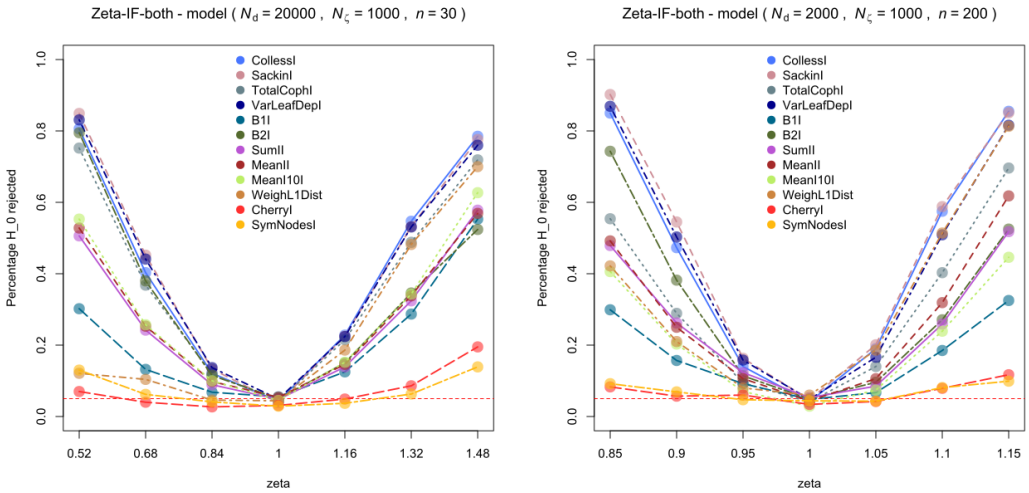


Figure 12: Comparison of the power of balance indices regarding the ζ-IF-both-model.

Can the list of balance indices be shortened? According to these comparisons it is never beneficial to choose the cherry or the symmetry nodes index over the B_1 index, the $\sum I'$ or the $meanI'$ index because the latter have higher power for every ζ value of every tested alternative

model. However for the ASB-model both indices were more powerful than a large part of the remaining established indices.

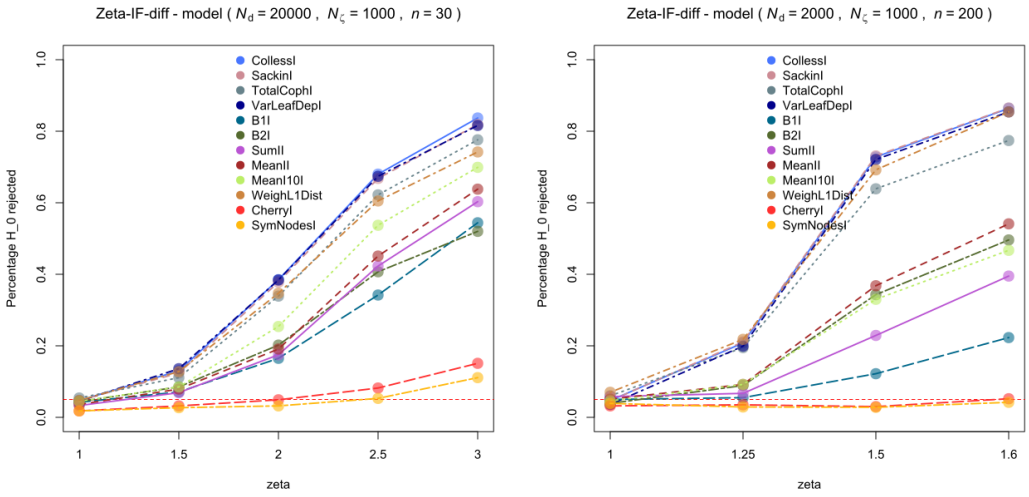


Figure 13: Comparison of the power of balance indices regarding the ζ-IF-diff-model.

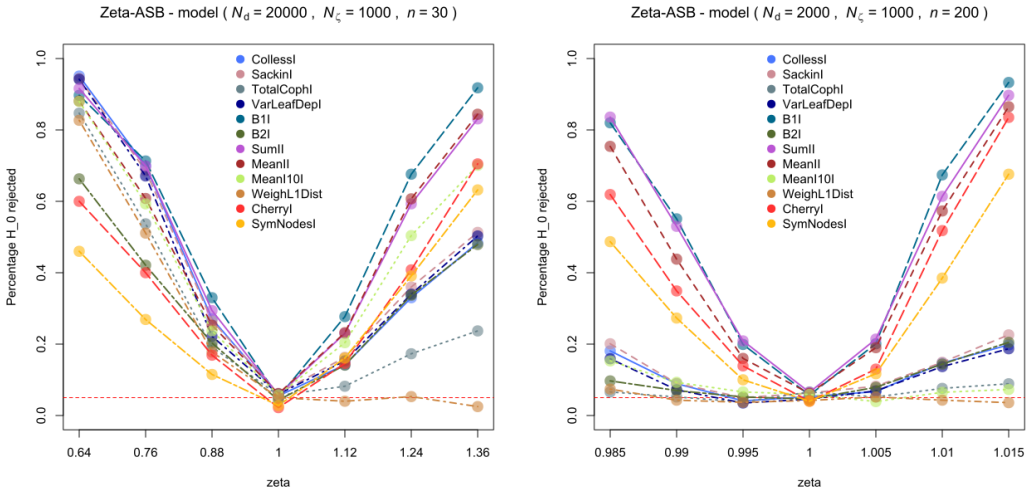


Figure 14: Comparison of the power of balance indices regarding the ζ-ASB-model.

Are the results comparable with previous studies? According to Heard's paper [26] it was expected for the B_1 index to be the best, the Colless index and the variance of leaf depths to be good and the average leaf depth (Sackin index) as well as the B_2 index to be weak for the IF-diff-model. As only low numbers of leaves were investigated in that study we can only compare it with our results for $n = 30$ (see Figure 13 on the left). Unfortunately, our data contradicts these expectations. B_1 is by far not as good as the Sackin and the Colless index, which perform equally well for the IF-diff-model.

However, our index rankings are in complete agreement with Agapow's studies [1] for the

ASB-model (compared with the continuous-time version used in the paper) as well as for the Brownian (trait-based) model. Furthermore, Blum's results [6] on a different version of the IF-diff-model ranking the Sackin index and the weighted l_1 distance high and the cherry index very low are comparable to our findings on our version of IF-diff.

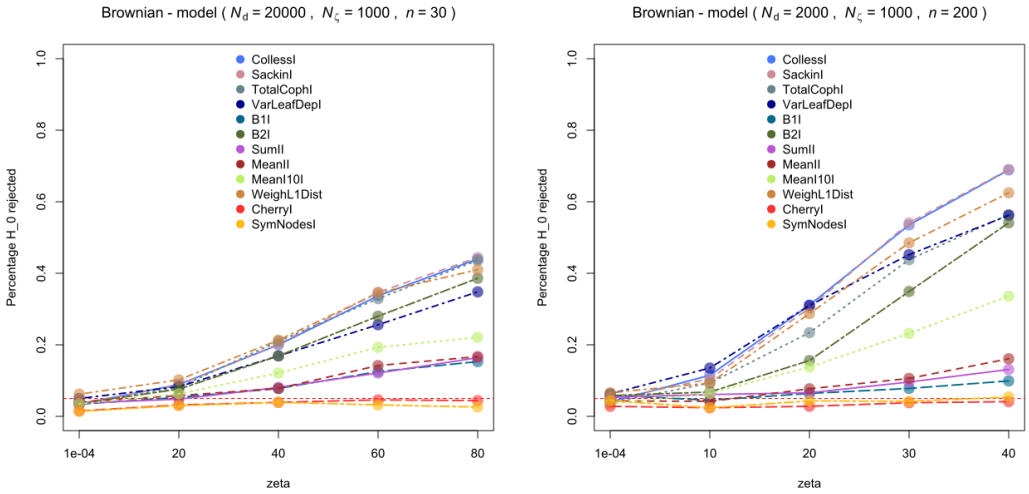


Figure 15: Comparison of the power of balance indices regarding the Brownian model.

Are there consequences for genetic programming? Because of the high run time we decided to use only a low number of leaves for genetic programming. We will use 25 leaves. The performance of so found indices will afterwards be tested for higher numbers of leaves in order to investigate if their power is robust to changes in tree size. Another consequence will be to test only for a smaller amount of trees that were generated under an alternative model than we did now with 5000-7000 trees. Furthermore, it was decided to exclude the DCO-model from further investigations because it was the least interesting and showed a similar balance index performance structure as the IF-models and the Brownian-model. Additionally, one range of ζ values did not have any noteworthy effect at all and can thus not be used to differentiate the performance of balance indices.

4 Genetic Programming

In the field of evolutionary algorithms evolution with mutation and selection is simulated to find and optimize a solution to a given problem. A population consisting of a set of individuals is created, with each of these individuals representing a more or less good solution strategy. One can differentiate between the genotype of an individual, which contains the information (e.g. $(2, 3, 5)$ binary coded as $(0010|0011|0101)$ with 4 bits per number for a polynomial of degree two) on how the individual is structured and can be affected by mutation operators, and the phenotype, which is the actual expression of the genotype (e.g. $p(x) = 2x^2 + 3x + 5$) and can be evaluated with regards to its performance in solving the problem. There are different forms of evolutionary algorithms like genetic algorithms, evolutionary strategies or genetic programming. They mostly differ in the way how they represent or "build" the genotypes of the population and consequently how the mutation operators work.

The underlying optimization problem can be abstract and not analytically solvable, the only thing needed is a *fitness function* f. This function assigns to each possible genotype a fitness value ($\in \mathbb{R}$) describing how good the corresponding phenotype is in solving the problem [3, p. 126]. The fitness value of an individual will then impact how likely it is that this individual produces offspring and that its solution will in some way prevail in the next generations. Defining f properly as a model of the problem is the key to obtaining solutions that really solve the problem in the intended way.

For example, if you want to interpolate a difficult function g using a number of interpolation points $(x_1, g(x_1)), (x_2, g(x_2)), ...$ one could take the least squares approach. The fitness function f would be equal to the sum of squares of the residuals $g(x_i) - ind(x_i)$ with ind being the function that is represented by an individual [3, p. 128]. Now we can see this as a minimization problem and select individuals with small residuals to produce offspring.

In robotics it is for example possible to train the parameters of a neural network for steering a robot using evolutionary algorithms. The difficulty lies in the well thought-out definition of the fitness function that punishes unwanted behavior. If the aim is a far and fast driving 2-wheeled robot then f should include a value for the speed of its wheels. If f was kept like that some robots may only spin on one spot without getting any distance from the starting point. To punish such behavior one could either let the length of the covered track be a positive factor of the fitness function or one could use the difference of speed between the two wheels as a negative factor. Individuals that drive more or less straight most of the time would have better fitness values and therefore be more likely to produce offspring [3, p. 363]. Finding a good definition of the fitness functions is a creative process that sometimes needs to be revisited if there is unwanted behavior that should be excluded.

As the `rgp` package, which is used in the GP runs, assumes a minimizing problem with lower fitness values being favored, we will use minimization problems in our examples or use the term "better" fitness values. Better can then refer to either the lower or higher fitness value depending on the problem. As our fitness function should assess the power of the balance indices we will use $f(i) = 1 - power(i)$ or a slightly modified version.

But how does the whole process look like? In general *generation based* or *generational* evolutionary algorithms can be summarized as follows [3, p. 134].

1.) Initialize a population of solving strategies and calculate their fitness values.

2.) Create a new generation by sequentially generating new individuals through crossover and mutation of parents that were selected depending on their fitness values.

3.) Calculate the fitness of the new individuals and check if a termination criterion is fulfilled. If this is not the case return to step 2.). Otherwise give out the best individual(s).

A lot of this, like the representation and structure of an individual or the selection and mutation processes, is not yet clear. Thus, we will now have a closer look how these points are defined and executed in genetic programming, which is the form of evolutionary algorithms that will be used here.

4.1 Genetic programming and its building blocks

Genetic programming (GP) is the subcategory of evolutionary algorithms that aims to evolve computer programs or functions and was recognized, developed and studied by Koza [27, 28]. Like other evolutionary algorithms it needs a fitness function for the given problem. This problem should be possible to be divided into subproblems for GP to be effective.

For GP it is common to use a (rooted) tree representation as many programming languages represent or interpret programs in this way [3, p. 113]. Similar to the Newick tree format, nested parentheses are converted into a tree shape. The program then represents the genotype of an individual and the phenotype is the behavior of the program and how it executes on given input. The tree consists of building blocks set by the user. They can be divided into the *terminal set* TS for the leaves of the tree and the *function set* FS for the inner nodes [3, p. 109]. The former contains all constants and input parameters that do not require any arguments, while the latter contains a set of functions that can operate on the given input symbols. It is difficult to choose the right size of these sets. The function set should contain the most important functions or terminal symbols to be *sufficient* to represent a proper solution to the problem, but on the other hand it should not be too large as this increases the search space and can make the problem harder to solve.

The terminal set can contain a finite set of constants, but it is also possible to include intervals or other sets with (in-)finite elements. For that placeholders have to be introduced into the terminal set for every sensible domain e.g. $TS = \{D_1, D_2, x\}$ with $D_1 = [-1, 1]$ and $D_2 = \{1, 2, ..., 100\}$. If a certain domain is chosen for a terminal symbol a constant will be drawn from it at random. They are called *random ephemeral constants*. A constant drawn for an individual will not change except under mutation and exactly this value can be inherited by the individual's offspring. There will not be a roll for a new D_1 value for the child individual [3, p. 109].

The `R` package `rgp` that will be used for our GP experiments uses so-called *constant factories* that resemble the domains. They are functions that return a value of the corresponding domain. The user has the option to introduce any probability distribution. Commonly the uniform distribution is used and will also be used here. However, we will round values to two decimal places at least. More detailed information is given in the respective parameter descriptions for the GP stages.

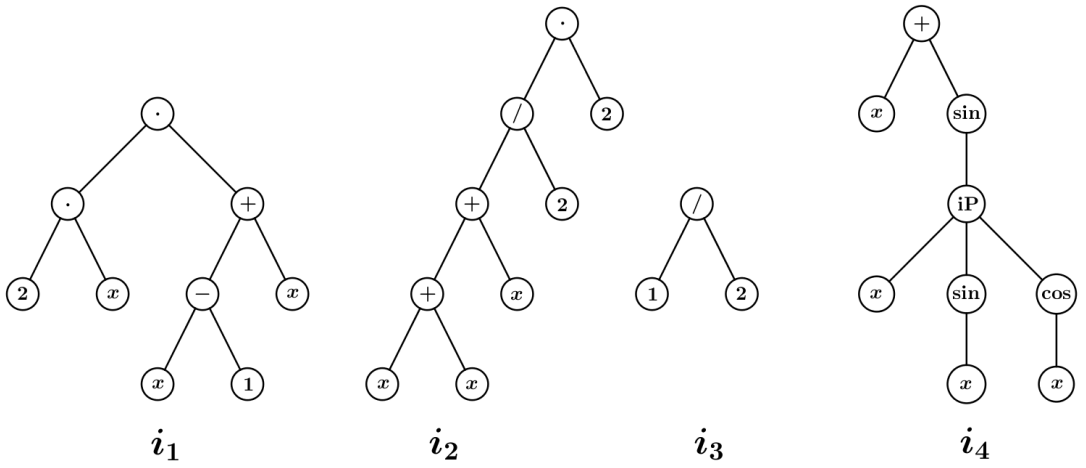


Figure 16: Examples of the tree representation of GP individuals.

Example: Let the building blocks be $TS = \{1, 2, x\}$ and $FS = \{+, -, \cdot, /\}$ and let the goal be to create functions e.g. for an interpolation problem. With these sets there are infinite possible combinations, for example $i_1(x) = (2 \cdot x) \cdot ((x-1)+x) = 4x^2 - 2x$, $i_2(x) = (((x+x)+x)/2) \cdot 2 = 3x$ or simply $i_3(x) = 1/2$ if we randomly choose functions and terminal symbols. Of course FS can also contain functions that have more or less input variables. An example with trigonometric functions and the ifPositive-function can be seen on the right. The function ifPositive (iP) executes the second branch if the first value is positive, and the third branch else. The example thus represents the function $i_4(x) = x + sin(sin(x))$ for $x > 0$ and $i_4(x) = x + sin(cos(x))$ for $x \leq 0$. The tree representations of these functions can be seen in Figure 16.

For this small problem we can derive at least three questions: First, how do we handle problems like dividing by zero e.g. in case of $1/(x - x)$? Second, how do we handle functions with different input and output values? Balance indices for example will take a tree as input and return a numerical value. And third, how can the first generation be initialized?

Closure of the function and terminal set

The general goal is to have a set of functions that work no matter the input value. This is called the *closure* property [3, p. 112]. New versions of functions are introduced that return a default value if the input is not appropriate. For example there is protected or safe division that returns zero in case of dividing by zero. Similarly, one can use a protected version of the square root or the logarithm that uses the absolute value of its input to not have errors due to negative values.

This entails that every function that is introduced to the FS has to be ensured not to produce errors that could stop the whole process of GP. The scripts `AuxiliaryPhylFunctionsForGP.R` and `GPParams*.R` thus contain modifications that remove stop commands and instead have default values returned. Some functions will be mentioned in the respective stage parameter descriptions, but for detailed information on the modification of each function please inspect the scripts on the attached CD.

Strongly typed genetic programming

Strongly typed genetic programming or sometimes only *typed genetic programming* is a GP version in which we can use functions with different in- and output types [36] [3, p. 298]. In our various GP runs we will use numeric, integer and boolean values, trees of format list as well as sets of nodes of the new defined format numeric vector. For the program to know how to build individuals and how to connect each function with the fitting parameters, each function in FS has to be extended by a definition of its in- and output types.

The `rgp` package includes this option. For example an arbitrary balance index would be defined like

$$\text{“}NameOfBalanceIndex\text{”}\% :: \%(list(st(\text{“}list\text{”}))\%- > \%st(\text{“}numeric\text{”}))$$

indicating that it turns a tree into a numeric value. Note that trees are of type “”list (list of edge matrix, number of inner nodes, leaf labeling, ...) and as it is the only parameter of this type, list will be synonymous for tree here. For a more complex example we look at the function vectorOver, which resembles a for-loop by saving the result of a given function for several given input values in a vector. With

$$\text{“}vectorOver\text{”}\% :: \%(list(st(\text{“}nodes\text{”}), st(\text{“}list\text{”}), st(\text{“}character\text{”}))\%- > \%st(\text{“}numericVector\text{”}))$$

it can be declared that the function needs a set of nodes (integer vector), a tree and a character vector as input and will then return a vector containing numeric values. In the definition of the function it can be assumed that the incoming parameters will fit these types. The character should be the name of a function which is using a single node as an input in addition to the tree. For this a constant factory was used that returned different function names. In vectorOver a vector will be created containing the value of this function over all given nodes. E.g. a vector containing the depth of every leaf of a tree can be created with a function call similar to `vectorOver(getLeafNodes(tree),tree,“depthNode”)`.

Similar to the functions all constants and the input have to be marked with their corresponding type as well.

Creating the initial population

There are different ways on how to create the individuals for the first generation. The three most common methods require the user to set a desired depth for the individuals' tree representation [3, p. 118] [27]. Then the initialization goes as follows for these three methods (see Figure 17):

i: The *full* method starts from the root and uses elements of the function set until the desired depth -1 is reached, then only terminal symbols are attached. This creates a tree with equal depth for every leaf.

ii: The *grow* method starts from the root and uses functions or terminal symbols. At the desired depth only terminal symbols are used. This creates a tree with a certain maximal depth of the leaves, but leaves are allowed to have a smaller depth.

iii: The *50-50-rule* also known as *ramped-half-and-half* creates half of the trees with the first and the second half with the second method.

If there are several desired depths the population will be commonly divided into equal parts for each depth value. Then one of the above mentioned methods is applied to each subset.

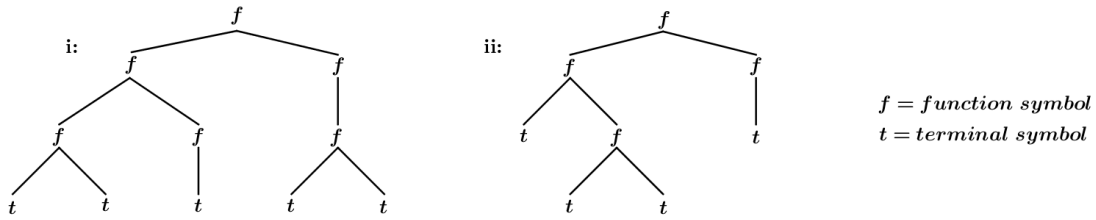


Figure 17: Initialization of trees with maximal depth 3 using method i and ii. In the second method not all leaves have that maximal depth.

For our GP runs with the `rgp` package we will use the "ramped-half-and-half" method with the default maximal depth of 8. Although the default implementation of the *full* method tries to reach the given depth, it will in our case often produce trees containing leaves with smaller depth when it has no choice but to use a terminal symbol. For example if it uses a balance index as a function symbol the direct child will always have to be a terminal symbol (e.g. the input tree) because there are no other functions available that have a tree as output. The program would not replace the balance index with another function symbol as this could cost a lot of time and would not guarantee success. However, this behavior of the program is beneficial for us as it keeps the individuals less complex.

4.2 Evolutionary operators: mutation and selection

Now, with the definition of the building blocks done, we go over to the *evolutionary* or *genetical operators*. Again these have to be modified for typed genetic programming, to only build sensible programs with matching types.

If we have a population of individuals and assume that their fitness has already been calculated, the next step is to create new individuals for the next generation. For that a mutation operator has to be chosen with each operator having a given probability to be chosen. There normally are two different forms, crossover and repeated point mutation, although the latter is often referred to only as mutation. Either of these operations require at least one parent individual. And this parent has to be chosen from the already existing population via selection that favors individuals with better fitness values.

Selection

Let $(f_1, f_2, ..., f_{n_{\text{pop}}})$ be the fitness values of a population of size n_{pop}. Then there are different methods on how to select an individual based on its fitness [3, p. 129]. For example the *fitness proportional selection* where each individual has a probability equal to its proportion of the total fitness of the population $f_i / \sum_j f_j$. There are some similar versions in which the influence of chance is lowered by giving each individual its rounded down expected number of offspring

$\lfloor f_i / \sum_j f_j \cdot n \rfloor$. The remaining free places of the next generation are then selected again by a modified fitness proportional selection or by a ranking. However, all of these methods have two disadvantages in common. First, they need positive fitness values and second if all fitness values are more or less equally high e.g. (999,997,1002,996,...) the pressure of selection is nullified because it is equal to a random selection under a uniform distribution. All these disadvantages could be avoided by scaling or other modifications of the fitness function, but there is a different form of selection that does not need further modifications of the fitness function:

The *tournament selection* method relies on the comparison of the fitness values of a small group [3, p. 132]. An example can be seen in Figure 18. A given number of individuals (tournament size n_{tour}) are drawn randomly from the given population. These individuals compete and the one with the best fitness value will be selected. This method does not suffer from the above mentioned disadvantages, but it is vulnerable to an unlucky draw that did not include the best individuals.

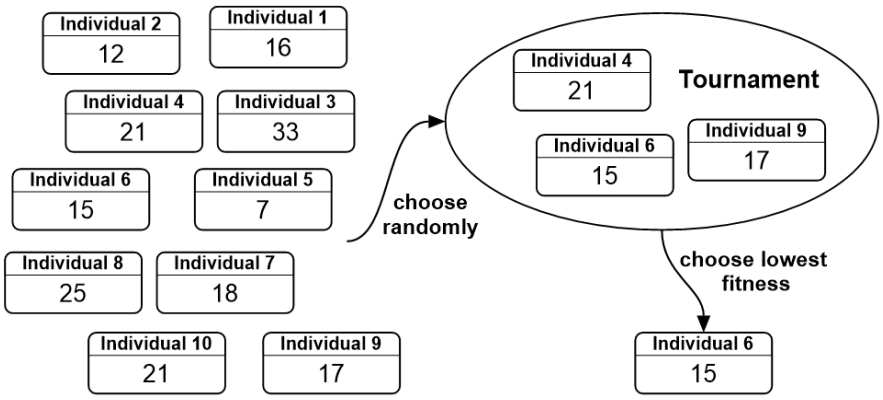


Figure 18: Example of tournament selection for a minimization problem with tournament size 3. Each individual is depicted with its fitness value.

Thus, this method (or other methods as well) is often linked with an *elite selection*. The elite will contain the n_{elite} best individuals and is updated for every new generation. Therefore, there will not be a loss of good performing individuals.

In our GP runs we will use tournament selection in combination with an elite that will here be called *archive* with size n_{arc}. Only the non-archive part of the generation will be replaced with $n_{\text{pop}} - n_{\text{arc}}$ new individuals in every step (see Figure 20 later with explanations).

Crossover

The crossover of two individuals is an exchange of subtrees and is shown in Figure 19. Every node of the first individual's tree has a probability to be affected by a crossover. If that happens a random subtree of the second indiviudal's tree is chosen to replace the subtree rooted in this node. The replacement can be any subtree ranging from the complete tree of the second individual to a single leaf. For typed genetic programming it has to be additionally ensured that this subtree has the same output type as the first one. The modified version of the first individual will then be returned as the result of the crossover [3, p. 122].

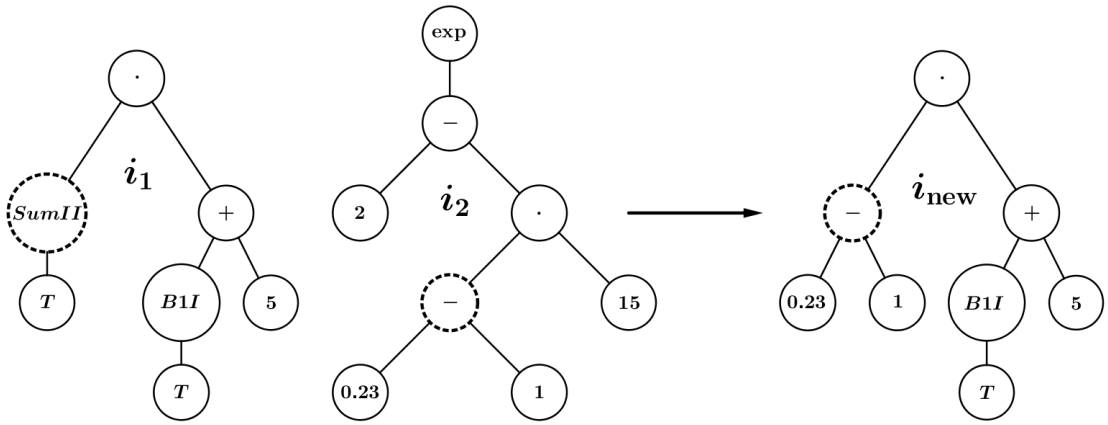


Figure 19: Example of a crossover of two indiviudals. The roots of the subtrees chosen for the crossover are marked.

Due to the partition of the population into archive and non-archive we will use two different forms of crossover. The two forms will only differ in the population pool from which the parent individuals will be drawn. For the first form we select one archive individual and one arbitrary and the second form uses two arbitrary individuals from the complete population. The former is the default crossover method in the `rgp` package and will get a higher probability because it prioritizes the better archive individuals. However, the latter version was additionally introduced to provide more variety as it makes it possible that a crossover happens to two non-archive indiviudals as well as that a non-archive individual will gain a subtree of an archive individual. The probability to choose these crossover operators for the creation of the next new individual are $p_{\text{cross-arc}}$ and $p_{\text{cross-noarc}}$, respectively.

We will use the default crossover probability of the `rgp` package of 0.1 for every node in Stage 1 and 3. The `rgp` crossover function terminates the crossover of a chosen node, if the output type of the subtree that is randomly picked in the second tree does not match the output type of the chosen node, because a search for a fitting subtree could take too long. Therefore, we will increase the node crossover probability to 0.2 for Stage 2 to compensate for the significantly larger function set that has a wider range of output types which therefore makes a crossover termination more probable.

Mutation

Mutation refers to the modification of one single individual to create a new offspring individual [3, p. 125]. Every node of the individual will mutate with a given probability. The mutation replaces the subtree rooted in this node with a newly constructed tree (see initialization of individuals). This new tree must have the same output type for typed genetic programming [3, p. 122]. The probability to choose mutation as a genetical operator for the creation of the next new individual is p_{mut}.

In the `rgp` package each node has a default probability of 0.1 to be mutated and the replacement tree has a maximal depth of 5. These settings were used for the GP runs in Stage 1 and 3. Similar to the crossover the `rgp` mutation function tries to build a replacement tree with

the same output type, but if it fails it will terminate the mutation of the chosen node. Thus, for Stage 2 the node mutation probability was increased to 0.2 because the number of functions and their range of types was higher, leading to a higher chance of building unfitting replacement trees.

Reproduction and further operators

Reproduction refers to returning an unmodified copy of the selected parent individual [3, p. 126]. As in our case crossover and mutation both have a probability to not modify the individual at all and as we additionally use an elite that saves the best individuals, it is not necessary to introduce reproduction as a further evolutionary operator.

There are more operations that can be applied to GP individuals. However, we will here only use the standard set of operators (crossover, mutation as well as indirectly reproduction) because we want to explore if a simple genetic programming setup can already be successful.

5 The experimental GP setup

The `rgp` package provided the basic structure for the genetic programming process. For the initialization and the evolutionary operators predefined functions could be chosen. We will use the archive-based GP version in our experiments. Its general process was provided as well, but it was further modified and enhanced by

- installing safety measures (data backup of archive individuals and their fitness values for the current generation) and a recording of development statistics (minimal, average and maximal fitness),
- introducing a second crossover operator with two random individuals as mentioned in Section 4.2 (the first crossover operator was given by default)
- and by implementing a parallelization of the fitness calculations for all new individuals.

The complete process of the archive-based genetic programming can be seen in Figure 20 or in the scripts `GPParams*.R`. All settings not yet defined will be explained in Section 5.1.1.

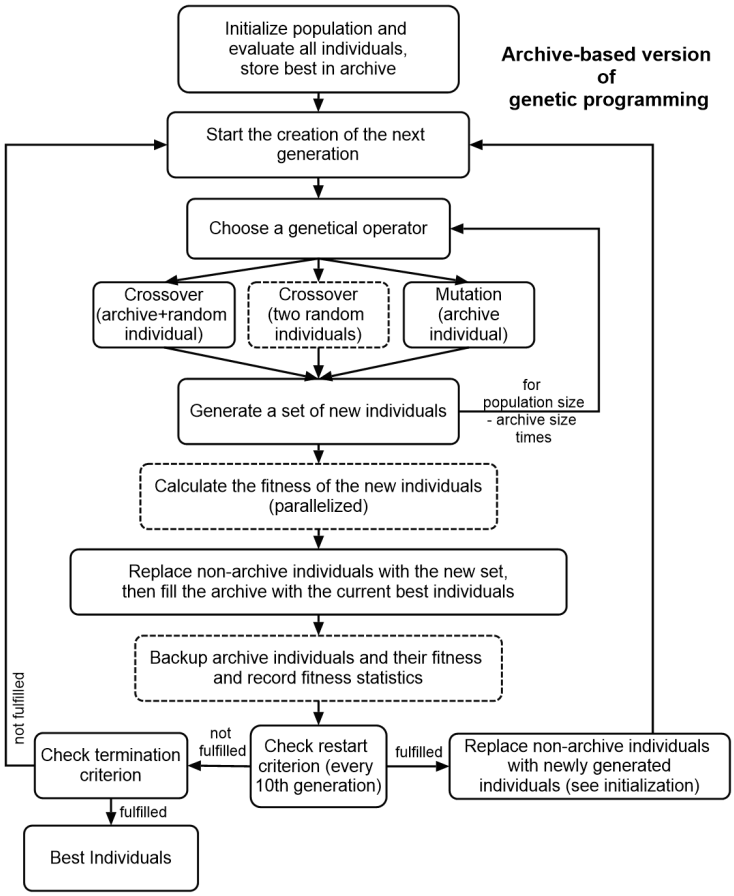


Figure 20: An overview of the procedure of archive-based genetic programming used in this thesis. The genetical operators will be chosen with probability $p_{\text{cross-arc}}$, $p_{\text{cross-noarc}}$ and p_{mut}, respectively. Modifications are marked with dotted lines.

5.1 Tasks for Stage 1 & 2

In order to investigate how good genetic programming is in developing new balance indices we will use two tasks that explore different aspects of the balance index optimization problem. There obviously is a huge variety of possible applications, but the number of tasks will be limited to two in order to keep everything clear as they are later also used for both stages.

(A) Best BI for both ASB and IF & Brownian: In the comparison of established indices in Section 3.2 we could see that there was a shift in the performance of certain balance indices depending on the model type. Due to that the first task is to find a balance index that performs well on both the ASB-model and the IF- and Brownian models. In lack of an established mixture we will generate half of the trees under the ASB-model and the other under a uniform mixture of the IF-both, the IF-diff and the Brownian model.

(B) Best BI for IF-diff: In contrast to the first task we also wanted to see how good GP is in optimizing balance indices for only one specific model. The IF-diff-model was selected because it showed a clear distinction in the performance of different indices that did not vary for increasing numbers of leaves and because it has already been used in literature. Furthermore, the IF-diff-model does not produce more asymmetric or symmetric trees for different ζ values, such that we will only have to test for one range of ζ. A mixture of trees for two different ζ values was used.

In order to provide the data on which the fitness calculations can be based on we first generated 10,000 trees for every ζ of every model in the composition. Then we created 100 files containing 2400 (or 1200 for the first trial runs) trees, each according to the composition in Table 4. For every evolution step in GP one of these 100 files will be drawn at random as a data set. This should serve as a compromise between time efficiency and preventing overfitting if the data set does not change at all.

Table 4: Tree composition for Task A and B ($\sum = N_\zeta = 2400$). For every model it will be listed how many trees will be generated for the corresponding ζ value.

A			B	
ASB	600, $\zeta = 0.76$	600, $\zeta = 1.24$		
If-both	200, $\zeta = 0.55$	200, $\zeta = 0.45$		
If-diff	400, $\zeta = 2.5$		1200, $\zeta = 2.5$	1200, $\zeta = 3$
Brownian	400, $\zeta = 80$			

The fitness of an individual i was then calculated using

$$f(i) = 1 - power_{\text{comp}}(i)$$

with $power_{\text{comp}}(i)$ being the percentage of alternative trees of one data set for which H_0 was rejected (a weighted mean of the powers for the different models and ζ-values). As we use a

composition of different tree models we cannot directly link the percentage of times H_0 is rejected to a certain model. Thus, we will have to see in the end how well the newly constructed indices actually perform for each single model.

5.1.1 The parameter settings

To explore the functionality and to find weaknesses in the performance a first set of GP runs (I) was used. The goal was to use the experiences gained in Set (I) to optimize the execution of the second set of GP runs (II). Here is a description of the parameters of each set:

Set (I) This set of runs was incomplete in the parameter settings with a mutation function with a very low probability to affect an individual and different probabilities for the genetical operators were tried out as well. Furthermore, it did not make use of small added features of the `rgp` package. Sensible restart conditions could only be implemented after the first runs: Every tenth generation is tested if the standard deviation of the fitness values falls below a threshold of 0.019. A small variance within the fitness values indicates low diversity within the population which often shows that the population is stuck in one area of the space of possible balance indices – a situation in which a restart can help to increase diversity and with that the chance of finding new well performing individuals. The aim was to make restarts possible, but not to have them happen too often. The threshold of 0.019 was first chosen randomly for both Task A and B from the range of the standard deviations of the first final generations (Ai) 0.017 and Aii) 0.014 as well as Bi) 0.017 and Bii) 0.021), but proved to be a good choice in the following runs of both stages. All in all, the following parameters were used in Set (I):

$n = 25$,	max. duration = 68 h
$N_d = 2000$,	$N_\zeta = 1200$,
$p_{\text{cross-arc}} = 0.7$ or 0.5,	$n_{\text{pop}} = 50$,
$p_{\text{cross-noarc}} = 0.24$ or 0.25, ,	$n_{\text{arc}} = 10$,
$p_{\text{mut}} = 0.06$ or 0.25, ,	$n_{\text{tour}} = 3$ (2 or 3 for archive).

Set (II) We used the following parameters for all stages for set (II). Extinction prevention[5] was activated and the developed restart conditions were copied from Set (I).

$n = 25$,	max. duration = 68 h
$N_d = 4000$,	$N_\zeta = 2400$,
$p_{\text{cross-arc}} = 0.6$,	$n_{\text{pop}} = 60$,
$p_{\text{cross-noarc}} = 0.16$,	$n_{\text{arc}} = 10$,
$p_{\text{mut}} = 0.36$,	$n_{\text{tour}} = 3$ (2 for archive).

[5] If the feature "extinction prevention" is activated in the `rgp` package, the algorithm prevents the introduction of duplicate individuals during the initialization of the first generation by successively adding only distinct individuals.

We discovered that all of the runs progressed extremely differently, more depending on a lucky seed than on variations in the parameter settings, such that results from (I) were sometimes as good or even better than runs from the second set. As a consequence we will combine the results of both sets to gain access to all well performing individuals, although it will lead to a longer list of runs that have to be described and examined.

All GP runs had the time limit of 68 hours as a termination criterion because the run time was limited to 72 by the university's computing network. 4 hours were considered a safety margin because depending on the complexity of the population the calculation of new generations can take a half up to several hours. It showed to be necessary and sometimes even too small as some runs could not finish completely. For these cases the above mentioned safety measures were installed such that no results were lost.

5.1.2 Selection of candidates for the final comparison

How much does a fitness value tell about the power of the actual index? To answer this the fitness values of the best established balance indices were calculated. As can be seen in Figure 21 quality differences can certainly be detected using these fitness values, however there still is a variance.

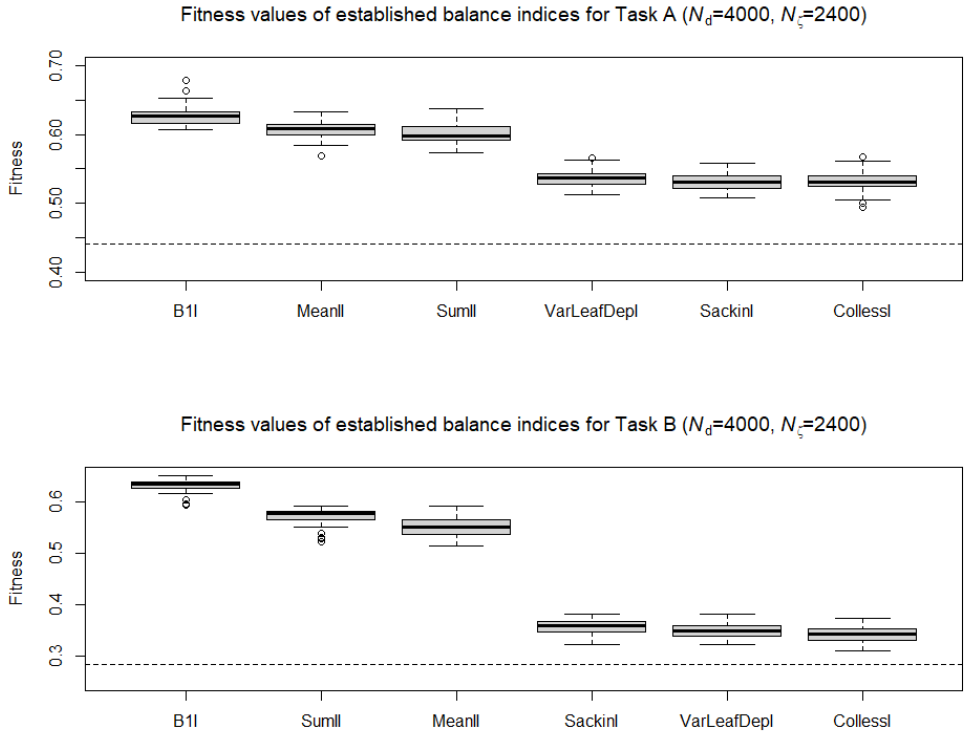


Figure 21: Boxplots for different established balance indices based on 50 fitness values each using the fitness function of Stage 1 Task A and B. GP individuals that have a fitness of less than 0.44 or 0.285 for Task B (dotted line) will be selected for the final comparison.

All individuals that have a fitness of less than 0.44 for Task A or 0.285 for Task B will be listed for the respective runs. However, only individuals that present a new idea on measuring tree balance will be enumerated as $a_1, a_2, ...$ or $b_1, b_2, ...$ and used in the final comparison, excluding individuals that consist only of one established balance index (with or without a factor or constant summand). The thresholds of 0.44 and 0.285 have been determined such that at most the ten best individuals will be selected for the final comparison. This ensures that the comparison with the best established indices will not be overloaded. Furthermore, we wanted more candidates for Task A as it seemed more promising.

The archive data of all runs can also be found on the attached CD (`archive_data.zip`).

6 Stage 1: Established balance indices as building blocks

In this first approach we will use the established balance indices and the number of leaves as building blocks that can be linked with a simple set of numerical functions. We thereby extend the idea of a previous study that compared the performance of linear combinations of pairs of balance indices [21, p. 56]. The building blocks are defined as follows. Please check the scripts `GPMainStage1.R` as well as `GPParamsStage1.R` for more detailed information.

Terminal symbols TS**:** Constant sets for $[0, 1]$ and $[1, 5]$ rounded to 2 decimal places as well as $\{10, 11, ..., 100\}$ combined with the input parameter *tree*. A uniform distribution will be used to draw constants from each domain. As the `rgp` package also requires a constant for the type "list" (tree) another constant factory was added that only returns the T_3^{cat}, but it was modified by a probability weight 0 to not be chosen over the input tree.

Function set FS**:** The standard operations $+, -, \cdot$ additional to safe versions of division, square root, natural logarithms as well as the exp-function. Furthermore, a function that returns the number of leaves of a tree and the list of balance indices that was used for the comparison (CollessI, SackinI, TotalCophI, VarLeafDepI, B1I, B2I, SumII, MeanII, MeanI10I, WeighL1Dist, CherryI and SymNodesI). The cherry and symmetry nodes index were not excluded to be able to explore if they could still be useful in combination with other indices.

We will here give an overview of the runs and their development and statistics (see Figure 22 and 23). The minimal fitness of each run is rounded to three decimal places. The global minimum is 0.419 for Task A as well as 0.257 for Task B. Its proximity was reached by multiple runs of both Set (I) and (II) for Task A, but only runs of Set(I) got near the global minimum for Task B. All in all, this stage provided a number of promising candidates.

Although the run time was the same for all runs their number of calculated generations differed greatly sometimes. This can partly be ascribed to the complexity of the individuals as the fitness calculation of complex formulas takes longer, but for some drastic differences – less than hundred generations in comparison to more than thousand – the reason was the number of available computing cores.[6]

6.1 Results of the GP runs for Task A (Stage 1)

For Task A both sets provided a total of five candidates. They are simply structured, using three different balance indices at most and only the standard operations $+, -$ as well as $\cdot$.

Set (I) Four runs were started and ran on 15 cores (except Repetition 2 on 47 cores). As mentioned above only after the first two runs a sensible restart condition was introduced because then it was possible to measure the standard deviation of the fitness values. For A i) produced better values and it was restarted with the new restart criterion (Repetition 1). The process of loading the trees for the fitness calculation was later further optimized and thus the second repetition was launched as well and reached significantly more evolution steps (Repetition 2).

[6] It was not possible to influence on which nodes of the university's computing network BRAIN the runs were started due to limited accessibility. Thus, the runs used either 15 or 47 cores. To keep this influence transparent the number of cores will be listed for each run.

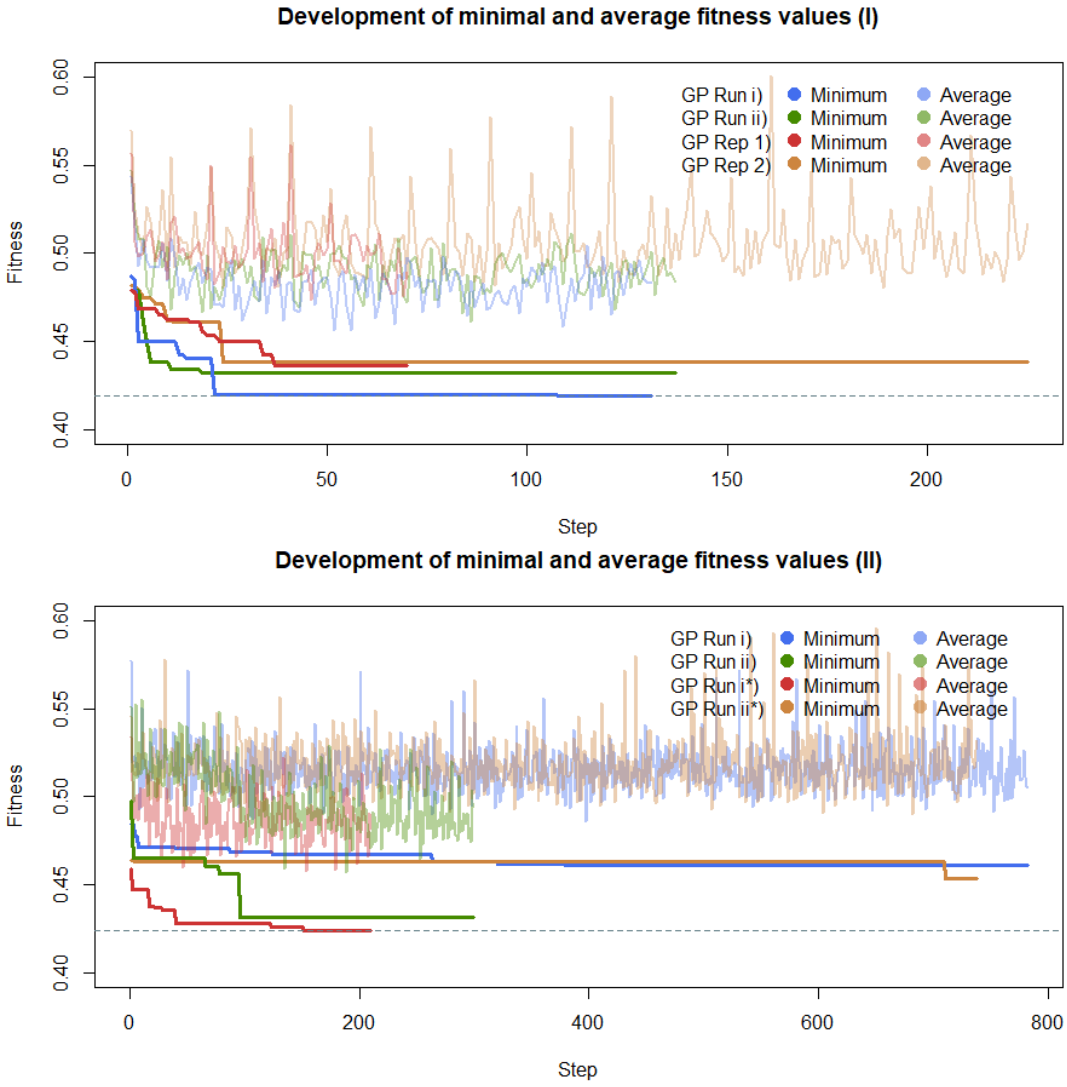


Figure 22: Development of fitness values over time for Task A. The global minimal value of each set is marked with a dashed line. (I): Note that the x-axis was cut for visualization purposes and does by far not show the total number of steps of Repetition 2. The minimal value of Repetition 2 stays nearly constant from step 40 onward and does not reach the global minimum.

i) Finished with 131 evolution steps and 0 restarts during the process (on 15 cores). Minimal Fitness: 0.419, BI candidates:

$$a_1(T) = SackinI(T) + CollessI(T) \cdot MeanI10I(T)$$
$$a_2(T) = SackinI(T) + SackinI(T) \cdot MeanI10I(T)$$

ii) Finished with 137 evolution steps and 0 restarts during the process (on 15 cores). Minimal Fitness: 0.433, BI candidates:

$$a_3(T) = MeanI10I(T) \cdot CollessI(T)$$

Repetition 1) Finished with 70 evolution steps and 5 restarts during the process (on 15 cores). Minimal Fitness: 0.437, BI candidates: a_3

Repetition 2) Finished with 1393 evolution steps and 137 restarts during the process (on 47 cores). Minimal Fitness: 0.436, BI candidates: $WeighL1Dist(T)$

Set (II) The second set contains four runs as well, the last two with the correct I' implementation.

i) Finished with 782 evolution steps and 17 restarts during the process (on 47 cores). Minimal Fitness: 0.461

ii) Finished with 300 evolution steps and 2 restarts during the process (on 47 cores). Minimal Fitness: 0.431, BI candidates:

$$\begin{aligned} a_4(T) &= SackinI(T) + SumII(T) \cdot getLeaves(T) \cdot 0.14 \\ &= SackinI(T) + SumII(T) \cdot n \cdot 0.14 \end{aligned}$$

i*) [7] Finished with 210 evolution steps and 0 restarts during the process (on 47 cores). Minimal Fitness: 0.424, BI candidates:

$$a_5(T) = SumII(T) + (SackinI(T) - B1I(T))$$

ii*) Finished with 738 evolution steps and 24 restarts during the process (on 47 cores). Minimal Fitness: 0.454

6.2 Results of the GP runs for Task B (Stage 1)

The runs for this task also provided some candidates. Their list is shorter though because many possible candidates were omitted as the were only normal established balance indices. Interesting is the individual b_2 of Set (II) i*) as it uses the exp-function. This could provide problems for differing numbers of leaves as the index value ranges will vary as well. Therefore, we will have a closer look at this candidate in the final comparison.

Set (I) The runs of this set are similar to Set(I) for Task A. For Task A we could not see any improvement in the repetition runs, however, for this task we can see in Figure 23 that the global minimum was only reached by the optimized fourth run (Repetition 2).

i) Finished with 140 evolution steps and 0 restarts during the process (on 15 cores). Minimal Fitness: 0.268, BI candidates:

$$b_1(T) = MeanI10I(T) + CollessI(T)$$

ii) Finished with 152 evolution steps and 0 restarts during the process (on 15 cores). Minimal Fitness: 0.285, BI candidates: $CollessI(T)$

[7] Runs marked with * used the corrected implementation of I'. For more detailed information see the Remark 7.2.1.

Repetition 1) Finished with 86 evolution steps and 2 restarts during the process (on 15 cores). Minimal Fitness: 0.287

Repetition 2) Finished with 1443 evolution steps and 107 restarts during the process (on 47 cores). Minimal Fitness: 0.257, BI candidates: $CollessI(T)$

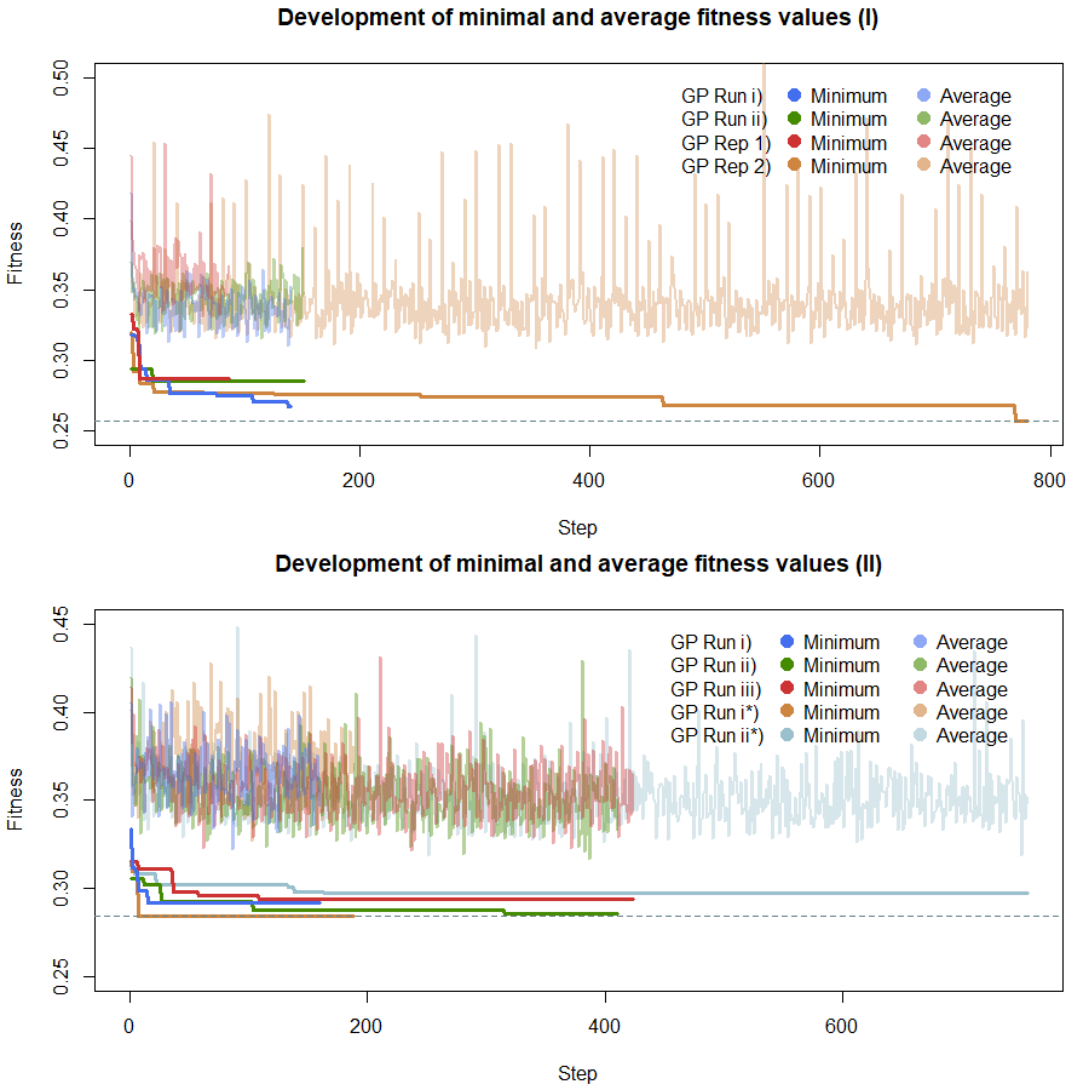


Figure 23: Development of fitness values over time for Task B. The global minimal value of each set is marked with a dashed line. (I): Note that the x-axis was cut for visualization purposes and does by far not show the total number of steps of Repetition 2.

Set (II)

i) Finished with 160 evolution steps and 0 restarts during the process (on 47 cores). Minimal Fitness: 0.292

ii) Finished with 410 evolution steps and 5 restarts during the process (on 47 cores). Minimal Fitness: 0.286

iii) Finished with 424 evolution steps and 1 restart during the process (on 47 cores).
Minimal Fitness: 0.294

i*) Finished with 188 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.284, BI candidates:

$$b_2(T) = 3.61 \cdot MeanII(T) + exp(CollessI(T) - MeanII(T))$$

ii*) Finished with 755 evolution steps and 12 restarts during the process (on 47 cores).
Minimal Fitness: 0.298

All in all, for both tasks the individuals were very simply structured and there was no need to do simplifications by hand. As explained in Section 4.1 small individuals can be created during the initialization or even just a single balance indices. Due to the fact that a lot of balance indices already are quite powerful for both tasks it is not improbable for them to become archive individuals. In that case they will more likely be a template for new individuals and thereby give rise to offspring that are less complex on average – an influence that could explain the observation atleast partly.

7 Stage 2: Auxiliary functions as building blocks

This second stage uses auxiliary functions as building blocks to potentially construct new balance indices. This requires a considerably more complex set of building blocks. Please check the scripts `GPMainStage2.R` as well as `GPParamsStage2.R` for more detailed information.

Terminal symbols TS**:** Constant sets for $[0, 1]$ and $[1, 5]$ rounded to 2 decimal places as well as $\{10, 11, ..., 100\}$ combined with the input tree T. A uniform distribution will be used to draw constants from each domain.

As the `rgp` package requires a constant for each different in- or output type more constant factories had to be added. As in the first stage T_3^{cat} was returned for type "list". Furthermore, three constant factories were introduced that each returned function names as type "character" of different sets of functions: operations that modify two vectors, a single vector as well as operations that use a single node as input.

A few further constant factories had to be added. All constant sets containing values that should not actually be used for individuals were modified with a probability weight of 0. This worked well for all but the constant factory for type "list" as T_3^{cat} was built into several individuals.

Function set FS**:** The function set is signifcantly larger than for Stage 1. There are more functions that operate on a larger range of in- and output types.

The standard operations $+, -, \cdot$ additional to safe versions of division, square root, natural logarithms as well as the exp-function were used as in Stage 1. Decision making functions like $ifPositive$ or if were available. All of the auxiliary functions of the balance indices (see script `AuxiliaryPhylFunctions.R`) additionally to a function that measures the distance of two nodes (number of edges on the unique path between the two nodes) were added as well.

The aim was to be able to construct every established balance index using these building blocks. Due to the fact that all indices collect the results of a function for several nodes, the function $vectorOver$ was introduced (see Section 4.1). As it creates a numerical vector as output functions have been introduced that can modify single or pairs of vectors as well as safe versions of sum, mean, median and variance. As node sets were required to not be predetermined, we use functions that dynamically retrieve either the root, all nodes, all interior nodes or all leaves from a given tree.

With all these functions at our disposal nearly all established balance indices can be recreated with only slight differences. For example
$SackinI(T) = sum(vectorOver(getInnerNodes(T), T, \text{“}NumberOfDescLeaves\text{”}))$.

7.1 Results of the GP runs for Task A (Stage 2)

By comparing the Figures 24 with the corresponding figures of Stage 1 it becomes apparent that the GP algorithm performed significantly worse. It was most probably overwhelmed with the amount of functions available. Even final fitness values did not reach the minimum from Stage 1 and were too high to indicate a viable candidate. Although being a promising approach be-

cause there could have potentially been new ideas and connections between established auxiliary functions, it does hold up to the first stage.

It could potentially be more successful with a longer run time and a greater population. These plans can not be realized within the time limit of this master thesis and with the computing resources available. Additionally, the function set could be reworked to sort out at least a few functions e.g. some vector modifications or the exp function.

Nevertheless, genetic programming proved to be actually successful in constructing sensible measurements. The final individuals for Task A with the best fitness were all sensible e.g. $mean(getDistSet(getAllNodes(T), T))$ the average distance of all pairs of nodes, just not powerful enough to keep up with the results from Stage 1. Some of these interesting results will be listed below even though they are no candidates for the final comparison.

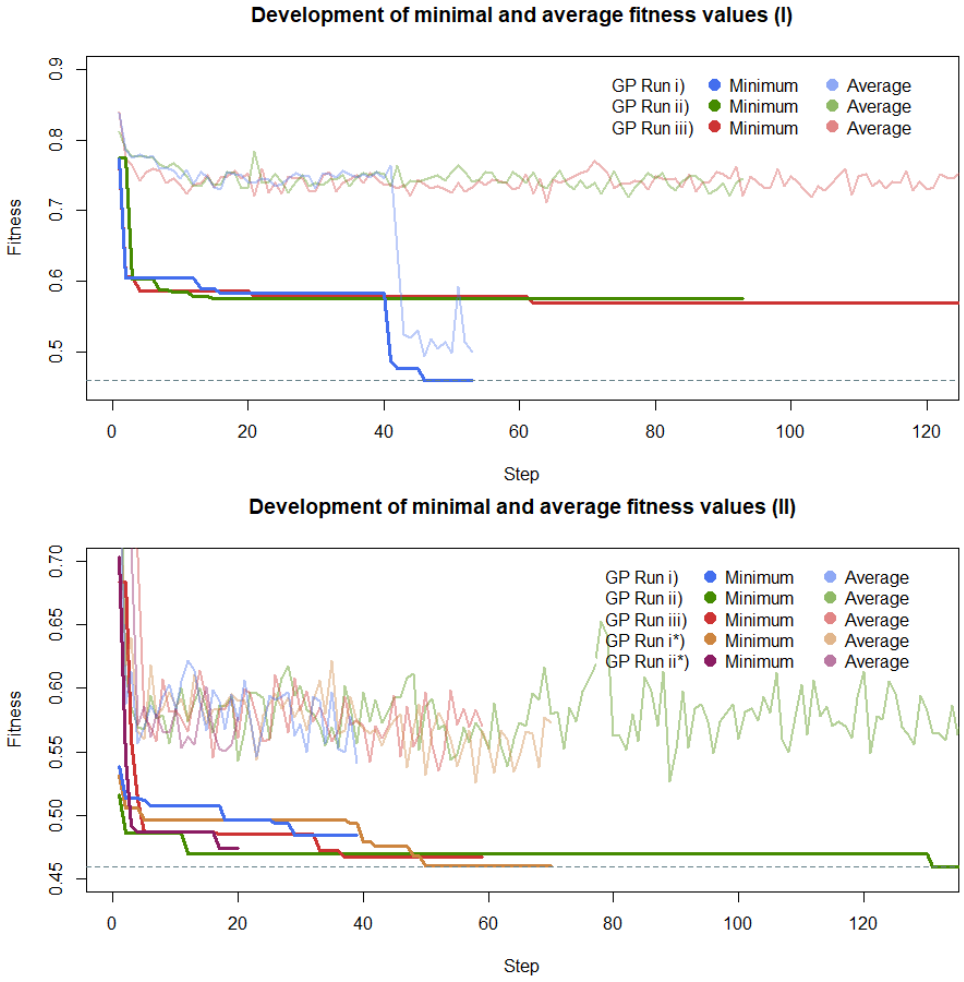


Figure 24: Development of fitness values over time for Task A. The global minimal value of each set is marked with a dashed line. (I): Note that the x-axis was cut for visualization purposes and does by far not show the total number of steps of run iii). The minimal fitness of run iii) stays constant and at around step 2300 falls to its minimum 0.476. (II): Note that this x-axis was cut as well and does by far not show the total number of steps of run ii).

Set (I) i) Finished with 53 evolution steps and 4 restarts during the process (on 15 cores).
Minimal Fitness: 0.459, best individual: $mean(getDistSet(getAllNodes(T), T))$

ii) Finished with 95 evolution steps and 1 restart during the process (on 15 cores).
Minimal Fitness: 0.576, best individual: $maxDepthLeaf(T, getRoot(T))$

iii) Finished with 2413 evolution steps and 17 restarts during the process (on 47 cores).
Minimal Fitness: 0.478, best individual: $var(getDistSet(getAllNodes(T), T))$

Set (II) i) Finished with 39 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.485, best individual: $var(getDistSet(getAllNodes(T), T))$

ii) Finished with 1477 evolution steps and 0 restarts during the process (on 15 cores).
Minimal Fitness: 0.459, best individual:
$mean(vectorOver(getInnerNodes(T), T, \text{“}depthNode\text{”}))$

iii) Finished with 59 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.468, best individual: $mymean(getDistSet(getInnerNodes(T), T))$

i*) Finished with 70 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.460, best individual: $myvar(getDistSet(getInnerNodes(T), T))$

ii*) Finished with 20 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.474, best individual: $mymean(getDistSet(getAllNodes(T), T))$

Astoundingly run ii) from Set (II) completed significantly more evolution steps than the other runs despite having less computing cores. This was double checked. A possible explanation could be that all other runs used the function *getDistSet* which gets the distance of every pair of the input nodes and thus has squared run time in comparison to *vectorOver* used in run ii) with linear run time. The real reason will remain speculative because an overview of all individuals over the generations would be needed to explore this assumption instead of only the currently available archive of the last generation.

7.2 Results of the GP runs for Task B (Stage 2)

The results for Task B were rather disappointing either because the solutions did not reach a low enough fitness value to consider them for the final comparison or because even the best solutions did not use sensible node sets. As the nodes could be arbitrarily labeled a fixed set of concrete node labels (e.g. $getLeafNodes(T_3^{cat}) = \{1, 2, 3\}$ which many individuals contained) is equivalent to drawing random nodes. For the node enumeration from the ζ-models used here it is common that the leaves are not labeled with $1, ..., n$ and thus it cannot even be ensured that predetermined nodes belong to a certain node set like the interior nodes or the leaves. It is not clear why the genetic programming algorithm so often used the constant tree T_3^{cat}, which had a zero probability weight as well, instead of the input tree T for this task as it proved to be no problem for Task A. Anyhow, there were individuals with fitness values falling below the threshold of 0.295. They are listed below with their simplifications and reasons why they will not be used in the final comparison. As it is interesting to see other results as well several more indices are listed below.

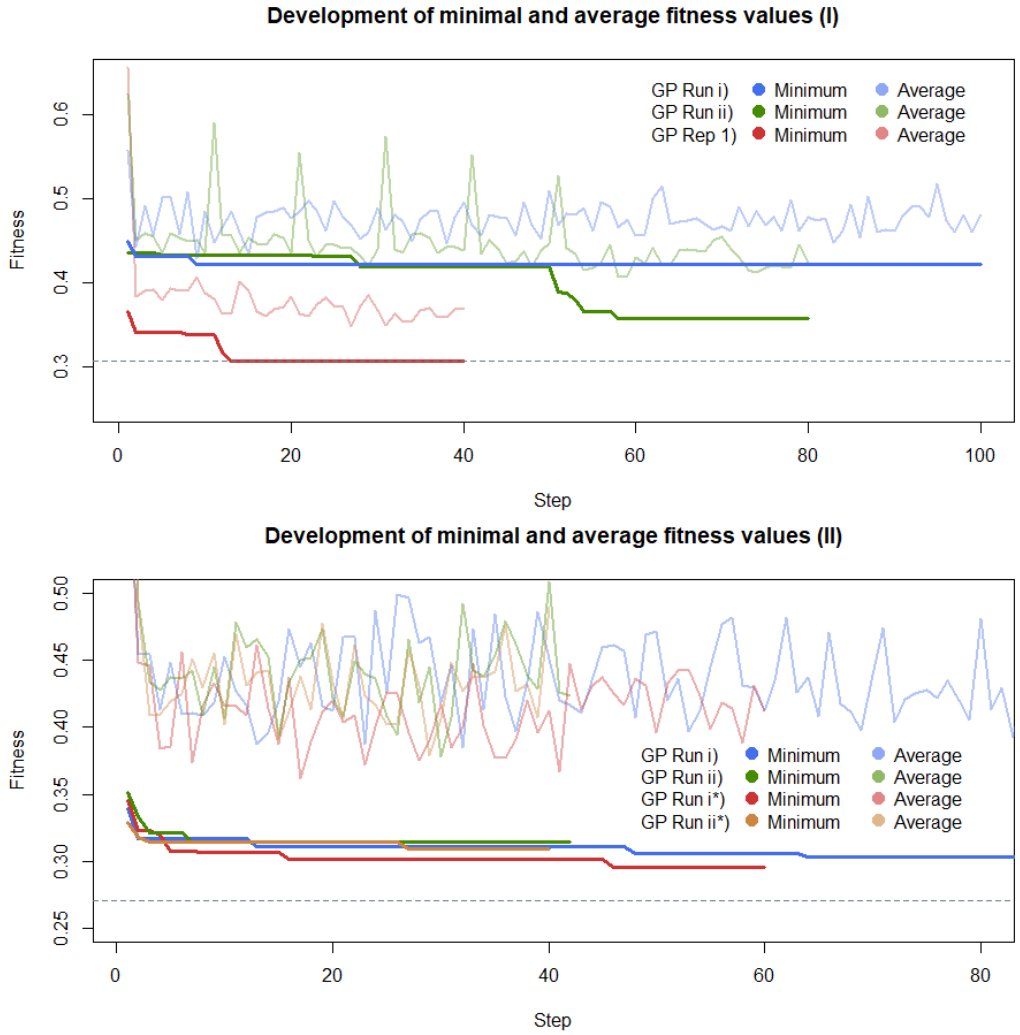


Figure 25: Development of fitness values over time for Task B. The global minimal value of each set is marked with a dashed line.

Set (I) i) Finished with 100 evolution steps and 0 restarts during the process (on 15 cores).
Minimal Fitness: 0.422

ii) Finished with 80 evolution steps and 5 restarts during the process (on 15 cores).
Minimal Fitness: 0.357

Repetition 1) Finished with 40 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.307, best individual: $mysum(getDistSet(getAllNodes(T), T))$

Set (II) i) Finished with 1204 evolution steps and 0 restarts during the process (on 15 cores).

Minimal Fitness: 0.270, BI candidates:

$$\begin{aligned} b_3(T) &= sum(modNumVec(\text{“}-\text{”}, vectorOver(getAllNodes(T_3^{cat}), T, \text{“}depthNode\text{”}))) \\ &= sum(modNumVec(\text{“}-\text{”}, vectorOver(\{1,2,3,4,5\}, T, \text{“}depthNode\text{”}))) \\ &= sum(vectorOver(\{1,2,3,4,5\}, T, \text{“}depthNode\text{”})) \\ b_4(T) &= sum(mNV(\text{“}rmZeros\text{”}, vectorOver(getAllNodes(T_3^{cat}), T, \text{”}depthNode\text{”}))) \\ &= sum(modNumVec(\text{“}rmZeros\text{”}, vectorOver(\{1,2,3,4,5\}, T, \text{“}depthNode\text{”}))) \\ &= sum(vectorOver(\{1,2,3,4,5\}, T, \text{“}depthNode\text{”})) \end{aligned}$$

In the simplification we used the following properties: the node set of a T_3^{cat} will normally be $\{1,2,3,4,5\}$, the function $modNumVec$ returns the unchanged vector if the modification operator is inappropriate like “$-$” and for a sum it does not matter if the zeros are sorted out beforehand. Both of these candidates are in fact partly Sackin indices. The Sackin index sums up the depths of all leaves, here both candidates sum only over five predetermined nodes (which will often, but not always be leaves depending on the node labeling). This is not sensible or transferable to other tree generation functions. Furthermore they do not present novel ideas on tree balance and if we replaced T_3^{cat} with T we would have the normal Sackin index again. Thus both of these individuals will not be used in the final comparison.

ii) Finished with 42 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.314, best individual:

$$\begin{aligned} &mean(getDistSet(getlcaSet(getlcaSet(getAllAncestors(\\ &T, getlca(T_3^{cat}, getRandomNode(\{1\}), getRandomNode(\{1\}))), T), T), T)) \\ =\ &mean(getDistSet(getlcaSet(getlcaSet(getAllAncestors(\\ &T, getlca(T_3^{cat}, 1, 1)), T), T), T)) \\ =\ &mean(getDistSet(getlcaSet(getlcaSet(getAllAncestors(T, 1), T), T), T)) \end{aligned}$$

This function collects all ancestors of the predetermined node 1, then collects the last common ancestors of each pair of this set, repeats this with the new set and finally calculates the average distance of node pairs of this last node set.

i*) Finished with 60 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.295, best individual: $myvar(getDistSet(getLeafNodes(T), T))$
The variance of the distances between all pairs of leaves.

ii*) Finished with 40 evolution steps and 0 restarts during the process (on 47 cores).
Minimal Fitness: 0.309, best individual: equal to best individual of run ii)

7.2.1 Remark

It was only discovered after the first runs of the second set that there was a small mistake in the implementation of I'. n was used as the number of leaves of the complete tree instead of as the number of leaves of the subtree for the factor $\frac{n-1}{n}$, resulting in a factor that was still <1

but larger than intended. For indices like the $\sum I'$ and $meanI'$ this lead to a higher weight of subtrees with odd size. Thus, two further runs were started for each stage and each task with the correct implementation. These runs will be listed under (II) as they use the same parameters but will be marked with a star (*). All of the old runs have to be treated carefully as I'-indices are indeed part of their best individuals. We will here include these individuals in the final comparison but use the correct I' implementation there.

Another error was found regarding the weighted $l1$ distance (the case $z = n$ for $p_n(z)$ was missed out and f_z did not portray the relative but the absolute frequency). Both the first and final comparison have been repeated with the correct implementations for both indices. Only the GP runs cannot be redone. Astoundingly, the comparisons showed that the power of the correct implementation of the weighted $l1$ distance was less powerful for a lot of ζ values than its flawed version. Therefore, it could be interesting to test variations of the weighted $l1$ distance in future projects to develop the indices' full potential. Especially this balance index provides a lot of possibilities for modifications like the weights of the summands or the choice of the norm.

8 Comparison of new and established balance indices

Here we will compare the best established balance indices with the best individuals from genetic programming using the same method as before in Section 3.2. The candidates are individuals with a fitness of less than 0.44 for Task A and 0.285 for Task B. b_3 and b_4 were omitted because they represented a weaker or insensible version of the Sackin index as shown in Section 7.2.

$$
\begin{aligned}
a_1(T) &= SackinI(T) + CollessI(T) \cdot MeanI10I(T) \\
a_2(T) &= SackinI(T) + SackinI(T) \cdot MeanI10I(T) \\
a_3(T) &= MeanI10I(T) \cdot CollessI(T) \\
a_4(T) &= SackinI(T) + 0.14 \cdot n \cdot SumII(T) \\
a_5(T) &= SumII(T) + SackinI(T) - B1I(T)
\end{aligned}
$$

$$
\begin{aligned}
b_1(T) &= MeanI10I(T) + CollessI(T) \\
b_2(T) &= 3.61 \cdot MeanII(T) + exp(CollessI(T) - MeanII(T))
\end{aligned}
$$

For the best established balance indices the Colless, the Sackin, the variance of leaf depths, the B_1, the $\sum I$ and $meanI$ index as well as the weighted l_1 distance were chosen because they proved to be among the best indices for each model for each tested n. Below in the Figures 26, 27, 28 as well as 29 the results of the comparisons are visualized. All figures including the ones for $n = 100$ can be found on the attached CD (`finalcomp_results.zip`).

As expected one of the newly found indices, namely b_2, seems to be affected by overfitting as it performs well for a lower number of leaves, but is extremely inefficient for $n = 200$. It can thus be discarded. The other indices are less affected by differing numbers of leaves.

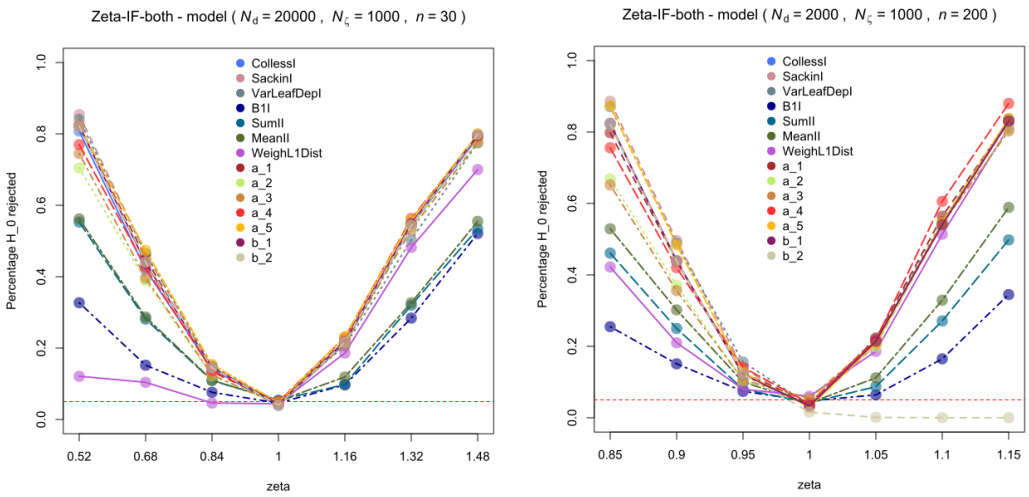


Figure 26: Comparison of the power of balance indices regarding the ζ-IF-both-model.

Regarding Task B, b_1 is expected to perform well on the IF-diff-model. However, nearly all constructed balance indices – regardless of the task they were made for – have the same high power for this alternative model. b_1 could be replaced e.g. by the Sackin index without a

noticeable loss in performance. Nevertheless, the task can certainly not be considered a failure as b_1 is the best balance index for 30 leaves and among the best for higher n.

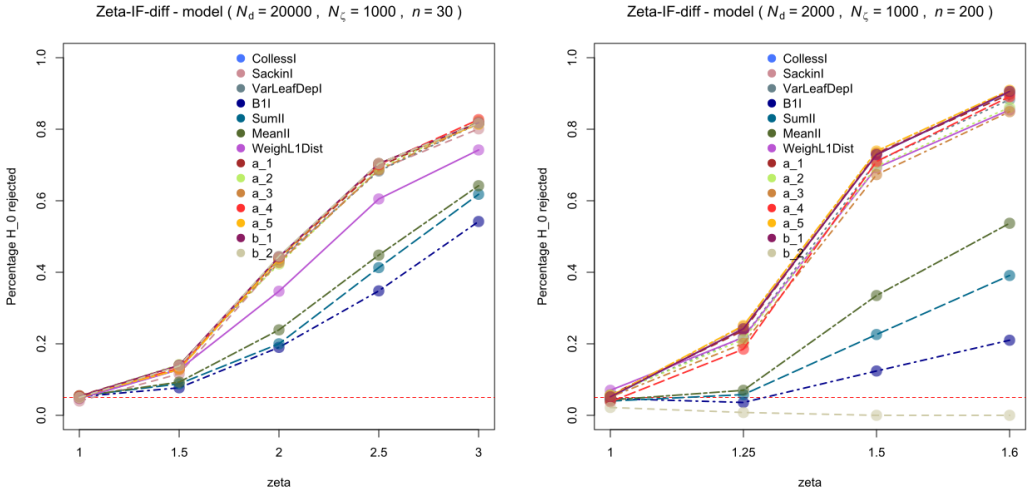


Figure 27: Comparison of the power of balance indices regarding the ζ-IF-diff-model.

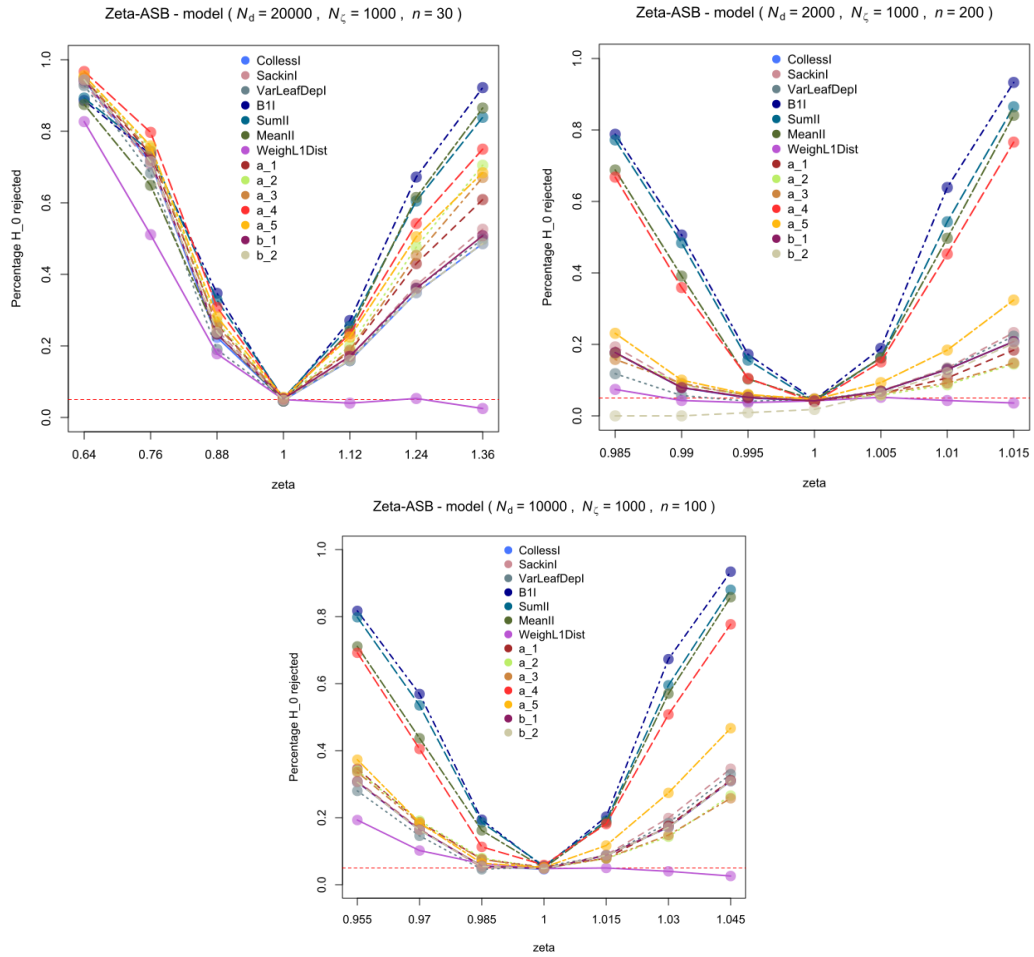


Figure 28: Comparison of the power of balance indices regarding the ζ-ASB-model.

Task A was to find an index that performs well on both the ASB and the other models. Here,

genetic programming seems to have discovered a new "allrounder" with a_4. For the IF-models and the Brownian model a_4 is always among the top. For the ASB-model it is by far the best index apart from the B_1, $\sum I$ and $meanI$ index, which are specialized on this model and perform significantly worse on the other alternative models. This can be most clearly seen in Figure 28 for $n = 100$ and $n = 200$. a_5 seems to show a similar behavior as a_4, but not as distinct. Some other indices perform better for the ASB-model as well, but only for 30 leaves. They should not be preferred over the Colless or Sackin index for $n \geq 100$ according to this data.

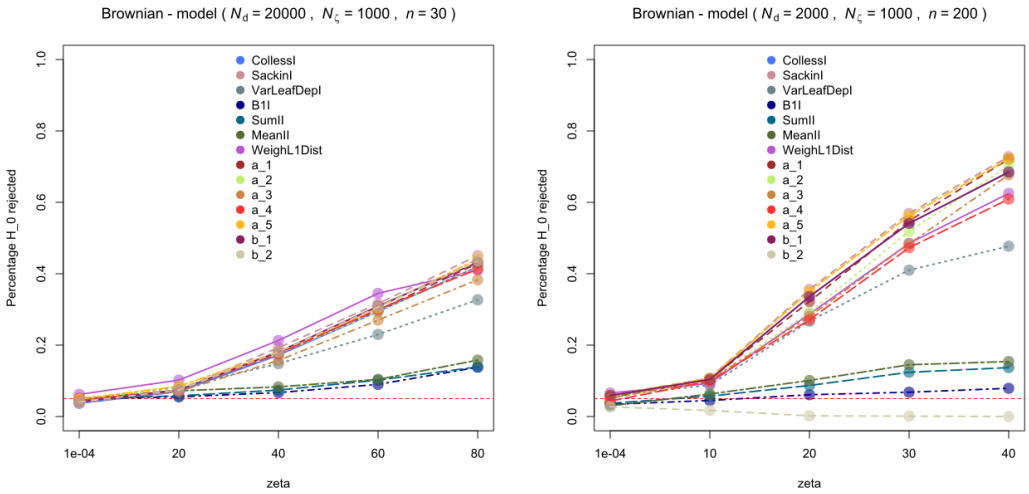


Figure 29: Comparison of the power of balance indices regarding the Brownian model.

With established balance indices as building blocks genetic programming did not succeed in creating indices that could perform significantly better than the established ones for a single alternative model. Possibly there is not much more improvement feasible, but this cannot be confirmed with only these limited approaches. However, genetic programming proved to be viable by finding an overall suitable index that could potentially be used if no prior information is available.

9 Discussion and Results

The results concerning the best choice of balance indices made in this thesis should not necessarily be seen as direct advise. Sometimes the choice of a balance index has to be made on the basis of other required features, e.g. using an I'-based index for comparing trees of different sizes, but even for cases in which these discoveries may apply[8] it should be kept in mind that they were obtained only on a selection of alternative models. It is save to say that the results in this thesis show that there are indeed different asymmetry patterns which can be created with different models and that prior information can lead to a better choice of a balance index. However, for further investigations well thought-out alternative models have to be used that have been shown to produce realistic trees for the given context e.g. phylogenetics. A first step could be to replace the ζ-ASB-model with its continuous age-based version [1, p. 867] to explore if the results are also transferable for higher numbers of leaves or to further introduce global-time-based models [37, p. 8].

Regarding genetic programming the possibilities have not yet been exhausted. Although not every run was successful, the results show that GP can be a viable optimization strategy. Some ideas were collected that could be experimented with in future GP projects on this topic:

- Even though the seed seemed to have more influence on the success of a GP run than the parameters it could be useful to try different parameter settings or evolutionary operators.
- Further improvements could be possible if there was no archive, only an elite that holds the best individuals but is not used as part of the population. Thus, the whole population could be replaced at a restart, which would prevent the archive from steering the population into the same composition again and again. As a result a single run would have the chance for several fresh starts.
- Going along with these two points a modification of the restart condition could prove beneficial as well. Instead of on the standard deviation of the fitness values (this encourages diversity) the condition could be based on the progress (of the minimal fitness values) of the last m generations. Thereby one can restart runs that are stuck for a longer period of time.
- Seeing that Stage 1 with the smallest function set showed the most useful results lead to the idea that it could be useful to restrict the function set further. It could be useful for similar approaches as done here, but it would also be possible to fine-tune already found balance indices like $SackinI(T) + 0.14 \cdot n \cdot SumII(T)$ and optimize only their numerical factors.
- From GP Set (I) to (II) N_d and N_ζ have already been doubled, but of course the higher these values are the more precise the calculated fitness will be. Thereby one could avoid having a lot of actually bad individuals which luckily got a better fitness value than expected.
- The balance indices as building blocks could be unified to only increase or decrease with a higher level of asymmetry. Normalized versions could be replacements as well, even

[8] Analysis of single trees with the Yule model as null hypothesis (as described in Section 3.1).

though they are not always available. Although genetic programming is capable of adjusting different indices in a way that they work well together in an individual on its own, both of these changes could simplify the task as it is not required to unify the balance indices' value ranges at the same time anymore.

- Another completely different GP experiment would be to detach from the idea of using the quantile-based approach for balance indices as test statistics and to aim for a logical expression that tells directly if a tree is more (a-)symmetric than expected under the null hypothesis. The result could therefore be a shortcut avoiding the time-consuming estimation of the balance indices' distributions and quantiles. Balance indices would still serve as building blocks and could form individuals like e.g. $(SymNodesI(T) > TotalCophI(T)) \vee (B2I(T) > SumII(T))$. It could be explored if such logical expressions with high power and limited first error rate exist and if they can compete with the quantile-based method $(q^*_{0.025} > Balance\ Index(T)) \vee (Balance\ Index(T) > q^*_{0.975})$.

Summary of the results

Next to the Yule model as the null hypothesis several different alternative models have been introduced: the ζ-models including the DCO-model (only direct children are affected by factor ζ), the IF-models simulating longterm inherited fertility and the ASB-model simulating age-based fertility as well as the Brownian speciational model which simulates a trait-based fertility. In the first comparison we explored the differences of a large number of established balance indices on this set of alternative models.

The most notable insight was the different performance of groups of balance indices for the ζ-ASB-model on the one hand and for the ζ-DCO-, ζ-IF- and Brownian model on the other hand. The group differences got more distinct for higher numbers of leaves. Depending on prior information on either trait-based fertility and fertility inheritance or age-dependent fertility it could therefore be advisable to select a different balance index in order to achieve the highest power. In case of information about the former it could be sensible to choose the Colless or Sackin index, but in case of the latter the $B1$ index should be preferred (see Figure 30 on the left).

Furthermore, we could show that the cherry and the symmetry nodes index should not be chosen over the $B1$ index, the $\sum I$ or $meanI$ for this set of alternative models atleast. Nevertheless, both indices do not always perform worse than all other indices: For the ASB-model they nearly reached the power of the best indices.

In the second part of the thesis we set up two different genetic programming approaches: the first uses the established balance indices itself as building blocks and the second uses their auxiliary functions. Both were challenged with the two tasks A) to find an overall good balance index that performs well on all alternative models and B) to find an optimal balance index for the IF-diff-model. After trial runs to explore possible parameter settings, a second set of runs was started with modified parameters and further features. However, the success of the single runs was found to be more depending on a lucky seed than on differences in the parameters. Thus, the results of both the trial and the second set of runs were used.

In the first approach for Task A we discovered a potentially useful index that performs well

on all of these models – a feature all of the tested established indices did not have to this extent 30 on the right). There even was a second candidate that performed worse than the first but would still be a better "alrounder" than all of the tested established indices. For Task B no index was found that was significantly better than the already existing ones.

The second approach yielded no astounding results. Nevertheless, the individuals created in the GP process for Task A were all sensible constructions that just could not hold up with the power of already established balance indices.

All in all, we could show that genetic programming can be a viable optimization tool for this topic. A lot of experiences have been made and the resulting ideas on improvements of the GP process are all listed in the Discussion.

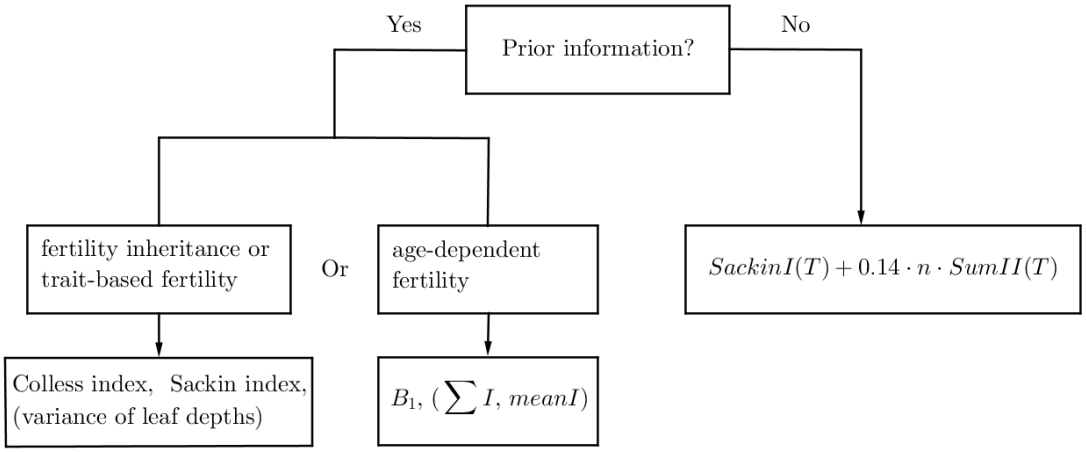
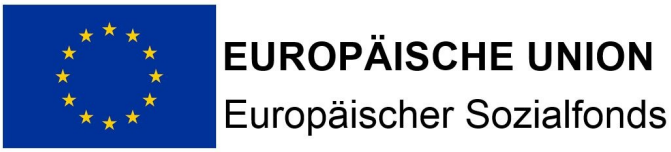


Figure 30: Summary of the most notable discoveries.

Förderhinweis

Die Masterarbeit wurde im Rahmen des Forschungsprojektes ***DIG-IT!*** angefertigt. Die Förderung des Projektes erfolgt aus Mitteln des Europäischen Sozialfonds (ESF) im Rahmen des Qualifikationsprogrammes „Förderung von Nachwuchswissenschaftlern in exzellenten Forschungsverbünden - Exzellenzforschungsprogramm des Landes Mecklenburg-Vorpommern“. (ESF/14-BM-A55-0017/19).



References

[1] Paul-Michael Agapow and Andy Purvis. Power of Eight Tree Shape Statistics to Detect Nonrandom Diversification: A Comparison by Simulation of Two Models of Cladogenesis. *Systematic Biology*, 51(6):866–872, December 2002.

[2] David Aldous. Probability Distributions on Cladograms. In Avner Friedman, Willard Miller, David Aldous, and Robin Pemantle, editors, *Random Discrete Structures*, volume 76, pages 1–18. Springer New York, New York, NY, 1996.

[3] Wolfgang Banzhaf, Peter Nordin, Robert E Keller, and Frank D Francone. *Genetic programming*. Springer, 1998.

[4] Michael G. B. Blum and Olivier François. Which Random Processes Describe the Tree of Life? A Large-Scale Study of Phylogenetic Tree Imbalance. *Systematic Biology*, 55(4):685–691, August 2006.

[5] Michael G. B. Blum, Evelyne Heyer, Olivier François, and Fréderic Austerlitz. Matrilineal Fertility Inheritance Detected in Hunter–Gatherer Populations Using the Imbalance of Gene Genealogies. *PLoS Genetics*, 2(8):e122, 2006.

[6] Michael G.B. Blum and Olivier François. On statistical tests of phylogenetic tree imbalance: The Sackin and other indices revisited. *Mathematical Biosciences*, 195(2):141–153, June 2005.

[7] Jürgen Bortz and Christof Schuster. *Statistik für Human- und Sozialwissenschaftler*. Springer-Lehrbuch. Springer, Berlin Heidelberg, 7., vollständig überarbeitete und erweiterte auflage edition, 2010. OCLC: 845714518.

[8] Bo Chen, Daniel Ford, and Matthias Winkel. A new family of Markov branching trees: the alpha-gamma model. *arXiv:0807.0554 [math]*, July 2008. arXiv: 0807.0554.

[9] C. Colijn and G. Plazzotta. A Metric on Phylogenetic Tree Shapes. *Systematic Biology*, 67(1):113–126, January 2018.

[10] Donald H. Colless. Review of "Phylogenetics: The Theory and Practice of Phylogenetic Systematics" by E. O. Wiley. *Systematic Zoology*, 31(1):100–104, March 1982.

[11] Tomás M. Coronado, Arnau Mir, Francesc Rosselló, and Gabriel Valiente. A balance index for phylogenetic trees based on rooted quartets. *Journal of Mathematical Biology*, 79(3):1105–1148, August 2019.

[12] Microsoft Corporation and Steve Weston. *doParallel: Foreach Parallel Adaptor for the 'parallel' Package*, 2019. R package version 1.0.15.

[13] J. L. Doob. Topics in the theory of Markoff chains. *Transactions of the American Mathematical Society*, 52(1):37–37, January 1942.

[14] Matt Dowle and Arun Srinivasan. *data.table: Extension of 'data.frame'*, 2019. R package version 1.12.8.

[15] Oliver Flasch. A friendly introduction to RGP. 2014.

[16] Oliver Flasch, Olaf Mersmann, Thomas Bartz-Beielstein, Joerg Stork, and Martin Zaefferer. *rgp: R genetic programming framework*. 2014.

[17] Daniel J. Ford. Probabilities on cladograms: introduction to the alpha model. *arXiv:math/0511246*, November 2005. arXiv: math/0511246.

[18] George W. Furnas. The generation of random, binary unordered trees. *Journal of Classification*, 1(1):187–233, December 1984.

[19] Giuseppe Fusco and Quentin C.B. Cronk. A new method for evaluating the shape of large phylogenies. *Journal of Theoretical Biology*, 175(2):235–243, July 1995.

[20] E. F. Harding. The probabilities of rooted tree-shapes generated by random bifurcation. *Advances in Applied Probability*, 3(1):44–77, 1971.

[21] Maryam Hayati, Bita Shadgar, and Leonid Chindelevitch. A new resolution function to evaluate tree shape statistics. *PloS one*, 14(11):e0224197, 2019.

[22] Stephen B. Heard. PATTERNS IN TREE BALANCE AMONG CLADISTIC, PHENETIC, AND RANDOMLY GENERATED PHYLOGENETIC TREES. *Evolution*, 46(6):1818–1826, December 1992.

[23] Stephen B. Heard and Graham H. Cox. The Shapes of Phylogenetic Trees of Clades, Faunas, and Local Assemblages: Exploring Spatial Pattern in Differential Diversification. *The American Naturalist*, 169(5):E107–E118, May 2007.

[24] John P. Huelsenbeck and Mark Kirkpatrick. DO PHYLOGENETIC METHODS PRODUCE TREES WITH BIASED SHAPES? *Evolution*, 50(4):1418–1424, August 1996.

[25] David G. Kendall. On the Generalized "Birth-and-Death" Process. *The Annals of Mathematical Statistics*, 19(1):1–15, March 1948.

[26] Mark Kirkpatrick and Montgomery Slatkin. SEARCHING FOR EVOLUTIONARY PATTERNS IN THE SHAPE OF A PHYLOGENETIC TREE. *Evolution*, 47(4):1171–1181, August 1993.

[27] John R Koza and John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[28] JohnR. Koza. Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, 4(2), June 1994.

[29] Udo Kuckartz, Stefan Rädiker, Thomas Ebert, and Julia Schehl. *Statistik: eine verständliche Einführung*. Lehrbuch. Springer VS, Wiesbaden, 2., überarbeitete auflage edition, 2013. OCLC: 861967528.

[30] Leonardo P. Maia, Alexandre Colato, and José F. Fontanari. Effect of selection on the topology of genealogical trees. *Journal of Theoretical Biology*, 226(3):315–320, February 2004.

[31] Andy McKenzie and Mike Steel. Distributions of cherries for two models of trees. *Mathematical Biosciences*, 164(1):81–92, March 2000.

[32] Olaf Mersmann. *microbenchmark: Accurate Timing Functions*, 2019. R package version 1.4-7.

[33] Olaf Mersmann. *emoa: Evolutionary Multiobjective Optimization Algorithms*, 2020. R package version 0.5-0.1.

[34] Microsoft and Steve Weston. *foreach: Provides Foreach Looping Construct*, 2020. R package version 1.5.0.

[35] Arnau Mir, Francesc Rosselló, and Lucı´a Rotger. A new balance index for phylogenetic trees. *Mathematical Biosciences*, 241(1):125–136, January 2013.

[36] David J Montana. Strongly typed genetic programming. *Evolutionary computation*, 3(2):199–230, 1995. Publisher: MIT Press.

[37] A. Mooers, Luke Harmon, Michael Blum, Dennis Wong, and Stephen Heard. Some models of phylogenetic tree shape. pages –. 2007.

[38] Arne O. Mooers and Stephen B. Heard. Inferring Evolutionary Process from Phylogenetic Tree Shape. *The Quarterly Review of Biology*, 72(1):31–54, March 1997.

[39] Brian R. Moore, Kai M. A. Chan, and Michael J. Donoghue. Detecting Diversification Rate Variation in Supertrees. In Andreas Dress and Olaf R. P. Bininda-Emonds, editors, *Phylogenetic Supertrees*, volume 4, pages 487–533. Springer Netherlands, Dordrecht, 2004. Series Title: Computational Biology.

[40] E. Paradis and K. Schliep. ape 5.0: an environment for modern phylogenetics and evolutionary analyses in R. *Bioinformatics*, 2018.

[41] Emmanuel Paradis. *Analysis of phylogenetics and evolution with R*. Use R! Springer, New York, 2nd ed edition, 2012. OCLC: ocn774538542.

[42] R Development Core Team. R: A Language and Environment for Statistical Computing, 2008.

[43] Liam J. Revell. phytools: An R package for phylogenetic comparative biology (and other things). *Methods in Ecology and Evolution*, 3:217–223, 2012.

[44] F. James Rohlf, W. S. Chang, R. R. Sokal, and Junhyong Kim. ACCURACY OF ESTIMATED PHYLOGENIES: EFFECTS OF TREE TOPOLOGY AND EVOLUTIONARY MODEL. *Evolution*, 44(6):1671–1684, September 1990.

[45] M. J. Sackin. "Good" and "Bad" Phenograms. *Systematic Biology*, 21(2):225–226, July 1972.

[46] Charles Semple and M. A. Steel. *Phylogenetics*. Number 24 in Oxford lecture series in mathematics and its applications. Oxford University Press, Oxford ; New York, 2003.

[47] Kwang-Tsao Shao and Robert R. Sokal. Tree Balance. *Systematic Zoology*, 39(3):266, September 1990.

[48] Ed Stam. DOES IMBALANCE IN PHYLOGENIES REFLECT ONLY BIAS? *Evolution*, 56(6):1292–1295, June 2002.

[49] M. A. Steel. *Phylogeny: discrete and random processes in evolution*. Number 89 in CBMS-NSF regional conference series in applied mathematics. Society for Industrial and Applied Mathematics, Philadelphia, 2016.

[50] G. Udny Yule. II.—A mathematical theory of evolution, based on the conclusions of Dr. J. C. Willis, F. R. S. *Philosophical Transactions of the Royal Society of London. Series B, Containing Papers of a Biological Character*, 213(402-410):21–87, February 1924.