Convex characters, algorithms and matchings

Steven Kelk, Ruben Meuwese, Stephan Wagner

Abstract

Phylogenetic trees are used to model evolution: leaves are labelled to represent contemporary species ("taxa") and interior vertices represent extinct ancestors. Informally, convex characters are measurements on the contemporary species in which the subset of species (both contemporary and extinct) that share a given state, form a connected subtree. Kelk and Stamoulis (*Advances in Applied Mathematics*, 84 (2017), pp. 34-46) showed how to efficiently count, list and sample certain restricted subfamilies of convex characters, and algorithmic applications were given. We continue this work in a number of directions. First, we show how combining the enumeration of convex characters with existing parameterised algorithms can be used to speed up exponential-time algorithms for the *maximum agreement forest problem* in phylogenetics. Second, we re-visit the quantity $g_2(T)$, defined as the number of convex characters on T in which each state appears on at least 2 taxa. We use this to give an algorithm with running time $O(\phi^n \cdot \text{poly}(n))$, where $\phi \approx 1.6181$ is the golden ratio and n is the number of taxa in the input trees, for computation of *maximum parsimony distance on two state characters*. By further restricting the characters counted by $g_2(T)$ we open an interesting bridge to the literature on enumeration of matchings. By crossing this bridge we improve the running time of the aforementioned parsimony distance algorithm to $O(1.5895^n \cdot \text{poly}(n))$, and obtain a number of new results in themselves relevant to enumeration of matchings on at-most binary trees.

Keywords. phylogenetics, matchings, enumeration, combinatorics, exponential-time algorithms, agreement forests

AMS subject classification. 68R05, 92B10, 68W40, 05C70, 05C30

1 Introduction

In bioinformatics evolution is often represented by a binary tree T whose leaves are bijectively labelled by a set X of labels ("taxa") which represent contemporary species. Interior nodes represent hypothetical ancestors of the species in X; the tree models the branching process which, via evolutionary phenomena such as mutation and speciation, caused X to evolve. Such a tree is known as a *phylogenetic tree*.

There is an extensive mathematical and algorithmic literature on constructing and comparing phylogenetic trees (see e.g. [25, 4]). Many corresponding optimisation problems are NP-hard. One algorithmic approach to tackling NP-hard problems is to develop exact exponential-time algorithms [13]: usually, algorithms that run in time $O(c^n \cdot \text{poly}(n))$ where c is a (small) constant and n is the size of the input. The literature on exponential-time algorithms focusses, logically, on optimising the constant c. The interest is not purely theoretical, since especially in the early phases of studying an optimisation problem an exponential-time algorithm might be the best practical method available, and then optimising c can make a crucial difference in practice. Exponential-time algorithms can also be useful for solving the smaller, reduced instances yielded by pre-processing strategies [14].

In this article we give improved exponential-time algorithms for three optimisation problems occurring in phylogenetics. In all three cases we make heavy use of enumeration; in particular, enumeration of so-called *convex characters*. For a phylogenetic tree T on X, a convex character is a partition of X such that the minimal spanning trees induced by the blocks of X are disjoint in T (see e.g. Section 4.1 of [25]). In [20] it was proven that a phylogenetic tree on n taxa has $\Theta(\phi^{2n})$ convex characters, and $\Theta(\phi^n)$ convex characters in which each block of the character contains at least two taxa, where $\phi \approx 1.681$ is the golden ratio and $\phi^2 \approx 2.6181$. These characters can also be efficiently counted and listed, which formed the basis for a $O(\phi^{2n} \cdot \text{poly}(n))$ time algorithm for the *unrooted maximum agreement forest* problem and a $O(\phi^n \cdot \text{poly}(n))$ time algorithm for the *maximum parsimony distance* problem; we shall formally define these problems later. The algorithms are based on the insight that the optimal solutions to these problems are either convex characters themselves, or "project down" onto convex characters - see [19] for related discussions. In the algorithmic literature it is common to use O^* notation, which suppresses polynomial terms, to denote the running times of exponential-time algorithms. This is due to the dominance of the exponential terms in the running time. Here we do the same: the two aforementioned algorithms thus have running times $O^*(\phi^{2n})$ and $O^*(\phi^n)$, respectively. We will only use this notation for algorithmic running times, not for enumerative results.

In this article we extend the results from [20] in several directions. We improve the $O^*(2.6181^n)$ algorithm for the unrooted maximum agreement forest problem to $O^*(2.2973^n)$, and improve the $O^*(2.42^n)$ algorithm for the *rooted* variant of this problem to $O^*(2.0649^n)$. In both cases the high-level idea is to leverage an existing branching-based algorithm that can answer the question, "Is the optimum at most k?" in time exponential in k (as opposed to n). This so-called *fixed parameter tractable* algorithm - see [7] for an introduction to such algorithms - is run incrementally up to a carefully-chosen threshold k'. If the threshold is exceeded, i.e. the optimum is "large", the optimum solution can be found by searching through a comparatively small subset of convex characters.

Our third contribution concerns the NP-hard problem *maximum parsimony distance on two-state characters* [18, 21]. This problem has a trivial $O^*(2^n)$ time algorithm, but unlike the more general variant of the problem there is no obvious link with convex characters. Here we establish such a link, using it to obtain an enumeration-based algorithm with running time $O^*(\phi^n)$, and then improve this to $O^*(1.5895^n)$, also using enumeration. The strengthening to $O^*(1.5895^n)$ is potentially of wider interest, because it operates by enumerating *matchings* (pairwise disjoint sets of edges) in transformed variants of the input trees. There is an extensive literature on the enumeration of matchings in trees[1]; see [27] for a survey. Due to the combinatorial structure of the maximum parsimony distance problem, we in fact only need to consider certain carefully constrained subsets of matchings, and it is these that we bound/enumerate using a technique from [23, 24]. Interestingly, this translates to a new result on normal (i.e. unconstrained) matchings; in particular, we show that on (not necessarily phylogenetic) trees with in total n nodes, maximum degree 3 and no degree-2 nodes adjacent to each other, there can be at most $O(1.5895^n)$ matchings. This supplements existing upper bounds of $O(\phi^n)$, valid for arbitrary trees, and $O(1.5538^n)$ for trees of maximum degree 3 where all interior nodes have degree 3 (see [2, Theorem 1 and Remark 5]; the precise constant is $\sqrt{1+\sqrt{2}}$).

In Section 2 we give preliminaries. Section 3 describes our improvements for the maximum agreement forest problem(s) while Sections 4 and 5 describe our improvement for the maximum parsimony distance problem on two state characters; the first of these two sections builds the correspondence with convex characters, and the second section builds the correspondence with matchings, the number of which we then bound. The proof of our new bound is computer-assisted in view of the heavy calculations it involves. In an extended technical discussion (Section 6) we provide a number of auxiliary insights and (lower) bounds emerging from Sections 4 and 5. Finally, in Section 7 we place our findings in a broader context and propose several directions for future work.

2 Preliminaries

For general background on mathematical phylogenetics we refer to [25, 9]. An *unrooted binary phylogenetic X-tree* is an undirected tree $T = (V(T), E(T))$ where every internal vertex has degree 3 and whose leaves are bijectively labelled by a set X, where X is often called the set of *taxa* (representing the contemporary species). We use n to denote $|X|$ and often simply write phylogenetic tree when it is clear from the context that we are talking about an unrooted binary phylogenetic X-tree. Two phylogenetic trees T, T' on X are considered equal if there is an isomorphism between them that is identity on X.

A *character* f on X is a surjective function $f : X \to \mathcal{C}$ for some set $\mathcal{C}$ of *states* (where a state represents some characteristic of the species e.g. number of legs). We say that f is an r-state character if $|\mathcal{C}| = r$. Each character naturally induces a partition of X and here we regard two characters as being equivalent if they both induce the same partition of X. Hence, the states can simply be regarded as the blocks of a partition. An *extension* of a character f to $V(T)$ is a function $h : V(T) \to \mathcal{C}$ such that $h(x) = f(x)$ for all x in X. For such an extension h of f, we denote by $l_h(T)$ the number of edges $e = \{u, v\}$ such that $h(u) \neq h(v)$. These edges are often called *mutation edges*, or simply *mutations*, reflecting the biological origin of the model. The *parsimony score* of a character f on T, denoted by $l_f(T)$, is obtained by minimising $l_h(T)$ over all possible extensions h of f. We say that a character $f : X \to \mathcal{C}$ is *convex on* T if $l_f(T) = |\mathcal{C}| - 1$. Equivalently: a character $f : X \to \mathcal{C}$ is convex on T if there exists an extension h of f such that, for each state $c \in \mathcal{C}$, the vertices of T that are allocated state c (by h) form a connected subtree of T. We call such an extension h a *convex extension* of f. The convexity of a character can be tested in polynomial [11, 15] (in fact, linear [3]) time. We note that a third, equivalent definition of convexity which does not use the machinery of parsimony scores or extensions is as follows: a character f is convex on T if the minimal spanning trees induced by the blocks of f are vertex-disjoint. In [20] the quantity $g_k(T)$ was introduced, $k \geq 1$, defined as the number of convex characters of T in which each state contains at least k taxa (see Figure 1 for an illustration). The quantity $g_1(T)$ is therefore simply the total number of convex characters of T. Given a tree T on n taxa the quantity $g_1(T)$ grows as $\Theta(\phi^{2n})$, independently of T, where $\phi \approx 1.6181...$ is the golden ratio. Because the bases of the exponential terms we describe in this article are often close together, and the use of these terms in bounding the worst-case running time of algorithms, we give 4 decimal places and always round up. This will provide a better view of our improvements. In

[1] The number of matchings in a tree is often called the *Hosoya Index*.

Section 4 of this article, $g_2(T)$ plays a prominent role. It grows as $\Theta(\phi^n)$, also independently of T [20].

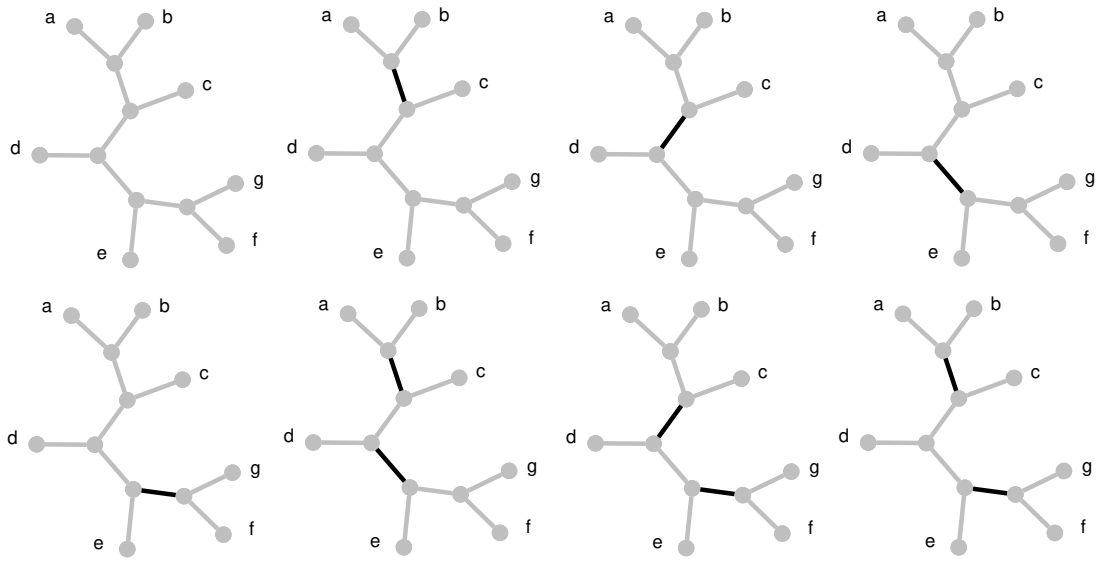


Figure 1: A phylogenetic tree T where $X = \{a, b, c, d, e, f, g\}$. For the given tree there are in total 233 convex characters [20]. There are 8 convex characters in which each state appears on at least 2 taxa, so $g_2(T) = 8$. One of these characters has exactly 1 state ($abcdefg$), 4 of these characters have 2 states ($ab|cdefg$, $abc|defg$, $abcd|efg$ and $abcde|fg$), and 3 of these characters have 3 states ($ab|cd|efg$, $abc|de|fg$ and $ab|cde|fg$). Here we use | to denote the states (i.e. blocks) of the character (i.e. partition). The 8 figures verify the convexity of these characters: the minimal spanning trees induced by the states of the character, shown in grey, are vertex-disjoint.

3 Improved exponential-time algorithms for agreement forests

Let T be a tree on X. For $X' \subseteq X$ we write $T[X']$ to denote the miminal subtree of T spanning X', and write $T|X'$ to denote the phylogenetic tree obtained from $T[X']$ by suppressing nodes of degree 2.

Let T and T' be two phylogenetic trees on X. Let $F = \{X_1, X_2, \ldots, X_k\}$ be a partition of X, where each block X_i with $i \in \{1, 2, \ldots, k\}$ is referred to as a *component* of F. We say that F is an *agreement forest* for T and T' if the following conditions hold.

(1) For each $i \in \{1, 2, \ldots, k\}$, we have $T|X_i = T'|X_i$.

(2) For each pair $i, j \in \{1, 2, \ldots, k\}$ with $i \neq j$, we have that $T[X_i]$ and $T[X_j]$ are vertex-disjoint in T, and $T'[X_i]$ and $T'[X_j]$ are vertex-disjoint in T'.

Let $F = \{X_1, X_2, \ldots, X_k\}$ be an agreement forest for T and T'. The *size* of F is simply its number of components, k. Moreover, an agreement forest with the minimum number of components (over all agreement forests for T and T') is called a *maximum agreement forest (MAF)* for T and T'. The number of components of a maximum agreement forest for T and T' is denoted by $d_{\text{uMAF}}(T, T')$. The Unrooted Maximum Agreement Forest (uMAF) problem is to compute $d_{\text{uMAF}}(T, T')$; it is NP-hard [1, 16].[2]

There is also a *rooted* version of the maximum agreement forest problem but we defer a discussion of this until later.

3.1 An $O^*(2.2973^n)$ algorithm for finding an uMAF

By part (2) of the definition of agreement forest, an agreement forest for T and T' is a convex character on both T and T', but the converse does not necessarily hold. A tree has $\Theta(\phi^{2n})$ convex characters and these can be listed

[2]Allen and Steel [1] also showed that another well-known phylogenetic dissimilarity measure, called Tree Bisection and Reconnection distance ($d_{TBR}(T, T')$), is related to the *uMAF* distance as follows: $d_{\text{uMAF}}(T, T') - 1 = d_{TBR}(T, T')$. For this reason some articles in the literature define the size of an agreement forest as its number of components *minus one*. However, given the emphasis on enumerative combinatorics in this article it is most natural to simply use the number of components. Our results, which are inherently asymptotic, hold under either definition of size.

efficiently [20]. Combined with the fact that it can easily be tested in polynomial time whether a convex character is an agreement forest, [20] gave an elementary $\Theta^*(\phi^{2n}) = \Theta^*(2.6181^n)$ algorithm for computing $d_{\text{uMAF}}(T, T')$, simply by looping through all characters that are convex on T and noting the smallest agreement forest found.

To improve upon this, we leverage an existing algorithm by Chen et al. [5] that can answer the question "Is $d_{\text{uMAF}}(T, T') \leq k$?" in time $O^*(3^k)$ – and, when the answer is YES, can construct an agreement forest of size at most k. Starting at 1 and incrementing k until the answer is YES gives an algorithm for computing $d_{\text{uMAF}}(T, T')$. Such *fixed parameter tractable* (FPT) algorithms have running times of the form $O(f(k) \cdot \text{poly}(n))$ where f is a computable function that does not depend on n. This decoupling of $f(k)$ and n is a central design principle of such algorithms: informally, they will run quickly when k is constant irrespective of how large n is [7]. This is why it is reasonable to summarize the running times of such algorithms as $O^*(f(k))$.

Our idea is to use the algorithm from Chen et al. [5] up to a certain tipping point $k' = cn$, for a constant c still to be determined, exploiting the fact that 3^k is far smaller than 2.6181^n up to quite large values of k. If the algorithm from Chen et al. [5] finds an agreement forest, we are done. Otherwise (i.e. the answer is still NO) we switch to enumeration of convex characters. Specifically, we list all characters that are convex on T and which have more than k' states. For each such character we check whether it is an agreement forest of T and T' and note the smallest agreement forest found this way.

To put this into practice, we need to bound the number of convex characters with 'many' states, ensure that they can be listed/enumerated efficiently, and determine the optimal tipping point k'.

Let us define the *size* of a convex character as its number of states/blocks. We will show that there is a dynamic programming (DP) algorithm that can enumerate every convex character with exactly k states for a tree T with a running time that will be bounded by the number of convex characters with exactly k states.

Steel [26] gives us an exact expression for how many convex character there are with exactly k states, denoted by $g(T, k)$ which equals $\binom{2n-k-1}{k-1}$. Note that this number is independent of the topology of T. The whole point of designing this DP is not to calculate how many convex characters there are but where they come from, so they can be efficiently constructed. Kelk and Stamoulis [20] gave a DP algorithm that enumerates convex characters where each state has at least a certain number of taxa. We draw inspiration from this algorithm and construct a similar DP algorithm. The DP requires the tree to be rooted, in order to establish an unambiguous parent-child relation between nodes. This can be easily achieved by subdividing an edge of the tree, since the presence of the subdividing vertex does not change the space of convex characters.

For a node u of a tree T, we let T_u be the subtree rooted at u. A convex character for T_u with k states can be created in two ways. It can be the disjoint union of a convex character from the left child node of u, and the right, such that the combination has k states. Or it can be a combination of a convex character from the left child node and the right whereby a state from the left character is merged with a state from the right i.e. the spanning tree for this merged state traverses the root. Because of this we need to store more than simply all the convex characters of subtrees in our DP algorithm; we also have to note which states of a convex character can 'reach the root'; we will formalise this in due course.

To keep track of these distinctions, for each node u of tree T we enumerate ordered pairs (f, A) where f is a convex character of T_u such that there is an optimal extension of f to T_u whereby the root of T_u is assigned state A. Essentially, this means that, if desired, state A of f will be available for merging with another state in the tree beyond T_u.

We write $(f, \emptyset)$ to denote the situation when f is a convex character of T_u but we simply do not care about which state is assigned to the root of T_u. We will not be merging any state of this character with a state from beyond T_u, which is why we do not care.

For the DP algorithm we store two types of values. First, the number of convex characters with k states, denoted by $g(T_u, k)$, and second the number of pairs (f, A), $A \neq \emptyset$, such that f has k states one of them is A who is assigned to the root of T_u, denoted by $h(T_u, k)$. Note that $g(T_u, k)$ simply counts all $(f, \emptyset)$ for T_u where f has k states.

Let us consider a simple example. Suppose u has two children, one a taxa c, the other a parent of two taxa labelled a and b. T_u has five convex characters:

$$\text{t} = \{\{a, b, c\}\}$$

$$\text{x} = \{\{a, b\}, \{c\}\}$$

$$\text{y} = \{\{a, c\}, \{b\}\}$$

$$\text{z} = \{\{b, c\}, \{a\}\}$$

$$\text{s} = \{\{a\}, \{b\}, \{c\}\}.$$

In this case $g(T_u, 1) = 1$, $g(T_u, 2) = 3$ and $g(T_u, 3) = 1$. We have $h(T_u, 1) = 1$ because this only counts the tuple $(t, \{a, b, c\})$. Further more we have $h(T_u, 2) = 4$ because this counts the tuples $(x, \{a, b\})$, $(x, \{c\})$, $(y, \{a, c\})$ and $(z, \{b, c\})$. $h(T_u, 2)$ does not count $(y, \{b\})$ because state $\{a, c\}$ blocks b from connecting to u. The same is true for $(z, \{a\})$. Finally $h(T_u, 3) = 3$ counting $(s, \{a\})$, $(s, \{b\})$, $(s, \{c\})$.

Let $L(T_u)$ be the set of leaves of the subtree T_u and let $m_u = |L(T_u)|$. This means that in the equations below k (and thus i and j) have a maximum value m_u.

We compute the values $g(T_u, k)$ and $h(T_u, k)$ for each value k using two recursions and move bottom-up from the leaves. If we assume for a node u that the left node l and right node r are calculated correctly the values of u are calculated as follows:

$$h(T_u, k) = \sum_{\substack{i,j \\ i+j=k}} h(T_l, i)g(T_r, j) + g(T_l, i)h(T_r, j) \quad + \sum_{\substack{i,j \\ i+j=k+1}} h(T_l, i)h(T_r, j) \tag{1}$$

Note that $i, j \geq 1$ because every subtree T_l and T_r is nonempty and therefore must have at least 1 state. Each pair (f, C) in T_u that is counted by the $h(T_u, .)$ quantities has 3 possible origins, demonstrated in Figure 2.

1. A pair (f_l, A) from the left child node, $A \neq \emptyset$, is combined with a pair $(f_r, \emptyset)$ from the right child node to create the pair $(f_l \cup f_r, A)$. The character $f_l \cup f_r$ has $|f_l| + |f_r|$ states.

2. A pair $(f_l, \emptyset)$ from the left child node is combined with a pair (f_r, B) from the right child node, $B \neq \emptyset$, to create the pair $(f_l \cup f_r, B)$. The character $f_l \cup f_r$ has $|f_l| + |f_r|$ states.

3. A pair (f_l, A) from the left child, $A \neq \emptyset$, is combined with a pair (f_r, B) from the right child node, $B \neq \emptyset$, to create the pair $(f_l \cup f_r, A \cup B)$. The character $f_l \cup f_r$ has $|f_l| + |f_r| - 1$ states.

The first two origins are counted in the first sum of Equation 1. The third origin is counted in the second sum of that equation.

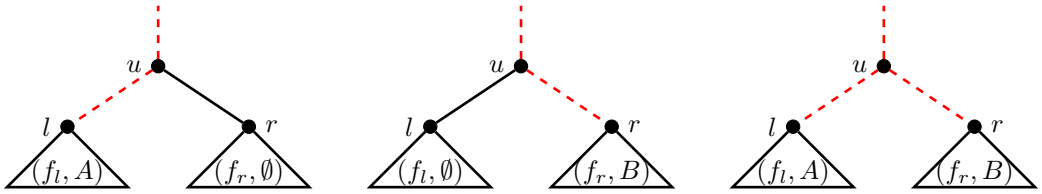


Figure 2: All 3 possible origins for a pair (f, C), as counted by $h(T_u, .)$.

The equation for $g(T_u, k)$ is as follows. The logic behind this equation is similar to that behind $h(T_u, k)$; see Figure 3.

$$g(T_u, k) = \sum_{\substack{i,j \\ i+j=k}} g(T_l, i)g(T_r, j) + \sum_{\substack{i,j \\ i+j=k+1}} h(T_l, i)h(T_r, j) \tag{2}$$

The only thing that remains is a way to enumerate these convex characters. In other words, a way to backtrack in the tree T. The recursion will be this guide. One way is looking at Equation 1 as if it were a list that starts with $h(T_l, 1)g(T_r, k-1)$, followed by $h(T_l, 2)g(T_r, k-2)$, passing through $g(T_l, 1)h(T_r, k-1)$ and ending at $h(T_l, k)h(T_r, 1)$. Rewriting this will show it in a more formal way.

$$h(T_u, k) = \sum_i h(T_l, i)g(T_r, k-i) + \sum_i g(T_l, i)h(T_r, k-i) + \sum_i h(T_l, i)h(T_r, k+1-i) \tag{3}$$

We do the same thing for $g(T_u, k)$.

$$g(T_u, k) = \sum_i g(T_l, i)g(T_r, k-i) + \sum_i h(T_l, i)h(T_r, k+1-i) \tag{4}$$

Thus finally we have a way to enumerate[3] every convex character with exactly k states.

[3]We note that, if desired, the term $g(T_u, k)$ can be replaced by Steel's closed expression $\binom{2n-k-1}{k-1}$. However, when enumerating and constructing such characters the explicit recurrence for $g(T_u, k)$ is required.

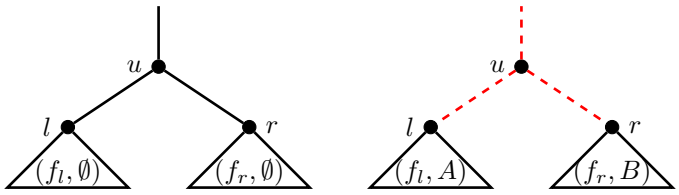


Figure 3: The two possible origins for a convex character with k states.

Theorem 1. *For a pair of binary trees T and T' we can compute and enumerate each convex character with at least k states for T, and determine whether it is a uMAF, in $O^*(f(n,k))$ time with $f(n,k) = \sum_{r=k}^{n} \binom{2n-r-1}{r-1}$.*

Proof. For each value of r between k and n we run the DP algorithm described above. The DP algorithm can easily be leveraged to impose a canonical ordering on the convex characters being counted. This is because the h and g values are cleanly defined as a summation of terms. For each node we calculate at most n different $h(T_u, r)$ and n different $g(T_u, r)$ values, each taking at most linear time to calculate. This can be done in polynomial time, as expected. For each convex character we backtrack through the dynamic programming tree in linear time. We then test if the character is also convex on T' which can be done in linear time according to Bachoore and Bodlaender [3]. If the character is convex on both trees, testing if it is an agreement forest can also be done in low-order polynomial time. Because we have to do this for each convex character with exactly r states this will take $O^*(\binom{2n-r-1}{r-1})$ time. The final running time arises because we loop through all values of r (i.e. number of states) between k and n. □

To get the overall runtime $O^*(2.2973^n)$ we need to solve the equation $3^k = f(n,k)$. This equation arises because we wish to balance the time taken by the FPT algorithm from Chen et al. [5], which is $O^*(3^k)$, with the time required to enumerate convex characters after the FPT algorithm has stopped.

Let us analyse the sum $f(n,k)$: the quotient of two consecutive terms is

$$\frac{\binom{2n-r-2}{r}}{\binom{2n-r-1}{r-1}} = \frac{(2n-2r)(2n-2r-1)}{r(2n-r-1)},$$

which is greater than 1 for $r < r_0 = \frac{10n-3-\sqrt{20n^2-20n+9}}{10} = (1-\frac{1}{\sqrt{5}})n + O(1)$, and less than 1 for $r > r_0$. In other words, the terms increase up to r_0 (provided that $k < r_0$), and decrease afterwards. We can combine this with the known bound $\binom{n}{pn} \leq e^{nH(p)}$, where $H(p)$ is the entropy function $H(p) = -p\ln p - (1-p)\ln(1-p)$ (see for instance Example 11.1.3 in [6]), to obtain the following bound on $f(n,k)$:

Theorem 2. *For positive integers n and k, we have, with $c = \frac{k}{n}$,*

$$f(n,k) \leq \begin{cases} \exp\Big((2-c)H\big(\frac{c}{2-c}\big)n + O(\ln n)\Big) & c \geq c_0 = 1 - \frac{1}{\sqrt{5}}, \\ \exp\Big((2-c_0)H\big(\frac{c_0}{2-c_0}\big)n + O(\ln n)\Big) & c \leq c_0 = 1 - \frac{1}{\sqrt{5}}. \end{cases}$$

Proof. By the aforementioned bound on binomial coefficients, we have

$$\binom{2n-r-1}{r-1} = \frac{r}{2n-r}\binom{2n-r}{r} \leq \frac{r}{2n-r}\exp\Big((2n-r)H\Big(\frac{r}{2n-r}\Big)\Big) = \exp\Big(\Big(2-\frac{r}{n}\Big)H\Big(\frac{r/n}{2-r/n}\Big)n + O(\ln n)\Big)$$

for all positive integers $r < n$. Next note that

$$\frac{d}{dx}(2-x)H\Big(\frac{x}{2-x}\Big) = 2\ln(2-2x) - \ln(2-x) - \ln x,$$

which is positive for $0 < x < c_0$, and negative for $c_0 < x < 1$. Now we distinguish the two cases $c = \frac{k}{n} \geq c_0$ and $c = \frac{k}{n} \leq c_0$:

- Suppose first that $c \geq c_0$. We have $\frac{r}{n} \geq c \geq c_0$ for all $r \geq k$, and since the function $x \mapsto (2-x)H(\frac{x}{2-x})$ is decreasing on the interval $(c_0, 1)$, this implies that

$$\Big(2-\frac{r}{n}\Big)H\Big(\frac{r/n}{2-r/n}\Big) \leq (2-c)H\Big(\frac{c}{2-c}\Big)$$

for all $r \geq k$. Thus

$$\binom{2n-r-1}{r-1} \leq \exp\Big((2-c)H\Big(\frac{c}{2-c}\Big)n + O(\ln n)\Big)$$

for every $r \geq k$. There are at most n such terms in the sum for $f(n,k)$, and the resulting factor n can be absorbed into the error term to yield the desired upper bound in this case.

- Now suppose that $c \leq c_0$. Since the function $x \mapsto (2-x)H(\frac{x}{2-x})$ attains its maximum when $x = c_0$, we obtain

$$\binom{2n-r-1}{r-1} \leq \exp\Big((2-c_0)H\Big(\frac{c_0}{2-c_0}\Big)n + O(\ln n)\Big)$$

in this case by an analogous argument, and the desired upper bound follows again.

□

We compare this with the running time of the FPT algorithm, i.e. 3^k. Since subexponential factors do not matter to us, this amounts to the equation

$$e^{(2-c)nH(\frac{c}{2-c})} = 3^{cn},$$

as one easily finds that $e^{(2-c_0)H(\frac{c_0}{2-c_0})} > 3^c$ for $c < c_0$. Solving the equation above for c gives a numerical value of $c \approx 0.7571$, and finally the desired runtime of $O^*(2.2973^n)$. Concluding:

Theorem 3. *Given two trees T, T' on n taxa, an unrooted maximum agreement forest for T and T' can be computed in time $O^*(2.2973^n)$.*

3.2 An $O^*(2.0649^n)$ algorithm for finding a rMAF

A *rooted* agreement forest is a variant of an agreement forest defined on rooted trees i.e. trees where there is a designated root and all arcs are oriented away from it. The corresponding optimisation problem Rooted Maximum Agremeent Forest (rMAF) is also NP-hard. We omit the definition and refer to recent survey articles such as [4] for an overview.

There is a $O^*(2.42^k)$ time algorithm for determining whether two rooted trees T, T' have a rooted agreement forest of size at most k [29]. In the worst case iterative deepening will yield an algorithm with running time $O^*(2.42^n)$. Prior to this article this was the best-known exponential-time (as opposed to parameterised) algorithm for this problem. With the same arguments as in the previous section we can reduce the base of the exponent. It is easy to show that, given a convex character on T, one can test in polynomial time if this induces a rooted agreement forest for T and T'. Thus we only have to solve the following equation for c.

$$2.42^{cn} = e^{(2-c)nH(\frac{c}{2-c})}$$

This yields $c \approx 0.8204$, which in turn results in a runtime of $O^*(2.0649^n)$.

Theorem 4. *Given two rooted trees T, T' on n taxa, a rooted maximum agreement forest for T and T' can be computed in time $O^*(2.0649^n)$.*

4 Maximum parsimony distance on two-state characters

Recall from the preliminaries the definition of the *parsimony score* $l_f(T)$ where f is a character and T is a phylogenetic tree on X. This is the foundation of the following optimisation problem, introduced in [10, 22]. Informally, the goal is to identify a character that has high parsimony score on one of the input trees and small on the other.

Definition 1. *Given two phylogenetic trees T and T' on the same set X of taxa, the parsimony distance between these trees is defined as*

$$d_{MP}(T, T') = \max_f |l_f(T) - l_f(T')|$$

A character f which maximises this distance is called an optimal character.

Note that, due to the fact that the parsimony score of a tree (for a given character) is not affected by the presence or absence of a root, parsimony distance is oblivious to whether the input trees are rooted or unrooted.

The problem is NP-hard and has attracted quite some attention in recent years. See recent articles such as [17, 28] for an overview of its theoretical and practical significance; in particular, it can be leveraged to efficiently generate very strong lower bounds on $d_{\text{uMAF}}(T, T')$. Kelk and Stamoulis [20] described and implemented an $O^*(1.6181^n)$ algorithm that calculates d_{MP} for two trees. The algorithm exploits the fact, proven in [18], that there exists an optimal character f which is convex on one of the two input trees and which has at least two taxa per state.

In this section we study a different variant of the problem, known as d^2_{MP}. The definition is the same except that we are restricted to characters f that only have two states. This problem, which yields lower bounds on d_{MP}, is also NP-hard [18] and permits a trivial $O^*(2^n)$ time algorithm simply by guessing the state of each taxon in X; see also [21] for further discussion of the problem. Interestingly, the $O^*(1.6181^n)$ time algorithm for the d_{MP} problem does not work for d^2_{MP}. This is because the pivotal assumption, that there exists an optimal character that is convex on one of the two trees, no longer holds when restricted to just two states. Nevertheless, with some effort we can establish a link with convex characters. We will use this link to prove the same $O^*(1.6181^n)$ runtime for d^2_{MP} - and then via matchings we will reduce the base of the exponent further.

We say that an extension g of a character f on a tree T is *optimal* if $l_g(T) = l_f(T)$ i.e. it is an extension which achieves the minimum number of mutations. From this point on we will refer to the states in a two-state character as red and blue.

Before proving our main theorem we first require some auxiliary lemmas.

Lemma 1. *Let T be a tree on X and f_2 a two-state character on X. Let g_2 be an optimal extension of f_2 to T. For every internal node u of T, at most one neighbour of u has a colour different to $g_2(u)$.*

Proof. Suppose that there is an internal node u which does have more than one neighbour coloured with a colour different to $g_2(u)$. This means that it has two or three neighbours with a different colour. Suppose it has three. Let w.l.o.g. u be red and all its neighbours blue. Flipping u to blue would reduce the number of mutations by three, contradicting the fact that g_2 is an optimal extension. Suppose it has two. In this case flipping the colour of u reduces the number of mutations by one which again contradicts the fact that g_2 is optimal. □

We obtain a small but useful corollary thanks to this lemma.

Corollary 1. *Let T be a tree on X and f_2 a two-state character on X. Let g_2 be an optimal extension of f_2 to T. Consider the forest of connected components obtained by deleting the mutation edges in g_2, i.e. edges $\{u, v\}$ where $g_2(u) \neq g_2(v)$. Every vertex with degree 0 in this forest is labelled by a taxon from X, and every leaf (i.e. degree 1 node) in this forest is labelled by a taxon from X.*

Proof. If there is a degree-0 node u in the forest that is not labelled by a taxon from X, then u is an internal node of T, and the colour assigned to u by g_2 is distinct from that assigned to all 3 of its neighbours; this contradicts Lemma 1. Similarly, suppose there is a leaf u in a connected component of the forest that is not labelled by a taxon. The leaf must, again, be an internal node of T. Meaning that, in g_2, it has two neighbours with a different colour to $g(u)$. This also contradicts Lemma 1. □

The following lemma resembles Observation 3.3 from Kelk and Fischer [18] but is now rewritten in our notation and with an extended proof.

Lemma 2. *Let T and T' be the input to the problem d^2_{MP}. There exists an optimal two-state character f_2, a tree $T^* \in \{T, T'\}$ and an optimal extension g_2 of f_2 to T^*, such that every connected component of the forest obtained by cutting mutation edges of T^* (with respect to g_2), contains at least two taxa.*

Proof. Let f_2 be an optimal two state character. Assume without loss of generality that $l_{f_2}(T) \leq l_{f_2}(T')$. Let g_2 be an optimal extension of f_2 to T and consider the forest induced by cutting the mutation edges of g_2 in T. Suppose that there is a connected component in the forest with a single node x. From Corollary 1 we know that x is labelled by a taxon, and therefore must have had a different colour to its unique parent in g_2. Flipping the colour of x will create a new two-state character f'_2. Let g'_2 be the extension of f'_2 to T which assigns states to interior nodes of T in the same way as g_2. Extension g'_2 has one fewer mutation than g_2, so $l_{f'_2}(T) \leq l_{f_2}(T) - 1$. However, $l_{f'_2}(T') \geq l_{f_2}(T') - 1$. To see why this is, observe that flipping the state of a single taxon can reduce the parsimony score of a character on a given tree by at most one. Hence, $l_{f'_2}(T') - l_{f'_2}(T) \geq l_{f_2}(T') - l_{f_2}(T) = d^2_{MP}(T, T')$, so f'_2 is also an optimal character, and g'_2 is an optimal extension of it to T. Observe that the forest induced by g'_2 has one fewer singleton component than g_2. We iterate this process until we have an optimal two-state character f^*_2, and a corresponding optimal extension g^*_2 of it to T, such that no singleton components appear in the induced forest. At this point, every connected component

in the induced forest contains at least two nodes, and thus at least two leaves; by Corollary 1 these leaves must be labelled by taxa. So we are done. □

We define a *partial* extension of a character to a tree T similarly to an extension, except that some interior nodes of T potentially have no state assigned to them; we call these nodes *uncovered*.

Definition 2. *Consider a convex character f on a tree T. The natural partial extension is the partial extension obtained by, for each state, taking the minimal connecting subtree of all the taxa within that state. If this partial extension covers every node of T (i.e. it is an extension) we call it the natural covering extension.*

Lemma 3. *Let f be a convex character on a tree T. Every optimal extension of f to T can be obtained by expanding the natural partial extension of f to T to the uncovered nodes. In other words, the natural partial extension is unavoidable in an optimal extension.*

Proof. Towards a contradiction, suppose there exists a convex character f on T and an optimal extension g of f to T that is not an expansion of the natural partial extension. Then, for some state s of f, at least one node on its minimal connecting spanning tree is not assigned to state s. As a result, at least one edge on the minimal connecting spanning tree for s is a mutation edge in g. Hence, if one cuts mutation edges in g, the state s will occur in two or more connected components of the resulting forest. Every state of f must occur in at least one connected component of the forest, so the number of connected components in the forest is at least equal to the number of states in f plus 1. The number of connected components in this forest is $l_f(T) + 1$, so $l_f(T)$ is at least equal to the number of states in f. However, a convex character on T has by definition the property that its parsimony score on T is equal to the number of states in the character *minus one*. Hence, f cannot possibly be convex; contradiction. □

The following lemma is actually somewhat stronger than we need - we only require one direction of the implication - but for completeness we prove both directions.

Lemma 4. *Let f be a convex character on a tree T. The natural covering extension of f exists if and only if there is a* unique *optimal extension of f to T.*

Proof. The fact that the existence of the natural covering extension implies uniqueness, follows immediately from Lemma 3. To prove the other direction, we use the contrapositive. Assume that the natural covering extension does *not* exist. Then there exists an interior node u that is uncovered by the natural partial extension. Let g be an optimal extension of f to T. We will use an algorithmic argument to prove that g cannot be unique. Let p, q, r be the three neighbours of u in T and let T_w, where $w \in \{p, q, r\}$, be the pendant subtree of T rooted at w. Let S_w, where $w \in \{p, q, r\}$, be the set of states appearing on taxa in T_w, with respect to f. Crucially, S_p, S_q and S_r are mutually disjoint, because otherwise u would have been covered in the natural partial extension. Now, we root T by subdividing the edge $\{u, r\}$ with a root node ρ and we call this new tree T'. We run Fitch's algorithm on T' to compute an optimal extension of f to T'; we provide a summary of the algorithm in Appendix A. We note that an optimal extension for T can be obtained from an optimal extension for T' simply by ignoring the state assigned to the root ρ; hence, if two optimal extensions for T' differ on a node that is distinct from ρ, then there exist at least two different optimal extensions for T. Towards this result, observe that in the bottom-up phase of Fitch's algorithm, a nonempty set $S'_w \subseteq S_w$ of states will be assigned to node w, $w \in \{p, q, r\}$. By the disjointness of the S_w, the S'_w are also disjoint. Hence, in the bottom-up phase the set $S'_p \cup S'_q \cup S'_r$ of states will be assigned to the root. Now, in the top-down phase, we are allowed to assign any state from $S'_p \cup S'_q \cup S'_r$ to the root. If we pick a state $s \in S'_p$, the node u will definitely be assigned state s. If we pick a state $t \in S'_q$, u will definitely be assigned t. Given that $s \neq t$, due to disjointness of S'_q and S'_p, $u \neq \rho$, and the fact that all possible executions of the top-down phase of Fitch's algorithm produce an optimal extension, we conclude that there exists more than one optimal extension of f to T. □

Let T and T' be the input to the problem d^2_{MP}. Let f_2 be an optimal two-state character, g_2 an optimal extension of f_2 and $T^* \in T, T'$ such that every connected component of the forest obtained by cutting mutation edges of T^* (with respect to g_2), contains at least two taxa. These properties and existence are guaranteed by Lemma 2. Consider the forest obtained by cutting mutation edges (with respect to g_2) in T^*. Let f_c be the character (i.e. partition of X) induced by the connected components of the forest. By the same lemma, each state of f_c contains at least two taxa. Moreover, by construction, f_c is convex on T^*. Next, we have a crucial lemma.

Lemma 5. *The natural partial extension of f_c to T^* is a natural covering extension.*

Proof. Recall that f_c was constructed from an optimal extension g_2 of an optimal two state character f_2. We know from Lemma 1 that, in g_2, every internal node u of T^* must have at least two neighbours with the same colour as u. Thus, in the forest obtained from g_2, every internal node must have degree 2 or 3 (and every component of the forest

contains at least 2 taxa). Furthermore we also know from Corollary 1 that every leaf in a component of the forest is labelled by a taxon. This means that for every internal node u of T^*, there exist two distinct taxa $x, y \in X$ such that x and y are assigned the same state by f_c, and u lies on the unique path from x to y in T^*. Given that the natural partial extension was created using minimal spanning trees, it follows that the natural partial extension assigns a state to every internal node of T^*, and thus is a natural covering extension. □

From Lemma 4 and 5 we obtain the following corollary.

Corollary 2. *The convex character f_c has a* unique *optimal extension on T^*, which is the natural covering extension.*

In the remainder of the paper we shall call convex characters where the natural covering extension exists (which by Lemma 4 are exactly those characters with unique optimal extensions) *covering convex characters*. We have shown that f_c is such a character[4]. Now we have everything to prove our main theorem.

Theorem 5. *Let T and T' be the input to the problem d^2_{MP}. There exists a covering convex character f_c on one of the trees, say T, such that (i) each state of f_c contains at least 2 taxa and (ii) the states of f_c can in polynomial time be relabelled red and blue, obtaining a 2 state character f_2, such that $d^2_{MP}(T,T') = |l_{f_2}(T) - l_{f_2}(T')| = l_{f_2}(T') - l_{f_2}(T)$.*

Proof. The earlier lemmas and corollaries prove the existence of a covering convex character f_c on $T^* \in \{T, T'\}$. We know that (i) holds, by construction of f_c. Now, we know that the desired character f_2 exists - because we used it (together with g_2) to build f_c in the first place - but to satisfy (ii) we have to show how to efficiently obtain f_2 given only f_c. We show (up to symmetry of red and blue) that f_2 is completely determined by f_c, and can easily be obtained from it.

Crucially, due to being a covering convex character, there is a unique optimal extension of f_c to T^*, which is the natural covering extension. Due to this uniqueness, any standard polynomial-time algorithm used for computing optimal extensions (e.g. Fitch's algorithm [11]) will obtain it; let us call this g_c. This natural covering extension has, by construction, exactly the same mutation edges as f_2. Hence, f_c partitions X exactly the same way as the deletion of mutation edges in g_2 partitions X. We now need to decide which states of f_c will become blue, and which will become red.

We define a new graph $G = (V, E)$ as follows.

V: For every connected component in the forest obtained by cutting mutation edges of g_c in T^*, we add a vertex to V representing the connected component.

E: For every mutation edge $\{u_1, u_2\}$ in the optimal extension g_c, we add the edge $\{v_1, v_2\}$ where the endpoints v_1 and v_2 are the connected components that u_1 and respectively u_2 belong to.

Note that G is still a tree; if it had cycles then so would T^*. Recall from Corollary 2 that g_c is unique. This means that G is also unique. Because G is a connected tree we can see it as a bipartite graph with a unique bipartition. This bipartition of X forms a two-state character. And because of the uniqueness of G the mapping from f_c to a two-state character is well-defined.

Finally, we note that the construction of G and the creation of the two-state character can be performed in polynomial time. This concludes the proof. □

Now that Theorem 5 has been proven it is time to use it algorithmically. To compute $d^2_{MP}(T,T')$ we loop over all covering convex characters f_c of T, derive f_2 from it, compute $|l_{f_2}(T) - l_{f_2}(T')|$ and note the largest value we see. We then symmetrically repeat the procedure for T', simply because we don't know whether T^* is T or T'. The character f_2 producing the largest value will be an optimal character.

All that remains is producing an algorithm that can enumerate all covering convex characters f_c of a tree T. An elementary approach is to loop through the space of all convex characters where each state has at least two taxa, and discard those that are not covering convex characters. We can test in polynomial time whether a convex character is a covering convex character by constructing the natural partial extension and checking whether it covers all nodes of T. Kelk and Stamoulis [20] proved that there are $\Theta(\phi^n)$ convex characters that have at least 2 taxa in each state and they can be efficiently listed. Putting all these pieces together, we end up with a running time of $O^*(\phi^n)$ for calculating $d^2_{MP}(T,T')$.

Theorem 6. *Given two trees T, T' on a set X of n taxa, $d^2_{MP}(T,T')$ can be computed in time $O^*(\phi^n)$.*

But we can do better! In the next section we will show that there are actually at most $O(1.5895^n)$ covering convex characters with at least two taxa per state, and that these can be efficiently listed, immediately giving a faster algorithm for d^2_{MP}. To prove this, we will turn to the topic of matchings.

[4]Note that not all convex characters are covering. For example, consider the unique phylogenetic tree on 3 taxa. If each of these three taxa is assigned a different state by a character, the single interior node of the tree will be uncovered by the natural partial extension. The 8 convex characters shown in Figure 1 are, however, covering.

5 Matchings on a core tree

We proved that f_c, from which we can construct an optimal two-state character for $d^2_{MP}(T, T')$, is a covering convex character on one of the input trees, and has at least two taxa in every state. The following observation shows that some characters with these properties can be safely ignored. This will ultimately allow us to improve the running time given in Theorem 6.

Suppose, without loss of generality, that f_c has a lower parsimony score on T than T'. Suppose furthermore that T contains two taxa a and b whereby the path from a to b has exactly three edges, and that f_c assigns a and b the same state - but that no other taxa are assigned this state. See Figure 4. An optimal extension here will be the natural covering extension, so the two red dashed edges in the figure will definitely be mutation edges in this extension. If f_c is mapped back to f_2 in the way described in Theorem 5, f_2 will assign a and b the same colour (say, blue) and the two mutation edges will be mutations from blue to red. But suppose, similar to the arguments used in Lemma 2, that we flip a and b both to red in f_2, obtaining a new two-state character f'_2. We have that $l_{f'_2}(T) \leq l_{f_2}(T) - 2$, because the two mutation edges disappear, but we also have $l_{f'_2}(T') \geq l_{f_2}(T') - 2$, because changing the state of two taxa can reduce the parsimony score by at most 2. Hence f'_2 is also an optimal character for the d^2_{MP} instance, and still obeys the additional criteria specified in Lemma 2. In the same way f_c was obtained from f_2, we can now construct f'_c from f'_2. An algorithm that can locate f'_c is thus also guaranteed to optimise d^2_{MP}. The important fact is that in an optimal extension of f'_c, neither of the red edges shown in Figure 4 will be mutation edges.

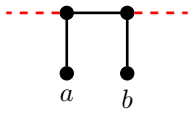


Figure 4: Red dashed edges are mutating (i.e. bichromatic) edges. When solving d^2_{MP} we can safely ignore covering convex characters whose natural covering extensions have this local 'island' structure.

This observation shows that we need to find a way to skip over covering convex characters that have the kind of pointless 'islands' described above and shown in Figure 4. To do this we turn to *matchings*. Recall that a matching of a graph is simply a subset of edges that do not share any endpoints. We will show a natural bijective correspondence between matchings (on an appropriately transformed tree) with covering convex characters that have at least two taxa in each state.

Definition 3. *The core tree T_c for a phylogenetic tree T on X is the non-phylogenetic tree that remains when every node labelled by X is removed.*

Lemma 6. *There exists a bijective function between covering convex characters of T with at least two taxa per state and matchings in T_c.*

Proof. We will prove this lemma by constructing two injective functions. The first one will map a covering convex character into a matching and the second one will be the other way round. Let f_c be a covering convex character on T with at least two taxa per state. We map f_c to the set of mutation edges in the unique optimal extension of f_c to T provided by Corollary 2. Note that this mapping is well-defined, since the optimal extension (and thus the set of induced mutation edges) is unique. Observe that none of these mutation edges can be incident to an element of X, because there are at least two taxa per state in f_c, so all these edges survive in T_c. Now, suppose the mutation edges do not form a matching in T_c. This means that there are two mutation edges that share an endpoint in T_c, and thus also in T. This is in contradiction with Lemma 1. Hence, the mutation edges must form a matching in T_c. To see that the mapping from f_c to matchings in T_c is injective, suppose f_c and f'_c are two distinct covering convex characters on T i.e. they induce different partitions of X. Then there exist $x, y \in X$ such that $x \neq y$ and without loss of generality x, y belong to the same state s in f_c but to different states in f'_c. In the unique optimal extension of f_c to T, all edges on the path from x to y must be covered by the spanning tree for s. However, in the unique optimal extension of f'_c to T, at least one edge on this path does not lie on any of the spanning trees induced by the states of f'_c - otherwise x, y would belong to the same state[5]. Hence the subset of edges of T that are covered by spanning trees in the unique optimal extension of f_c to T is not the same subset as those corresponding to f'_c. The set of mutation (and thus matching) edges is exactly those *not* on spanning trees, and as noted earlier all these mutation edges survive in T_c, so f_c and f'_c induce different matchings in T_c. Hence, injectivity holds.

[5]Recall that the spanning trees will be vertex-disjoint, because the tree has maximum degree 3, so it cannot occur that the path is the disjoint union of several spanning trees from distinct states.

In the other direction, consider a matching M of T_c. Observe that degree-1 nodes in T_c correspond to internal nodes of T that have two taxa children, and degree-2 nodes in T_c to internal nodes of T that have one taxon child. Suppose we delete the edges in M from T_c, obtaining the forest $T_c - M$. We map M to the natural partition of X induced in T by the connected components of $T_c - M$, i.e. each node in $T_c - M$ contributes the taxa it is adjacent to in T. This mapping is clearly well-defined. If we can show that each connected component of $T_c - M$ contains at least one degree-1 node (or at least two degree-2 nodes), where here the degrees refer to T_c before the deletion of M, it follows that the mapping produces a partition of X whereby each block contains at least two taxa. Suppose a connected component consists only of a single node. Then it must be a degree-1 node, and thus correspond to 2 taxa, so we are fine. (It cannot be a degree-2 or degree-3 node, because then the component would contain at least one edge). So suppose a connected component contains at least one edge, and thus has at least two leaves in $T_c - M$. Note that nodes of a connected component of $T_c - M$ *cannot* be degree-3 nodes in T_c, because this would mean that the connected component could actually be grown further: this is because there can be at most one matching edge incident to a degree-3 node. So for a connected component in $T_c - M$ with at least two leaves in $T_c - M$ each of the leaves must be a degree-1 or degree-2 node. Hence, we indeed obtain a partition of X with at least two taxa per block. To see that this partition is in fact a covering convex character, observe that the connected components of $T_c - M$ are (when projected onto T) minimal spanning trees for the blocks of X. They are minimal because, as noted above, the leaves of components in $T_c - M$ must be degree-1 or degree-2 nodes in T_c, and such nodes are (in T) incident to taxa. The minimal spanning trees cover all nodes of T, because the connected components of $T_c - M$ cover all nodes of T_c. From this we conclude that the partition of X that we created is indeed a covering convex character (and that the edges in M are precisely the mutation edges in the unique optimal extension for this character).

Finally, to see that the mapping from matchings to characters is injective, suppose two matchings M, M' of T_c map to the same covering convex character f_c with at least two taxa per state. As stated above, M and M' are mutation edges in an optimal extension of f_c to T. By Corollary 2 a covering convex character has a unique optimal extension, so $M = M'$. □

Definition 4. *We call a matching M in T_c* illegal *if $T_c - M$ has a component consisting of a single edge that is adjacent to two edges in M. Otherwise it is called a* legal *matching.*

Observe that illegal matchings induce f_c with the 'islands' discussed at the start of the section, and shown in Figure 4. We know it does not compromise optimality to ignore such characters. Hence, in solving d^2_{MP} we can henceforth focus on counting and listing only legal matchings.

What is an upper bound for the number of legal matchings in T_c, and thus an upper bound for the number of candidate f_c? To answer this question we establish a recursion, which we will also use to create a DP algorithm to enumerate all the legal matchings. When bounding the recursion we will require some auxiliary support from the software package Mathematica; we shall explain this further in due course.

To calculate the upper bound, we apply the method described in [23] (and earlier in [24]). To this end, we need to introduce some additional notation.

In the remainder of this section, to ensure relevance to the wider matchings literature, T will refer to any non-phylogenetic tree which has maximum degree 3. (In our specific d^2_{MP} applications context, T will be the tree T_c created earlier in the section, but this extra information is not necessary.) Again, to keep consistency with the matchings literature, we will let n henceforth refer to the total number of nodes in T, not just the number of leaves. Fortunately, this will not asymptotically distort any running times when we translate the results back to the phylogenetic context. That is because a phylogenetic tree with $|X|$ leaves has exactly $|X| - 2$ internal nodes, so T_c (where we enumerate legal matchings) has exactly $|X| - 2$ nodes in total.

We consider all trees as *rooted* trees of maximum degree 3, where each node has at most one left child and at most one right child. We select an arbitrary leaf of the tree as the root; this allows us to avoid certain technicalities later on.

We show how to count legal matchings in a recursive fashion, going up from the leaves to the root. For this purpose, it will be convenient to allow empty trees with no nodes in the following. Define

$$e(T) = [T \text{ is empty}] = \begin{cases} 1 & T \text{ is an empty tree,} \\ 0 & \text{otherwise.} \end{cases}$$

Moreover, we consider two special types of matchings of a rooted tree T. In this context, we think of T as a rooted subtree of a larger rooted tree (e.g., the subtree rooted at u in Figure 5):

- Legal matchings where the root has only one child and only one grandchild, the root is not covered by a matching edge, but its child is. The number of these matchings is denoted by $b_0(T)$ and illustrated in Figure 5.

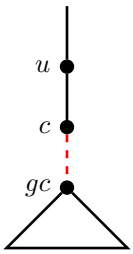


Figure 5: Legal matchings counted by $b_0(T)$, where T is the subtree rooted at u. A red dashed edge represents an edge in the matching. As in the other figures in this section the edge entering from above represents the edge that enters the subtree in the original tree.

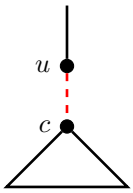


Figure 6: Legal matchings counted by $b_1(T)$. A red dashed edge represents an edge in the matching.

- Legal matchings where the root has one child only and is covered by a matching edge. Their number is denoted by $b_1(T)$ and illustrated in Figure 6.

We denote the number of remaining legal matchings not covering the root by $a_0(T)$, illustrated in Figure 7.

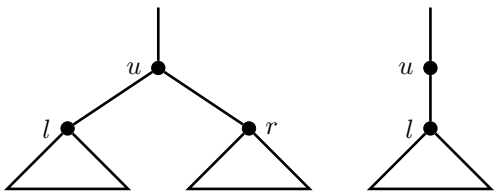


Figure 7: Legal matchings counted by $a_0(T)$.

Finally, we let $a_1(T)$ be the number of remaining legal matchings covering the root of T, illustrated in Figure 8.

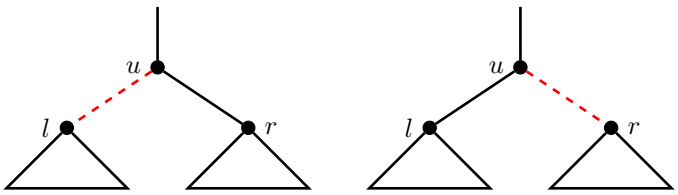


Figure 8: Legal matchings counted by $a_1(T)$. A red dashed edge represents an edge in the matching.

If T is empty, we set $a_0(T) = a_1(T) = b_0(T) = b_1(T) = 0$. Let T be a rooted tree whose root branches are T_l and T_r (possibly empty). More formally, T_l (respectively, T_r) is the subtree of T rooted at the left (respectively, right) child of T. In the following recursions, whereby T can be thought of as representing a subtree of a larger tree, we emphasise that a_0, a_1, b_0, b_1 count matchings that are legal on the *original* tree, not just the subtree. This is important because although subtrees T_l, T_r encountered in the recursion can potentially have a root of degree 1 or 2 (or even degree 0), the subtree will have an incoming edge when viewed in the context of the original tree. This ensures that some matchings which would be illegal if the subtree was considered in isolation, but which are legal when viewed in

the context of the original tree, are correctly counted.

$$\begin{aligned}
a_0(T) &= \big(a_0(T_l) + a_1(T_l) + b_0(T_l) + b_1(T_l) + e(T_l)\big)\big(a_0(T_r) + a_1(T_r) + b_0(T_r) + b_1(T_r) + e(T_r)\big) \\
&\quad - b_1(T_l)e(T_r) - e(T_l)b_1(T_r), \\
a_1(T) &= a_0(T_l)\big(a_0(T_r) + a_1(T_r) + b_0(T_r) + b_1(T_r)\big) + \big(a_0(T_l) + a_1(T_l) + b_0(T_l) + b_1(T_l)\big)a_0(T_r), \\
b_0(T) &= b_1(T_l)e(T_r) + e(T_l)b_1(T_r), \\
b_1(T) &= a_0(T_l)e(T_r) + e(T_l)a_0(T_r).
\end{aligned}$$

Note that, if T is a single node, $a_0(T)$ evaluates to 1 (corresponding to the empty matching), but $a_1(T)$, $b_0(T)$ and $b_1(T)$ all evaluate to 0. The reasoning behind the recursion is as follows. We obtain a matching not covering the root in T by combining an arbitrary matching in T_l (possibly the empty matching if T_l is empty) and an arbitrary matching in T_r (also possibly empty). If both components are legal, then so is the new matching. It is a special matching counted by b_0 if one of the two branches is empty and the matching chosen in the other branch is a special matching counted by b_1.

Likewise, we obtain a matching covering the root by first choosing one of the root edges, e.g. the edge between the root of T and the root of T_l. Then we combine it with an arbitrary matching in T_r and a matching not covering the root of T_l (or vice versa if the other root edge was chosen). For the matching to remain legal, the matching in T_l may not be of special type b_0. A special matching b_1 is obtained in this way if one branch is empty.

It is useful to write this recursion in matrix form: associating a vector

$$\mathbf{V}(T) = \big[a_0(T), a_1(T), b_0(T), b_1(T), e(T)\big]^T$$

to a tree T, we have $\mathbf{V}(T) = B\big(\mathbf{V}(T_l), \mathbf{V}(T_r)\big)$, where the map $B : \mathbb{R}^5 \times \mathbb{R}^5 \to \mathbb{R}^5$ is defined by

$$B\Big(\begin{bmatrix} v_1 \\ w_1 \\ x_1 \\ y_1 \\ z_1 \end{bmatrix}, \begin{bmatrix} v_2 \\ w_2 \\ x_2 \\ y_2 \\ z_2 \end{bmatrix}\Big) = \begin{bmatrix} (v_1 + w_1 + x_1 + y_1 + z_1)(v_2 + w_2 + x_2 + y_2 + z_2) - y_1z_2 - z_1y_2 \\ (v_1 + w_1 + x_1 + y_1)v_2 + v_1(v_2 + w_2 + x_2 + y_2) \\ y_1z_2 + z_1y_2 \\ v_1z_2 + z_1v_2 \\ 0 \end{bmatrix}.$$

Note that B is a bilinear map: we have

$$B(\mathbf{v}_1 + \mathbf{w}_1, \mathbf{v}_2 + \mathbf{w}_2) = B(\mathbf{v}_1, \mathbf{v}_2) + B(\mathbf{w}_1, \mathbf{v}_2) + B(\mathbf{v}_1, \mathbf{w}_2) + B(\mathbf{w}_1, \mathbf{w}_2)$$

and

$$B(c_1\mathbf{v}_1, c_2\mathbf{v}_2) = c_1c_2B(\mathbf{v}_1, \mathbf{v}_2).$$

Note also that it has only nonnegative coefficients: the terms y_1z_2 and z_1y_2 in the first component actually cancel if the product is multiplied out. Furthermore, the vector associated with the empty tree is $[0, 0, 0, 0, 1]^T$.

Theorem 7. *The maximum number M_n of legal matchings in a tree with n nodes is $O(\alpha^n)$ with $\alpha = (13384 + 8\sqrt{2793745})^{1/22} \approx 1.5895$.*

Proof. There exists a set $\mathcal{S}$ of 62 5-dimensional vectors with nonnegative entries that have the following property:

(1) It contains the vector $[0, 0, 0, 0, 1/\alpha]^T$,

(2) For any pair of two vectors $\mathbf{v}_1, \mathbf{v}_2 \in \mathcal{S}$, the vector $B(\mathbf{v}_1, \mathbf{v}_2)$ lies in the set

$$\mathit{conv}_{\le}(\mathcal{S}) = \big\{\mathbf{w} \in \mathbb{R}^5 \,:\, \mathbf{w} \ge \mathbf{0}, \mathbf{w} \le \sum_{\mathbf{v}\in\mathcal{S}} c_{\mathbf{v}}\mathbf{v} \text{ for some constants } c_{\mathbf{v}} \ge 0, \sum_{\mathbf{v}\in\mathcal{S}} c_{\mathbf{v}} = 1\big\}.$$

Here, the inequalities hold componentwise. Note that $\mathit{conv}_{\le}(\mathcal{S})$ is a bounded and convex set by construction. The set $\mathcal{S}$ is listed in Appendix B. It can be verified directly (with the help of a computer) that the second condition is indeed satisfied. For this purpose, we have provided a comprehensively annotated Mathematica file in the supplementary material.

Given this property of $\mathcal{S}$, we can now prove the following by induction on n: for every rooted binary tree T with n nodes, the vector $\alpha^{-n-1}\mathbf{V}(T)$ lies in $\mathit{conv}_{\le}(\mathcal{S})$. This is trivial for $n = 0$, since we get the vector $[0, 0, 0, 0, 1/\alpha]^T$ for the empty tree, which lies in $\mathcal{S}$ by property (1) and thus in turn in $\mathit{conv}_{\le}(\mathcal{S})$. For the induction step, we can apply

property (2) of $\mathcal{S}$. Assume that the two branches T_l and T_r (possibly empty) of T satisfy the statement, and let them have k and $n-k-1$ nodes respectively. We have

$$\alpha^{-n-1}\mathbf{V}(T) = \alpha^{-n-1}B(\mathbf{v}_1, \mathbf{v}_2) = B(\alpha^{-k-1}\mathbf{v}_1, \alpha^{-n+k}\mathbf{v}_2).$$

By the induction hypothesis, both $\alpha^{-k-1}\mathbf{v}_1$ and $\alpha^{-n+k}\mathbf{v}_2$ lie in $conv_{\leq}(\mathcal{S})$, so there exist linear combinations of the elements of $\mathcal{S}$ with nonnegative coefficients such that

$$\alpha^{-k-1}\mathbf{v}_1 \leq \sum_{\mathbf{v}\in\mathcal{S}} c_{\mathbf{v}}^{(1)}\mathbf{v}$$

and

$$\alpha^{-n+k}\mathbf{v}_2 \leq \sum_{\mathbf{v}\in\mathcal{S}} c_{\mathbf{v}}^{(2)}\mathbf{v},$$

thus by (bi-)linearity of B

$$\begin{aligned}\alpha^{-n-1}\mathbf{V}(T) &= B(\alpha^{-k-1}\mathbf{v}_1, \alpha^{-n+k}\mathbf{v}_2)\\ &\leq \sum_{\mathbf{v}\in\mathcal{S}}\sum_{\mathbf{w}\in\mathcal{S}} c_{\mathbf{v}}^{(1)}c_{\mathbf{w}}^{(2)}B(\mathbf{v},\mathbf{w}).\end{aligned}$$

Since $B(\mathbf{v},\mathbf{w}) \in conv_{\leq}(\mathcal{S})$ for all $\mathbf{v}$ and $\mathbf{w}$ in $\mathcal{S}$ by property (2) and $conv_{\leq}(\mathcal{S})$ is convex, it follows that $\alpha^{-n-1}\mathbf{V}(T) \in conv_{\leq}(\mathcal{S})$, completing the induction.

In particular, we have shown that the entries of the vector $\alpha^{-n-1}\mathbf{V}(T)$ are bounded. The total number of legal matchings of T is the sum of the entries of $\mathbf{V}(T)$, so it follows that this number is $O(\alpha^n)$. □

Remark. The set $\mathcal{S}$ in our proof was constructed by iteration as described in [23], starting with the two vectors $[0,0,0,0,1/\alpha]^T$ and $[\alpha^{-7}(4+6616/\sqrt{2793745}), 0, 0, \alpha^{-7}(2+3446/\sqrt{2793745}), 0]$. The choice of the two vectors is justified as follows: the first vector corresponds to the empty tree; the second stems (as a limit point) from the sequence of trees that yields the lower bound presented in the appendix. If one was to apply the iteration purely starting with the first vector, the iteration process would only produce vectors getting arbitrarily close to the second, without ever terminating.

Now that the upper bound on the number of legal matchings has been proved, all we need is a dynamic programming algorithm that allows us to enumerate every legal matching. As noted earlier, we take T_c as the tree T. We have a recursion that allowed us to count all the legal matchings in T. Thus, just like the algorithm in section 3, we use this to calculate the exact number of legal matchings in T_c and use the recursion to backtrack in T_c to enumerate each legal matching. Specifically, observe that the terms a_0, a_1, b_0, b_1 together sum to the number of legal matchings, and that each of these terms is itself a summation of terms; note here that the negative terms in a_0 actually cancel when the product is expanded. Hence, it is straightforward to canonically index the legal matchings and to use this to steer the backtracking.

After that we can transform the matchings back to a corresponding covering convex character f_c in T. The states can then be coloured red-blue in the fashion described earlier, to obtain all possible f_2, and we are done. This brings us to our main theorem:

Theorem 8. *For a pair of binary trees T and T' on $|X|$ taxa we can compute d_{MP}^2 in $O(\alpha^{|X|}\cdot poly(|X|))$ time with $\alpha = (13384+8\sqrt{2793745})^{1/22} \approx 1.5895$.*

Proof. Theorem 5 proves that the process of enumerating all covering characters f_c on T with at least two taxa per state will give us a f_2 such that $d_{MP}^2(T,T') = |l_{f_2}(T) - l_{f_2}(T')| = l_{f_2}(T') - l_{f_2}(T)$ when T has the lower parsimony score at optimality. Kelk and Fischer [18] proved that $d_{MP}^2(T,T')$ is asymmetrical, in the sense that, if we range over all two-state characters f, the maximum value of $l_f(T') - l_f(T)$ might differ from the maximum of $l_f(T) - l_f(T')$. This means we don't know *a priori* which of T and T' has the lowest parsimony score at optimality. Therefore we simply enumerate the f_c of both trees. This has no impact on the asymptotic runtime.

The process of creating T_c (and in a second iteration, T_c') can be achieved in polynomial time. The combination step in the dynamic programming algorithm - where we use the values of two child nodes to derive the values of the parent node - can be done in constant time and thus the whole dynamic programming algorithm is clearly polynomial. A legal matching can be translated to a covering convex character f_c in polynomial time. The steps taken for each f_c to create f_2 can also be done in polynomial time. Optimal extensions, and thus the parsimony score, can be computed in polynomial time so each $|l_{f_2}(T) - l_{f_2}(T')|$ is also polynomial-time computable. Theorem 7 proves that there are at most $O(\alpha^n)$ legal matchings in T_c, where in that theorem n refers to the *total* number of nodes in T_c. Here $|X|$ refers

to the number of leaves/taxa in T. However, as observed earlier, $n = |X| - 2$, so the asymptotics do not change in the translation.

Hence, the dynamic programming algorithm will enumerate at most $O(\alpha^{|X|})$ natural covering extensions.

Combining everything gives the final runtime $O(\alpha^{|X|} \cdot \text{poly}(|X|))$. □

6 Technical discussion: lower bounds, fully legal matchings, and a new upper bound for matchings on restricted trees

In this section we prove a number of extensions of the upper bound result from the previous section.

6.1 A matching lower bound for legal matchings

First, we show in Lemma 9 in Appendix C that there is a lower bound matching the upper bound in Theorem 7. From this we conclude that, ranging over trees with n nodes and at most degree 3, the maximum number of legal matchings is not just $O(\alpha^n)$ but in fact $\Theta(\alpha^n)$, where $\alpha = (13384 + 8\sqrt{2793745})^{1/22} \approx 1.5895$.

The rather complicated structure of the trees that produce the matching lower bound, with a repeating pattern of length 22, also explains to some extent why a computer-assisted proof, based on the work of Rote [24] and Rosenfeld [23], was necessary. It seems very difficult, if not impossible, to prove the bound purely by hand.

6.2 A new upper bound on the number of matchings in a restricted subfamily of trees

Next, we translate the upper bound on the number of legal matchings in trees of maximum degree at most 3 to an upper bound on the number of matchings in *good* trees, henceforth defined as trees of degree at most 3, where no two degree 2 nodes are adjacent. To prove this we first need the following lemma.

Lemma 7. *For every tree T with maximum degree at most 3, there is a tree T^* with the same number of nodes, maximum degree at most 3 and no two adjacent nodes of degree 2 (i.e. a good tree) that has more or equally many legal matchings.*

Proof. Suppose that v_1 and v_2 are two adjacent nodes of degree 2 in T, and let their other neighbours be v_0 and v_3 respectively. Let the edges v_0v_1, v_1v_2 and v_2v_3 be denoted e_0, e_1, e_2 respectively. We remove the edge e_0 and replace it by $e_0' = v_0v_2$ to obtain a new tree T'. Now let Φ map e_0 to e_0' and all other edges to themselves. We show that $\Phi(M)$ is a legal matching of T' whenever M is a legal matching of T. Note first that the only edges that could be adjacent in $\Phi(M)$ (given that there are no adjacent edges in M) are e_0' and e_2. However, if both e_0 and e_2 are in M, then this would contradict the assumed legality of M. So $\Phi(M)$ is indeed a matching. Next, observe that the legality of M implies that at most one of e_0, e_1, e_2 are in M, and that in T', v_1 and v_2 no longer have degree 2. Hence, if $\Phi(M)$ was not a legal matching, it could only be due to $\Phi(M)$ containing a matching edge incident to v_0 or v_3. But then M was itself not legal. Hence, $\Phi(M)$ is indeed legal. As Φ is injective, T' has at least as many legal matchings as T. Iterating the construction, we end up with a tree that has no adjacent nodes of degree 2. □

Lemma 7 shows that, among trees with n nodes and of maximum degree 3, the number of legal matchings is maximised by some good tree. Moreover, by construction it follows that *every* matching in a good tree is a legal matching. The following result then follows immediately from Theorem 7.

Theorem 9. *A tree with n nodes and maximum degree 3, where no two degree 2 nodes are adjacent (i.e. a good tree), has $O(\alpha^n)$ matchings, where $\alpha = (13384 + 8\sqrt{2793745})^{1/22} \approx 1.5895$.*

Theorem 9 shows that the absence of adjacent degree 2 nodes is sufficient to slightly improve the well-known upper bound of $O(\phi^n)$ matchings in trees. The lower bound construction given in Lemma 9 in the appendix constructs good trees that asymptotically attain the bound of Theorem 9, so α cannot be lowered any further for good trees. We note that for trees where every internal node has degree 3, there are $O(1.5538^n)$ matchings (see [2, Theorem 1 and Remark 5]; the precise constant is $\sqrt{1+\sqrt{2}}$)).

6.3 Strengthening legality: full legality

When computing d^2_{MP} we could restrict our attention to legal matchings in the core tree T_c, because we showed that there exists at least one character f_c that optimises d^2_{MP}, such that f_c is induced by a legal matching. The central idea was that an *illegal* matching induces a covering convex character f whereby, in its unique optimal extension, an island

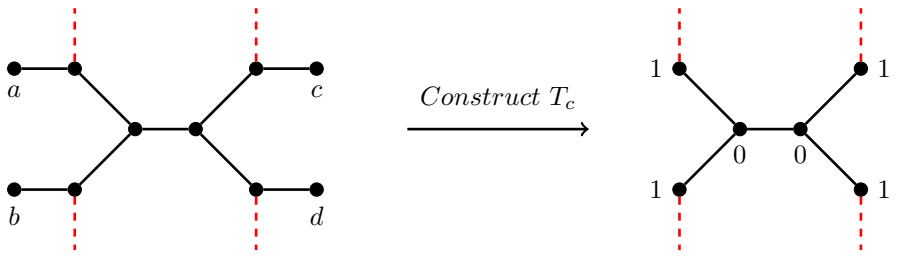


Figure 9: A red dashed edge represents an edge in the matching. This cannot be a fully legal matching, because in T_c the island enclosed by the 4 matching edges has total weight 4. Hence, such a matching can safely be ignored when optimising d^2_{MP}.

comprising two taxa is incident to two mutations. By flipping the state assigned by the character to the two taxa, we could obtain a new covering convex character f_c without the island, such that $|l_{f_c}(T') - l_{f_c}(T)| \geq |l_f(T') - l_f(T)|$.

We can extend this idea to bigger islands. This requires the introduction of weights to the nodes of T_c. The weight represents the number of taxa a node is connected to in T. Now, consider a matching M of T_c. Let I be a maximal, connected subtree of T_c that does not include an edge of M. Suppose the total weight w of I is less than or equal to the number of edges of M it is incident to. In this case, the state flipping argument described above will go through. (Specifically, flipping the state saves at least w mutations in the tree with lower parsimony score, but cannot decrease the parsimony score of the other tree by more than w.) This state flipping could be applied, for example, in the situation described in Figure 9.

Once all such situations have been excluded, we see that when computing d^2_{MP} we can safely restrict our attention to matchings where every region enclosed between matching edges has total weight strictly larger than the number of incident matching edges.

The following two definitions formalise this idea.

Definition 5. *Let T be a binary phylogenetic tree on X and T_c the core tree of T. Every node of T_c is assigned a weight that is equal to the number of leaves in T it is adjacent to. Note that the weight of v can also be expressed as $3 - \deg(v)$, where $deg(v)$ is the degree of v in T_c.*

A matching M of T_c is called k-legal if for every connected component S of $T_c - M$ with at most k vertices, the total weight of S (i.e., the sum of the weights of all its nodes) is strictly greater than the number of edges in M that are incident to S.

Note that every matching M of T_c is a 0-legal and 1-legal matching because every component in $T_c - M$ has at least two vertices. With this new definition we can translate the definition of legal, as used earlier in the article, to 2-legal. The only way for a matching to not be 2-legal is for it to contain a component shown in Figure 4.

Definition 6. *A matching M of T_c is called fully legal if it is k-legal for every value of k. In other words, for every component S of $T_c - M$, the total weight of S is strictly greater than the number of edges in M that are incident to S. Note that if M is n-legal with n the number of leaves in X, M is automatically fully legal.*

Let S be a component of $T_c - M$ for some matching M, let $s = |S|$ be the number of vertices and m the number of edges in M incident to S. The sum of the degrees of the vertices in S is $2(s-1)+m$, as edges between vertices of S are counted twice, and incident edges of M counted once. Thus the total weight of S is $3s-(2(s-1)+m) = s-m+2$, which allows us to reformulate the definition of k-legality as follows:

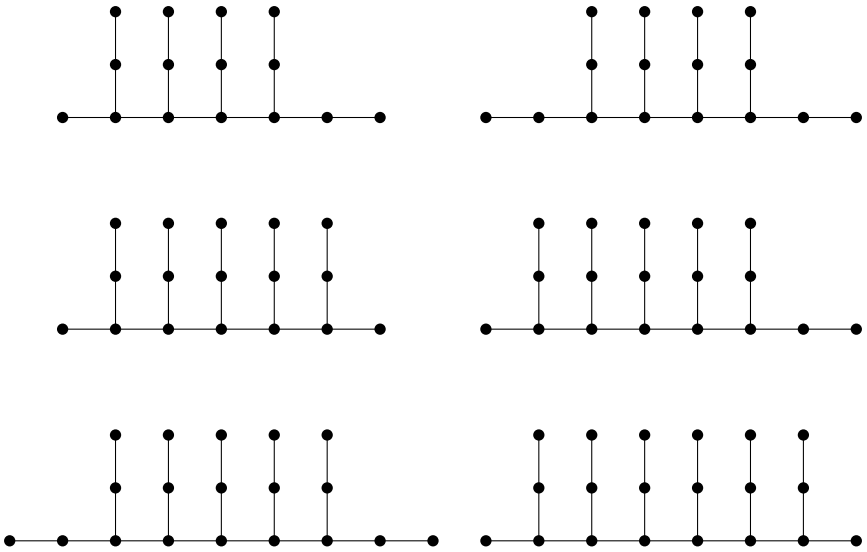
Lemma 8. *A matching M of T_c is k-legal if and only if, for every connected component S of $T_c - M$ with at most k vertices, the number of vertices s and the number of incident edges m satisfy $s > 2m - 2$.*

Theorem 7 only provides an upper bound on the number of 2-legal matchings. How far can this upper bound be lowered if we restrict our attention purely to fully legal matchings?

At this stage we only have partial answers. The greater k, the more complex the problem becomes, and the more auxiliary quantities are required. For 4-legal matchings, one obtains a system of recurrences and in turn a 13-dimensional bilinear map akin to the map B in the proof of Theorem 7. With the help of a computer - we provide a detailed Mathematica file in the supplementary material - one can then use the same method to show that the number of such matchings is $O((19/12)^n)$ in trees with n nodes (this bound is, unlike Theorem 7, probably not sharp). Note here that $19/12 \approx 1.5834 < \alpha \approx 1.5895$ - thus implying a marginally improved $O^*(1.5834^n)$ algorithm for d^2_{MP}. It

follows that the number of fully legal matchings in trees with n nodes is also $O((19/12)^n)$. Further improvements to this bound should be possible by considering larger values of k, but the resulting recursions become quite unwieldy.

Finally, we provide a lower bound. By means of a computer search, one finds that the trees with 15 to 20 nodes and the greatest number of fully legal matchings look as follows:



Let us therefore consider the sequence of "comb-shaped" trees C_k that consist of a "shaft" (a path on k vertices) and k "teeth" (paths of length 2 attached to the vertices of the shaft), and let us estimate the number of fully legal matchings in these trees. The edges of a matching M that belong to the shaft divide the tree C_k into pieces.

Let us count the number of fully legal possibilities for a piece that contains ℓ vertices of the shaft. Assume for now that it is a piece between two matching edges (i.e., not at one of the ends of the comb). For the first and last tooth, we only have two possibilities: there can either be no edge of it included in M, or precisely one, which is not incident to the shaft. For the other teeth, we have three possibilities: no edge, the edge that is incident to the shaft, or the edge that is not incident to the shaft. Suppose that the first possibility occurs a times, the second b times and the last c times. We must clearly have $a+b+c=\ell-2$. The components of $C_k - M$ that are single edges do not affect legality, so we focus on the component that contains part of the shaft. Its total weight is $3a+b+6$, $3a+b+4$, or $3a+b+2$, depending on whether 0, 1 or 2 edges of the first and last tooth are contained in M. On the other hand, the total number of incident matching edges is $b+c+2$, $b+c+3$ or $b+c+4$ respectively. So the legality condition becomes $3a+4>c$, $3a+1>c$ or $3a-2>c$ respectively.

Noting that there are $\binom{\ell-2}{a,b,c}$ ways to distribute the matching edges of M among the different teeth, we eventually find that the number of possibilities for a piece that contains ℓ vertices of the shaft is

$$p(\ell) = \sum_{a+b+c=\ell-2} \binom{a+b+c}{a,b,c} \big([3a+4>c] + 2[3a+1>c] + [3a-2>c]\big),$$

using Iverson's notation: $[P]=1$ if P is true, $[P]=0$ otherwise. This only changes slightly for a piece at either of the two ends, which is only incident to one of the matching edges on the shaft: here, the number becomes

$$q(\ell) = \sum_{a+b+c=\ell-1} \binom{a+b+c}{a,b,c} \big([3a+3>c] + [3a>c]\big).$$

A fully legal matching of C_k can be obtained by putting together pieces to a sequence: if such a matching contains $h \geq 1$ edges of the shaft, then the total number of possibilities is

$$\sum_{\substack{\ell_1+\ell_2+\cdots+\ell_{h+1}=k \\ \ell_1,\ell_{h+1}\geq 1;\, \ell_2,\ldots,\ell_h\geq 2}} q(\ell_1)q(\ell_{h+1}) \prod_{j=2}^{h} p(\ell_j). \tag{5}$$

Here, $\ell_1, \ell_2, \ldots$ represent the lengths of the different pieces. Define the generating functions $P(x) = \sum_{j\geq 2} p(j)x^j$ and $Q(x) = \sum_{j\geq 1} q(j)x^j$. The expression in (5) is precisely the coefficient of x^k in $Q(x)^2 P(x)^{h-1}$. The total number of fully legal matchings with at least one edge on the shaft is therefore the coefficient of x^k in

$$\sum_{h\geq 1} Q(x)^2 P(x)^{h-1} = \frac{Q(x)^2}{1-P(x)}.$$

We ignore fully legal matchings without an edge on the shaft for simplicity as their number is comparatively small: it is trivially $O(3^k)$. The asymptotic growth rate of the coefficients of $\frac{Q(x)^2}{1-P(x)}$ is governed by its singularity that is closest to the origin, which is the unique positive real root of the denominator

$$1 - P(x) = 1 - 3x^2 - 8x^3 - 24x^4 - \cdots,$$

see Chapter IV in [12] for details. Letting $\rho \approx 0.2633$ be that root, we find that the number of fully legal matchings of C_k is $\Theta(\rho^{-k})$. Note that C_k has $n = 3k$ vertices. We have thus constructed a sequence of trees for which the number of fully legal matchings grows as $\Theta(\beta^n)$, with $\beta = \rho^{-1/3} \approx 1.5603$. This provides us with a lower bound on how far the idea of legal matchings can be used to improve the algorithm further.

7 Conclusions and future work

In this article we have given faster exponential-time algorithms for the unrooted and rooted maximum agreement forest problems on two binary phylogenetic trees with n taxa. These new algorithms have running times $O^*(2.2973^n)$ and $O^*(2.0649^n)$ respectively. We achieved this by running existing fixed parameter tractable (FPT) algorithms up to the point that the size of a maximum agreement forest becomes too large, and then switching to the enumeration of convex characters, which project down onto the space of agreement forests. The critical insight here is that the space of (relevant) convex characters shrinks sharply once the size of a maximum agreement forest becomes too large. We expect that a similar technique will work in other situations when these ingredients are all present. It is natural to ask whether the two running times given above can be improved further. In particular, the running time for rooted maximum agreement forests is now tantalisingly close to $O^*(2^n)$; is such a running time achievable?

We have also given an algorithm with running time $O^*(1.5895^n)$ for the maximum parsimony distance problem on two-state characters (d^2_{MP}). To achieve this we established a correspondence between certain convex characters and specially-constrained matchings on an auxiliary tree structure. Once this was established powerful machinery from the enumerative combinatorics literature could be applied to carefully count these matchings; moreover, the bounds are tight. Interestingly, although inspired by phylogenetics, this analysis had the spin-off effect of establishing new, tight upper bounds on the number of matchings in degree-constrained trees. In terms of future work, we notice that two-state convex characters can have many distinct optimal extensions, and thus the mapping from two-state convex characters to general convex characters (via their extensions) is needlessly one-to-many in our analysis; would removing this redundancy help in any way? Finally, we note that there are as yet no non-trivial non-enumerative algorithms for computing d^2_{MP}. It seems reasonable to believe, given that algorithms with running time $O^*(2^n)$ are trivial, that our running time of $O^*(1.5895^n)$ might be achieved without the extensive enumeration deployed in this article. The problem is (algorithmically) not yet so well-understood, a recent kernelization result notwithstanding [8].

8 Acknowledgements

We thank Mark Jones for useful discussions. Ruben Meuwese was supported by the Dutch Research Council (NWO) KLEIN 1 grant *Deep kernelization for phylogenetic discordance*, project number OCENW.KLEIN.305. Stephan Wagner was supported by the Knut and Alice Wallenberg Foundation. We also thank the anonymous reviewers for detailed reading and feedback.

References

[1] B. Allen and M. Steel. "Subtree transfer operations and their induced metrics on evolutionary trees". In: *Annals of Combinatorics* 5.1 (2001), pp. 1–15.

[2] E. Andriantiana and S. Wagner. "On the number of independent subsets in trees with restricted degrees". In: *Mathematical and Computer Modelling* 53.5-6 (2011), pp. 678–683.

[3] E. Bachoore and H. Bodlaender. "Convex recoloring of leaf-colored trees". In: (2006). URL: `http://hdl.handle.net/1874/22179`.

[4] L. Bulteau and M. Weller. "Parameterized algorithms in bioinformatics: An overview". In: *Algorithms* 12.12 (2019), p. 256.

[5] J. Chen, J. Fan, and S. Sze. "Parameterized and approximation algorithms for maximum agreement forest in multifurcating trees". In: *Theoretical Computer Science* 562 (2015), pp. 496–512.

[6] T. Cover and J. Thomas. *Elements of information theory (2. ed.)* Wiley, 2006. ISBN: 978-0-471-24195-9.

[7] M. Cygan et al. *Parameterized algorithms*. 1st. Springer Publishing Company, Incorporated, 2015. ISBN: 978-3-319-21275-3.

[8] E. Deen et al. "A Near-Linear Kernel for Two-Parsimony Distance". In: *arXiv preprint arXiv:2211.00378* (2022).

[9] A. Dress et al. *Basic phylogenetic combinatorics*. Cambridge University Press, 2012.

[10] M. Fischer and S. Kelk. "On the Maximum Parsimony distance between phylogenetic trees". In: *Annals of Combinatorics* 20.1 (2016), pp. 87–113.

[11] W. Fitch. "Toward defining the course of evolution: minimum change for a specific tree topology." In: *Systematic Zoology* 20.4 (1971), pp. 406–416.

[12] P. Flajolet and R. Sedgewick. *Analytic combinatorics*. Cambridge University Press, Cambridge, 2009, pp. xiv+810. ISBN: 978-0-521-89806-5.

[13] F. Fomin and D. Kratsch. *Exact exponential algorithms*. Springer, 2010.

[14] F. Fomin et al. *Kernelization: Theory of parameterized preprocessing*. Cambridge University Press, 2019.

[15] J. Hartigan. "Minimum mutation fits to a given tree". In: *Biometrics* (1973), pp. 53–65.

[16] J. Hein et al. "On the complexity of comparing evolutionary trees". In: *Discrete Applied Mathematics* 71.1-3 (1996), pp. 153–169.

[17] M. Jones, S. Kelk, and L. Stougie. "Maximum parsimony distance on phylogenetic trees: A linear kernel and constant factor approximation algorithm". In: *Journal of Computer and System Sciences* 117 (2021), pp. 165–181.

[18] S. Kelk and M Fischer. "On the complexity of computing MP distance between binary phylogenetic trees". In: *Annals of Combinatorics* 21.4 (2017), pp. 573–604.

[19] S. Kelk and R. Meuwese. "Sharp upper and lower bounds on a restricted class of convex characters". In: *arXiv preprint arXiv:2107.10871* (2021).

[20] S. Kelk and G. Stamoulis. "A note on convex characters, Fibonacci numbers and exponential-time algorithms". In: *Advances in Applied Mathematics* 84 (2017), pp. 34–46.

[21] S. Kelk et al. "Phylogenetic incongruence through the lens of Monadic Second Order logic". In: *Journal of Graph Algorithms and Applications* 20.2 (2016), pp. 189–215.

[22] V. Moulton and T. Wu. "A parsimony-based metric for phylogenetic trees". In: *Advances in Applied Mathematics* 66 (2015), pp. 22–45.

[23] M. Rosenfeld. "The growth rate over trees of any family of sets defined by a monadic second order formula is semi-computable". In: *Proceedings of the 2021 ACM-SIAM Symposium on Discrete Algorithms, SODA 2021, Virtual Conference, January 10 - 13, 2021*. Ed. by Dániel Marx. SIAM, 2021, pp. 776–795.

[24] G. Rote. "The maximum number of minimal dominating sets in a tree". In: *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2019, San Diego, California, USA, January 6-9, 2019*. Ed. by Timothy M. Chan. SIAM, 2019, pp. 1201–1214.

[25] C. Semple and M. Steel. *Phylogenetics*. 2003.

[26] M. Steel. "The complexity of reconstructing trees from qualitative characters and subtrees". In: *Journal of Classification* 9.1 (1992), pp. 91–116.

[27] S. Wagner and I. Gutman. "Maxima and minima of the Hosoya index and the Merrifield-Simmons index". In: *Acta Applicandae Mathematicae* 112.3 (2010), pp. 323–346.

[28] R. van Wersch et al. "Reflections on kernelizing and computing unrooted agreement forests". In: *To appear in Annals of Operations Research, arXiv preprint arXiv:2012.07354* (2021).

[29] C. Whidden, R. Beiko, and N. Zeh. "Fixed-parameter algorithms for maximum agreement forests". In: *SIAM Journal on Computing* 42.4 (2013), pp. 1431–1466.

Steven Kelk, Ruben Meuwese, Department of Data Science and Knowledge Engineering (DKE), Maastricht University, P.O. Box 616, 6200 MD Maastricht, The Netherlands

E-mail address, S. Kelk: steven.kelk@maastrichtuniversity.nl

E-mail address, R. Meuwese: r.meuwese@maastrichtuniversity.nl

Stephan Wagner, Department of Mathematics, Uppsala Universitet, P.O. Box 480, 751 06 Uppsala, Sweden

E-mail address, S. Wagner: stephan.wagner@math.uu.se

Appendix A: Fitch's algorithm

Fitch's algorithm [11], which computes $l_f(T)$ and a corresponding optimal extension g of f to T, has two phases. In the first phase, known as the *bottom-up* phase, we start by assigning to each taxon a subset of states consisting of only the state it is assigned by f. The internal nodes of T are assigned subsets of states recursively, as follows. Suppose a node p has two children u and v, and the bottom-up phase has already assigned subsets $F(u)$ and $F(v)$ to the two children, respectively. If $F(u) \cap F(v) \neq \emptyset$ then set $F(p) = F(u) \cap F(v)$ (in which case we say that p is an *intersection* node). If $F(u) \cap F(v) = \emptyset$ then set $F(p) = F(u) \cup F(v)$ (in which case we say that p is a *union* node). The number of union nodes in the bottom-up phase is equal to $l_f(T)$. To actually create an optimal extension g of f to T, we require the *top-down* phase of Fitch's algorithm. Start at the root r and let $g(r)$ be an arbitrary element in $F(r)$. For an internal node u with parent p, we set $g(u) = g(p)$ (if $g(p) \in F(u)$) and otherwise (i.e. $g(p) \notin F(u)$) set $g(u)$ to be an arbitrary element of $F(u)$. Note that the arbitrary choices in the top-down phase can be used to generate multiple distinct optimal extensions.

Appendix B

The vectors of the set $\mathcal{S}$, with $R = 2793745$:

$$
\begin{aligned}
\mathbf{v}_1 &= \alpha^{-2}[1,0,0,0,0]^T,\\
\mathbf{v}_2 &= \alpha^{-6}\Big[2+\frac{3446}{\sqrt{R}}, 2+\frac{3170}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_3 &= \alpha^{-6}[4,4,0,0,0]^T,\\
\mathbf{v}_4 &= \alpha^{-21}\Big[2016+\frac{3380832}{\sqrt{R}}, 2208+\frac{3671904}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_5 &= \alpha^{-21}[4032,4416,0,0,0]^T,\\
\mathbf{v}_6 &= \alpha^{-21}\Big[2016+\frac{3367584}{\sqrt{R}}, 2208+\frac{3693984}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_7 &= \alpha^{-21}\Big[\frac{5628705696}{R}+\frac{3367584}{\sqrt{R}}, \frac{6174294432}{R}+\frac{3693984}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_8 &= \alpha^{-14}\Big[\frac{1969470208}{9R}+\frac{3291861361024}{9R^{3/2}}, \frac{2179611976}{9R}+\frac{3643134552760}{9R^{3/2}}, 0,0,0\Big]^T,\\
\mathbf{v}_9 &= \alpha^{-10}\Big[12+\frac{20676}{\sqrt{R}}, 14+\frac{22466}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{10} &= \alpha^{-3}\Big[\frac{143768593}{108R}+\frac{85015}{108\sqrt{R}}, \frac{105898423}{72R}+\frac{66145}{72\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{11} &= \alpha^{-3}\Big[-\frac{1625}{108}+\frac{2887105}{108\sqrt{R}}, \frac{1715}{72}-\frac{2737003}{72\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{12} &= \alpha^{-3}\Big[\frac{108745}{108}-\frac{65\sqrt{R}}{108}, \frac{28441}{24}-\frac{17\sqrt{R}}{24}, 0,0,0\Big]^T,\\
\mathbf{v}_{13} &= \alpha^{-3}\Big[\frac{1}{2}+\frac{1447}{2\sqrt{R}}, \frac{1}{2}+\frac{1999}{2\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{14} &= \alpha^{-14}\Big[\frac{201818664}{R}+\frac{120744}{\sqrt{R}}, \frac{267240144}{R}+\frac{159888}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{15} &= \alpha^{-14}\Big[72+\frac{120744}{\sqrt{R}}, 96+\frac{159888}{\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{16} &= \alpha^{-14}[144,192,0,0,0]^T,\\
\mathbf{v}_{17} &= \alpha^{-7}\Big[\frac{31}{18}+\frac{116617}{18\sqrt{R}}, \frac{281}{54}+\frac{208991}{54\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{18} &= \alpha^{-7}\Big[\frac{108745}{18}-\frac{65\sqrt{R}}{18}, \frac{219163}{27}-\frac{131\sqrt{R}}{27}, 0,0,0\Big]^T,\\
\mathbf{v}_{19} &= \Big[\frac{8142156817}{23328R}+\frac{3615127}{23328\sqrt{R}}, \frac{2689931479}{11664R}+\frac{4131265}{11664\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{20} &= \Big[-\frac{2104505}{23328}+\frac{3526059745}{23328\sqrt{R}}, \frac{2807237}{23328}-\frac{4680672973}{23328\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{21} &= \Big[\frac{11814523825}{23328}-\frac{7068425\sqrt{R}}{23328}, \frac{1999380955}{2916}-\frac{1196195\sqrt{R}}{2916}, 0,0,0\Big]^T,\\
\mathbf{v}_{22} &= \Big[\frac{7345}{432}-\frac{12119705}{432\sqrt{R}}, -\frac{6197}{288}+\frac{10499773}{288\sqrt{R}}, 0,0,0\Big]^T,\\
\mathbf{v}_{23} &= \Big[\frac{2443777}{8R}+\frac{1447}{8\sqrt{R}}, \frac{3242521}{8R}+\frac{1999}{8\sqrt{R}}, 0,0,0\Big]^T,
\end{aligned}
$$

$$\mathbf{v}_{24} = \Big[\frac{32805161}{108R} + \frac{54881040239}{108R^{3/2}}, \frac{29641217}{72R} + \frac{49498725911}{72R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{25} = \Big[\frac{1237818669070513}{1458R^2} + \frac{740509100311}{1458R^{3/2}}, \frac{838121265457801}{729R^2} + \frac{501401051935}{729R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{26} = \alpha^{-15}\Big[\frac{71754261114955960}{81R^2} + \frac{119933657317951854472}{81R^{5/2}},$$
$$\frac{291727305240092752}{243R^2} + \frac{487607592266798252080}{243R^{5/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{27} = \alpha^{-8}\Big[\frac{1407252633280527321143 93}{1458R^3} + \frac{84193524123331244303}{1458R^{5/2}},$$
$$\frac{2862839555284102265954 27}{2187R^3} + \frac{171278806193160289301}{2187R^{5/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{28} = \alpha^{-4}\Big[\frac{10512907255001141}{1944R^2} + \frac{6289683368723}{1944R^{3/2}}, \frac{10709134162880129}{1458R^2} + \frac{6407069939495}{1458R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{29} = \Big[\frac{794956577}{2592R} + \frac{464135}{2592\sqrt{R}}, \frac{524137043}{1296R} + \frac{325541}{1296\sqrt{R}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{30} = \alpha^{-15}\Big[\frac{17072730067}{54R} + \frac{28535661518197}{54R^{3/2}}, \frac{3873220609}{9R} + \frac{6473856199063}{9R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{31} = \alpha^{-8}\Big[\frac{803579524335268753}{23328R^2} + \frac{480751987830295}{23328R^{3/2}},$$
$$\frac{547329217239002015}{11664R^2} + \frac{327434508313145}{11664R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{32} = \alpha^{-8}\Big[\frac{293738136445}{23328R} + \frac{470526609087163}{23328R^{3/2}}, \frac{197305889945}{11664R} + \frac{325111894094399}{11664R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{33} = \alpha^{-8}\Big[\frac{8133848923273}{23328R} - \frac{4522191905}{23328\sqrt{R}}, -\frac{7653084333757}{11664R} + \frac{4813128245}{11664\sqrt{R}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{34} = \alpha^{-2}\Big[\frac{34964309}{54R} + \frac{21179}{54\sqrt{R}}, \frac{29330411}{27R} + \frac{17753}{27\sqrt{R}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{35} = \alpha^{-17}\Big[\frac{2039517464}{3R} + \frac{3408952424936}{3R^{3/2}}, \frac{10267634576}{9R} + \frac{17161853188592}{9R^{3/2}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{36} = \alpha^{-13}\Big[38 + \frac{63818}{\sqrt{R}}, 64 + \frac{106960}{\sqrt{R}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{37} = \alpha^{-6}\Big[\frac{892769641}{216R} + \frac{538495}{216\sqrt{R}}, \frac{62416754}{9R} + \frac{37790}{9\sqrt{R}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{38} = \alpha^{-6}\Big[-\frac{1355}{216} + \frac{3337411}{216\sqrt{R}}, \frac{5}{2} + \frac{8339}{2\sqrt{R}}, 0, 0, 0\Big]^T,$$
$$\mathbf{v}_{39} = \alpha^{-4}[1, 2, 0, 0, 0]^T,$$
$$\mathbf{v}_{40} = \alpha^{-4}\Big[\frac{1895}{216} - \frac{2436799}{216\sqrt{R}}, 0, 0, -\frac{1625}{108} + \frac{2887105}{108\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{41} = \alpha^{-4}\Big[\frac{605232455}{216R} + \frac{368465}{216\sqrt{R}}, 0, 0, \frac{143768593}{108R} + \frac{85015}{108\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{42} = \alpha^{-11}\Big[26 + \frac{43142}{\sqrt{R}}, 0, 0, 12 + \frac{20676}{\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{43} = \alpha^{-15}\Big[\frac{4149082184}{9R} + \frac{6934995913784}{9R^{3/2}}, 0, 0, \frac{1969470208}{9R} + \frac{3291861361024}{9R^{3/2}}, 0\Big]^T,$$
$$\mathbf{v}_{44} = \Big[\frac{23696513}{54R} + \frac{14327}{54\sqrt{R}}, 0, 0, \frac{1877966}{9R} + \frac{1142}{9\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{45} = \Big[\frac{1}{8} + \frac{2551}{8\sqrt{R}}, 0, 0, \frac{1}{8} + \frac{343}{8\sqrt{R}}, 0\Big]^T,$$

$$\mathbf{v}_{46} = \Big[\frac{18403}{54} - \frac{11\sqrt{R}}{54}, 0, 0, \frac{11711}{72} - \frac{7\sqrt{R}}{72}, 0\Big]^T,$$
$$\mathbf{v}_{47} = \Big[\frac{73}{216} - \frac{8081}{216\sqrt{R}}, 0, 0, -\frac{7}{36} + \frac{20783}{36\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{48} = \alpha^{-7}[8, 0, 0, 4, 0]^T,$$
$$\mathbf{v}_{49} = \alpha^{-7}\Big[4 + \frac{6616}{\sqrt{R}}, 0, 0, 2 + \frac{3446}{\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{50} = \alpha^{-3}[1, 0, 0, 1, 0]^T,$$
$$\mathbf{v}_{51} = \alpha^{-5}\Big[\frac{1895}{216} - \frac{2436799}{216\sqrt{R}}, 0, -\frac{1625}{108} + \frac{2887105}{108\sqrt{R}}, \frac{1895}{216} - \frac{2436799}{216\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{52} = \alpha^{-5}\Big[\frac{605232455}{216R} + \frac{368465}{216\sqrt{R}}, 0, \frac{143768593}{108R} + \frac{85015}{108\sqrt{R}}, \frac{605232455}{216R} + \frac{368465}{216\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{53} = \alpha^{-12}\Big[26 + \frac{43142}{\sqrt{R}}, 0, 12 + \frac{20676}{\sqrt{R}}, 26 + \frac{43142}{\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{54} = \alpha^{-16}\Big[\frac{4149082184}{9R} + \frac{6934995913784}{9R^{3/2}}, 0, \frac{1969470208}{9R} + \frac{3291861361024}{9R^{3/2}},$$
$$\frac{4149082184}{9R} + \frac{6934995913784}{9R^{3/2}}, 0\Big]^T,$$
$$\mathbf{v}_{55} = \alpha^{-1}\Big[\frac{23696513}{54R} + \frac{14327}{54\sqrt{R}}, 0, \frac{1877966}{9R} + \frac{1142}{9\sqrt{R}}, \frac{23696513}{54R} + \frac{14327}{54\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{56} = \alpha^{-1}\Big[\frac{1}{8} + \frac{2551}{8\sqrt{R}}, 0, \frac{1}{8} + \frac{343}{8\sqrt{R}}, \frac{1}{8} + \frac{2551}{8\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{57} = \alpha^{-1}\Big[\frac{18403}{54} - \frac{11\sqrt{R}}{54}, 0, \frac{11711}{72} - \frac{7\sqrt{R}}{72}, \frac{18403}{54} - \frac{11\sqrt{R}}{54}, 0\Big]^T$$
$$\mathbf{v}_{58} = \alpha^{-1}\Big[\frac{73}{216} - \frac{8081}{216\sqrt{R}}, 0, -\frac{7}{36} + \frac{20783}{36\sqrt{R}}, \frac{73}{216} - \frac{8081}{216\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{59} = \alpha^{-8}[8, 0, 4, 8, 0]^T,$$
$$\mathbf{v}_{60} = \alpha^{-8}\Big[4 + \frac{6616}{\sqrt{R}}, 0, 2 + \frac{3446}{\sqrt{R}}, 4 + \frac{6616}{\sqrt{R}}, 0\Big]^T,$$
$$\mathbf{v}_{61} = \alpha^{-4}[1, 0, 1, 1, 0]^T,$$
$$\mathbf{v}_{62} = \alpha^{-1}[0, 0, 0, 0, 1]^T.$$

The set S, consisting of vectors $[v, w, x, y, z]^T$, can be split into 4 groups:

1. Vectors 1 to 39 have $x = y = z = 0$ and the rest is positive.
2. Vectors 40 to 50 have $w = x = z = 0$ and the rest is positive.
3. Vectors 51 to 61 have $w = z = 0$ and $v = y$ and the rest is positive.
4. Vector 62 has $v = w = x = y = 0$ and $z > 0$.

Now, if we look at $B(a, b)$ we can observe that we can split the image into 4 groups. When we say "combine" two vectors a and b we mean look at the result of $B(a, b)$.

i Vector combinations from every group except vector 62 will result in a vector that has $x = y = z = 0$. This is because $z = 0$ in all the vectors we combine.

ii Combining vector 62 with group 1 above will result in a vector that has $w = x = z = 0$.

iii Combining vector 62 with group 2 above will result in a vector that has $w = z = 0$ and $v = y$ and the rest is positive.

iv Combining vector 62 with group 3 above will result in a vector that has $w = z = 0$ and $x = y$ and the rest is positive.

These observations will make checking that S has the desired property much easier; we provide further details in the supplementary Mathematica files: `https://github.com/skelk2001/legal-matchings`

Appendix C

Let M_n denote the maximum number of legal matchings in a tree with n nodes. In the following, we determine a lower bound on the rate of growth of M_n. Recall that a *good* tree is a tree with maximum degree 3, where no two degree 2 nodes are adjacent.

Lemma 9. *$M_n = \Omega(\alpha^n)$ with $\alpha = (13384 + 8\sqrt{2793745})^{1/22} \approx 1.5895$. This lower bound is achieved on good trees.*

Proof. We provide an explicit construction. To this end, let us introduce some notation. For a rooted tree T with root r, let $z(T)$ be the total number of matchings of T and $z_0(T)$ the number of matchings where the root is not covered (i.e., the root is not an end of one of the matching edges).

Now let us construct a sequence of rooted trees T_k with a repeating pattern of length 22. The starting point can be any good rooted tree T_0; then all further trees in the sequence are good trees as well.

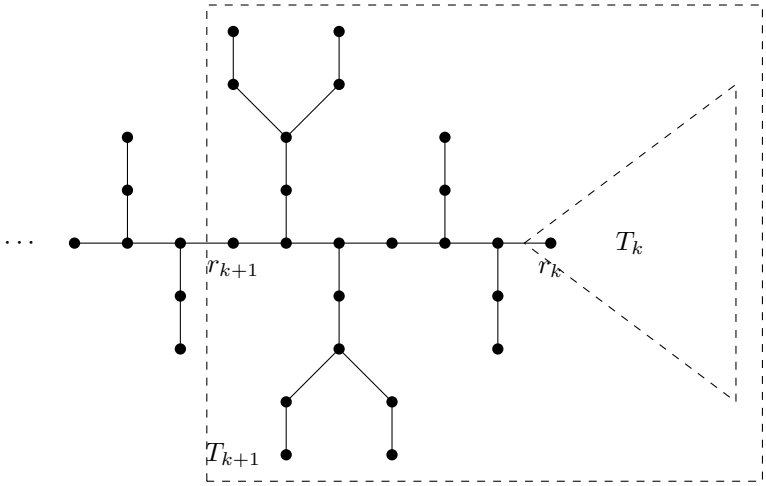


Figure 10: Construction of a sequence of trees with many legal matchings.

The tree T_{k+1} is constructed by attaching a piece of 22 nodes (and 22 edges) to the root of T_k and assigning a new root r_{k+1} as indicated in Figure 10. Now we can determine a recursion for $z(T_{k+1})$ and $z_0(T_{k+1})$ in terms of $z(T_k)$ and $z_0(T_k)$. There are 10144 matchings of the 22 new edges that cover r_k, and 19888 matchings that do not cover it. The former can be extended with any matching of T_k that does not cover r_k, the latter with an arbitrary matching of T_k. Hence we have

$$z(T_{k+1}) = 19888 z(T_k) + 10144 z_0(T_k).$$

Likewise, considering only matchings that leave r_{k+1} uncovered, we obtain

$$z_0(T_{k+1}) = 13456 z(T_k) + 6880 z_0(T_k).$$

Hence we have

$$\begin{bmatrix} z(T_k) \\ z_0(T_k) \end{bmatrix} = \begin{bmatrix} 19888 & 10144 \\ 13456 & 6880 \end{bmatrix}^k \begin{bmatrix} z(T_0) \\ z_0(T_0) \end{bmatrix}.$$

Since the eigenvalues of the 2×2-matrix are $13384 \pm 8\sqrt{2793745}$, it follows that $z(T_k) = \Theta\big((13384 + 8\sqrt{2793745})^k\big)$. Note that T_k has $n = 22k + O(1)$ nodes, and that we can cover all possible congruence classes modulo 22 by choosing T_0 accordingly. Hence we have $M_n = \Omega(\alpha^n)$ with $\alpha = (13384 + 8\sqrt{2793745})^{1/22} \approx 1.5895$. □