Online seminar on Algorithms and Complexity in Phylogenetics

# Sharp upper and lower bounds on a restricted class of convex characters -- and algorithmic applications!

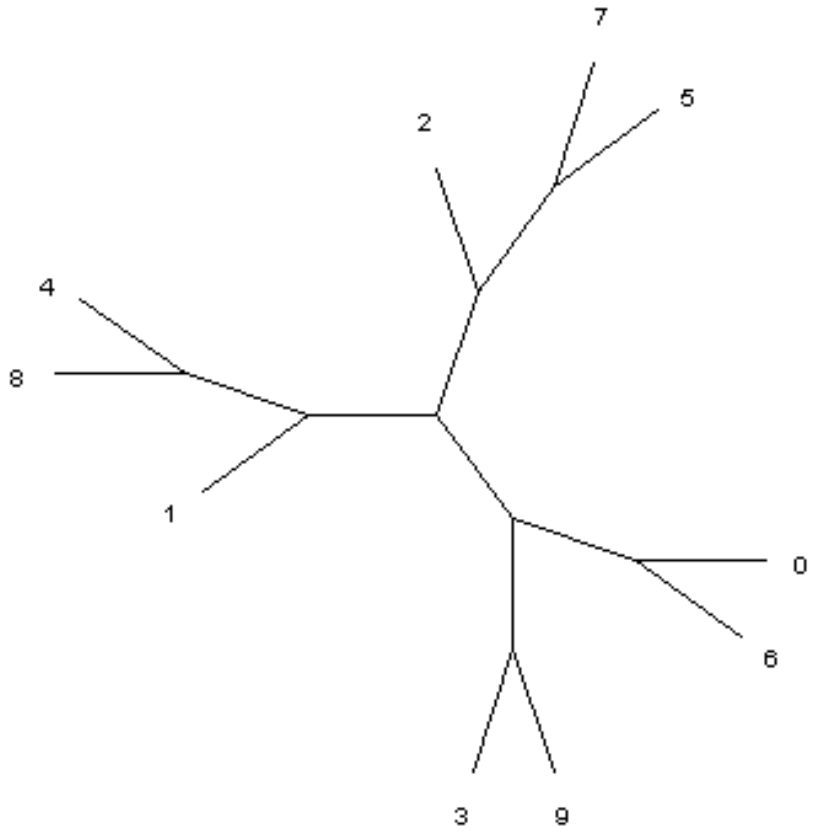
Steven Kelk

Department of Advanced Computing Sciences (DACS)
Maastricht University, Netherlands

Based on joint work with Ruben Meuwese (DACS)

In: Electronic Journal of Combinatorics 29(1) (2022)

- Phylogenetic trees summarise the evolution of a set of species *X.*

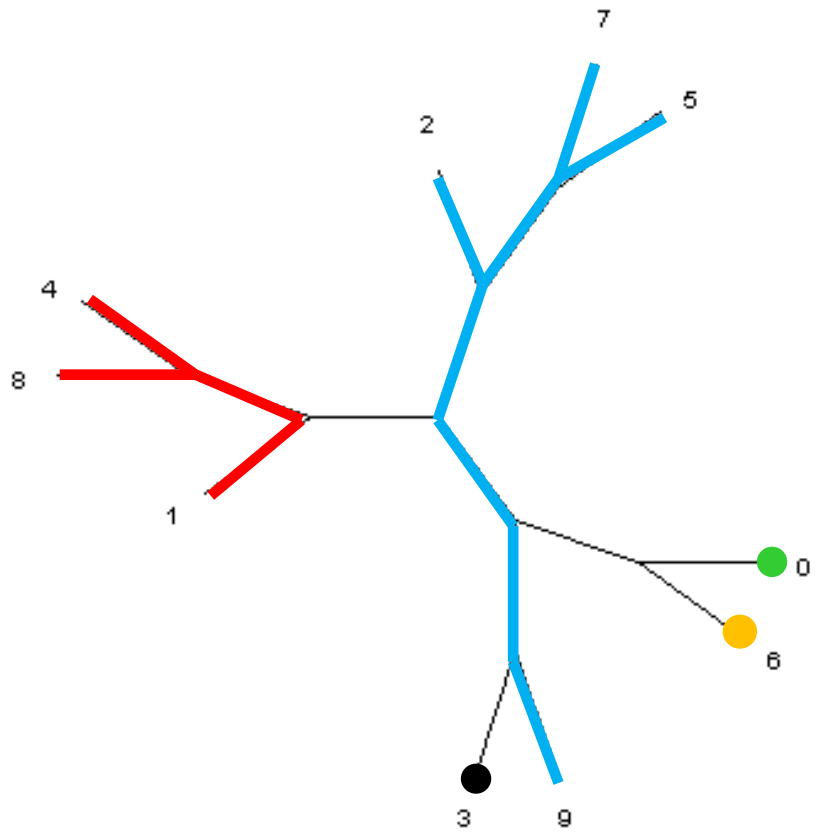


• Let T be an unrooted (binary) phylogenetic tree on X.

• A *character f* is simply a partition of *X*.

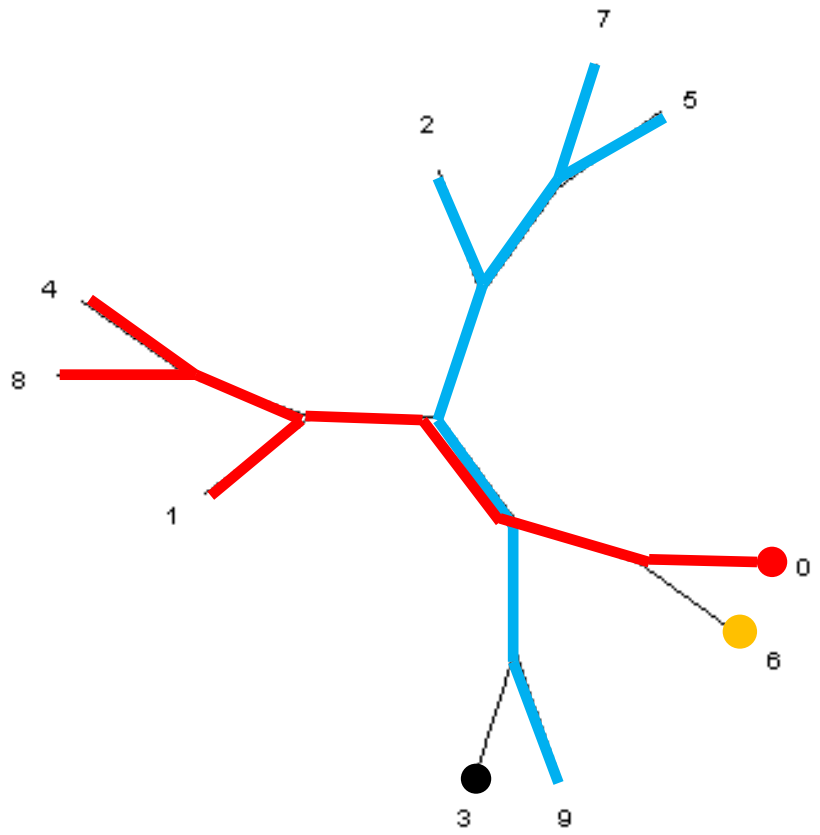
• The blocks of *f* are called *states.*

• A character is *convex* on *T*, if the spanning trees induced by the states – one spanning tree per state - are vertex disjoint in *T*.



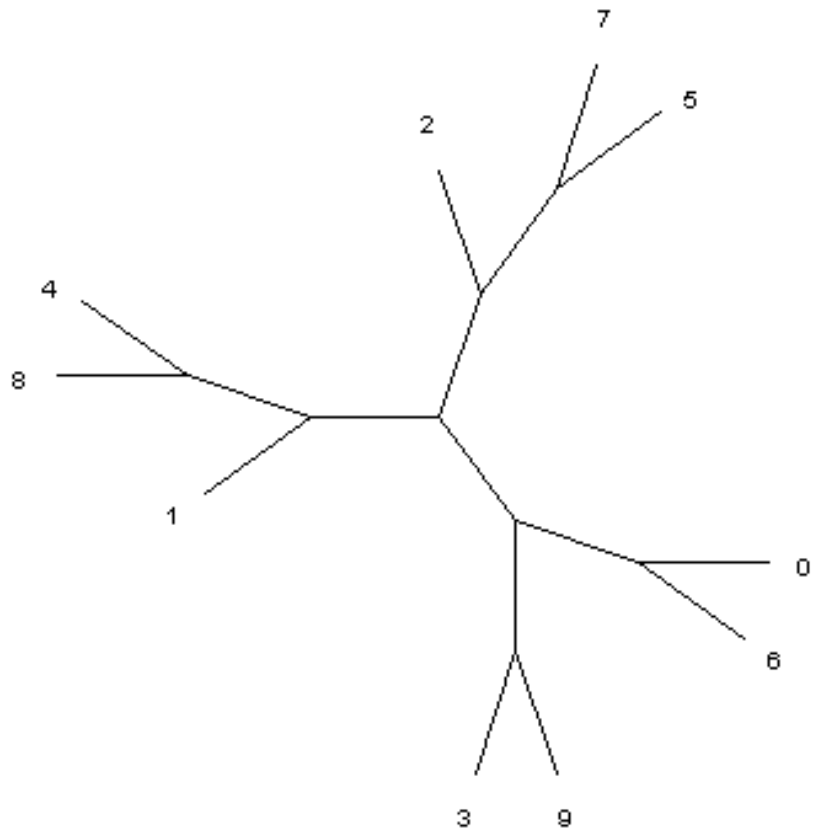

*T*

Convex character { {1,8,4}, {3}, {9,2,7,5}, {0}, {6} } on *T*
- spanning trees are disjoint!




*T*

Non-convex character { {1,8,4,0}, {3}, {9,2,7,5}, {6} } on *T*
- spanning trees are not disjoint




*T*

How many convex characters can a tree have?

**What we already knew (K. & Stamoulis 2017, based on Steel 1992):**

• The number of convex characters on an unrooted binary tree *T* with *n* leaves is independent of the topology of *T,* and is equal to the *(2n-1)*th Fibonacci number. This is equal to:

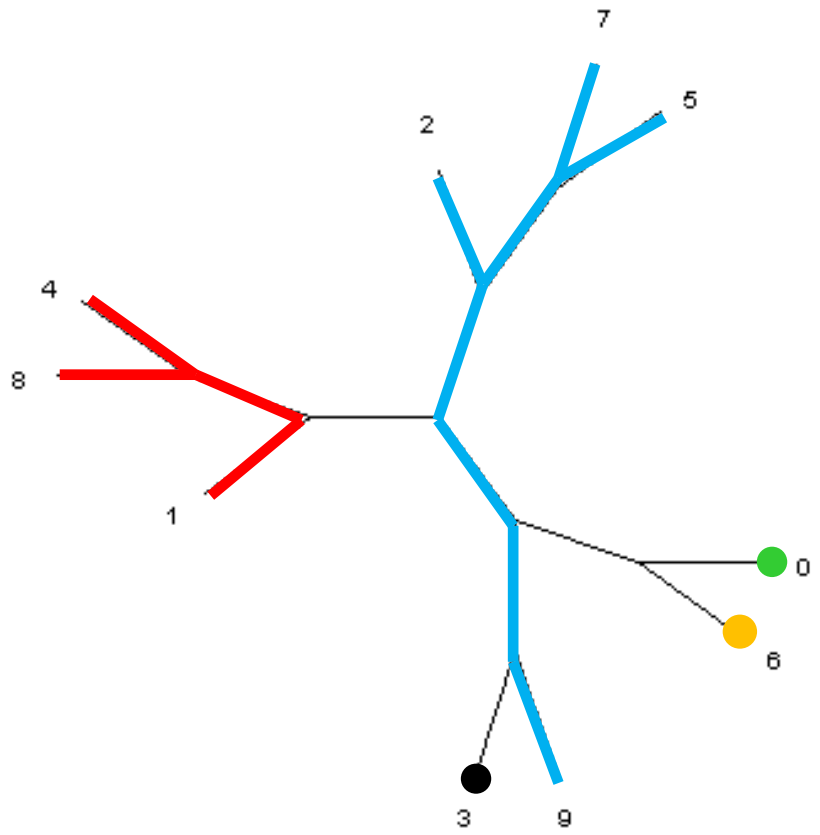
$$\left\lfloor \frac{\phi^{2n-1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor = \Theta(1.681^{2n}) = \Theta(2.681^{n})$$

**What we already knew (K. & Stamoulis 2017):**

• What about convex characters where each state contains **at least 2** elements of $X$ (i.e. singleton states are not allowed)?

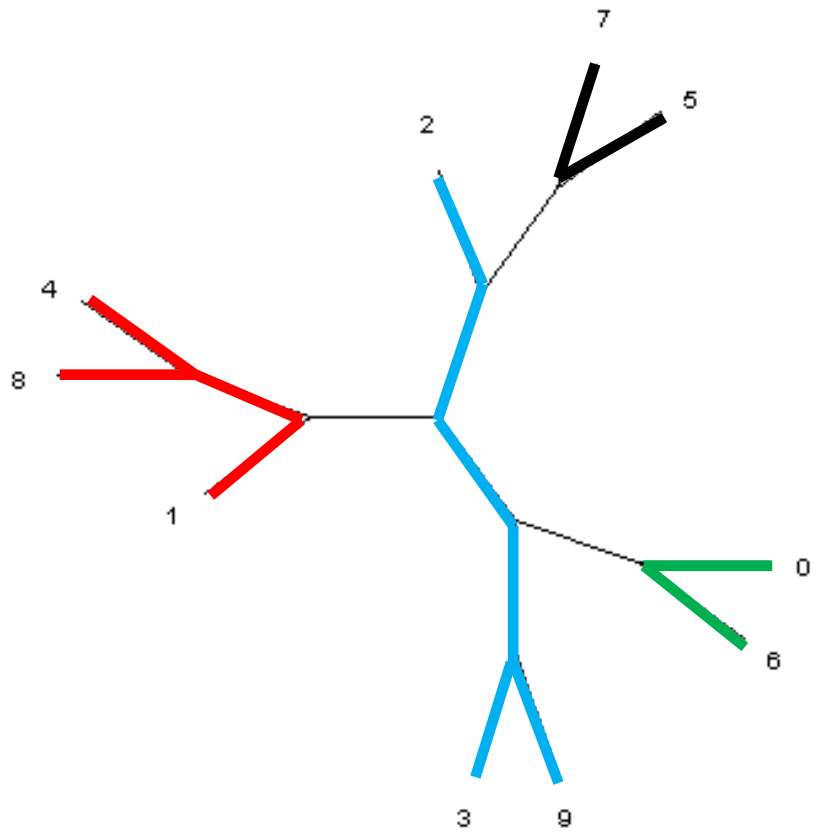
• Let us denote the number of such characters $g_2(T)$, and define $g_k(T)$ similarly.

• Note that $g_1(T)$ is just the set of all convex characters.




*T*

This is **not** a valid $g_2$ character, because of the states {0}, {3} and {6}.




*T*

This **is** a valid $g_2$ character:
{1,4,8}, {5,7}, {2,3,9}, {0,6}

**What we already knew (K. & Stamoulis 2017):**

• The number of $g_2$ convex characters on an unrooted binary tree *T* with *n* leaves is independent of the topology of *T,* and is equal to the *(n-1)*th Fibonacci number. This is equal to:

$$g_2(n) = \left\lfloor \frac{\phi^{n-1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor = \Theta(1.681^n)$$

**What we already knew (K. & Stamoulis 2017):**

• The number of $g_2$ convex characters on an unrooted binary tree $T$ with $n$ leaves is independent of the topology of $T$, and is equal to the $(n-1)$th Fibonacci number. This is equal to:

$$g_2(n) = \left\lfloor \frac{\phi^{n-1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor = \Theta(1.681^n)$$

• However, $g_3$ is topology dependent ☹

• This raised the following question. For each $k \geq 3$,
  • What is the maximum value that $g_k$ can attain on $n$ leaves?
  • What is the minimum value that $g_k$ can attain on $n$ leaves?

## Our new results (1): the maximum

- Let $Cat_n$ be the unrooted caterpillar topology on $n$ leaves. (Note that the actual leaf labels are not important in this work).

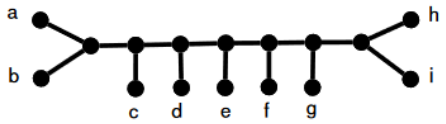



Figure 2: A caterpillar tree on 9 taxa.

## Our new results (1): the maximum

- Let $Cat_n$ be the unrooted caterpillar topology on $n$ leaves. (Note that the actual leaf labels are not important in this work).

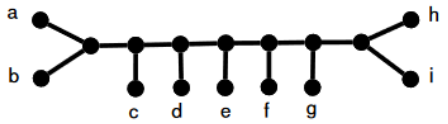



Figure 2: A caterpillar tree on 9 taxa.

**Corollary 7.** *For every $n$, the maximum value of $g_k$ ranging over all trees on $n$ taxa is $g_k(Cat_n)$, which is $\Theta(\alpha^n)$, where $\alpha$ is the positive real root of the characteristic polynomial $x^k - x^{k-1} - 1$.*

## Our new results (1): the maximum

- Let $Cat_n$ be the unrooted caterpillar topology on $n$ leaves. (Note that the actual leaf labels are not important in this work).

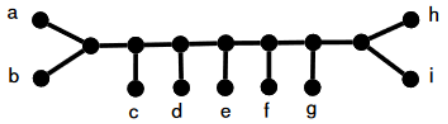



Figure 2: A caterpillar tree on 9 taxa.

**Corollary 7.** *For every $n$, the maximum value of $g_k$ ranging over all trees on $n$ taxa is $g_k(Cat_n)$, which is $\Theta(\alpha^n)$, where $\alpha$ is the positive real root of the characteristic polynomial $x^k - x^{k-1} - 1$.*

- I will explain later where this comes from.

## Our new results (2): the minimum

**Theorem 14.** *Let $n \geqslant k$. Every fully $k$-loaded tree $\mathcal{T}$ on $n$ taxa is a minimizer for $g_k$.*

**Corollary 15.** *For $n \geqslant k \geqslant 2$, the minimum value of $g_k$ ranging over all trees on $n$ taxa is exactly*

$$\left\lfloor \frac{\phi^{\lceil \frac{n}{k-1} \rceil - 1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor.$$

• A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).

• More about this later, too. Let’s first discuss algorithmic significance.

## Algorithmic significance:

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 1: Each entry $\alpha$ indicates that the minimum (respectively, maximum) value of $g_k(\mathcal{T})$, ranging over all trees on $n$ taxa, grows at the rate $\Theta(\alpha^n)$.

## Algorithmic significance:

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 1: Each entry $\alpha$ indicates that the minimum (respectively, maximum) value of $g_k(\mathcal{T})$, ranging over all trees on $n$ taxa, grows at the rate $\Theta(\alpha^n)$.

• From the results of K. & Stamoulis (2017) we can efficiently count, list and uniformly sample these characters.

• So you can loop through all $g_k$ characters in time $\Theta(g_k(T) \cdot poly(n))$ – and the constants hidden by the $\Theta$ notation are small.

## Algorithmic significance:

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

• From the results of K. & Stamoulis (2017) we can efficiently count, list and uniformly sample these characters.

• So you can loop through all $g_k$ characters in time $\Theta(g_k(T) \cdot poly(n))$ – and the constants hidden by the $\Theta$ notation are small.

## Algorithmic significance:

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- By the standards of exponential-time algorithms, this scales quite well.
- So what?

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

- For a number of (NP-hard) phylogenetic optimization problems, an optimal solution can be projected onto some convex character.

- If you find that convex character, you can map backwards to find the original optimal solution → rapid prototyping of algorithms simply by looping through convex characters. Some examples:

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

- **Q: Given a set of trees on the same taxa *X*, can you partition *X* into size-4 subsets such that in each tree the induced quartets are disjoint?**

- I don't actually know whether this is NP-hard or not. Probably it is. But in any case I can build an algorithm by looping through all $g_4$ characters in one of the trees and checking whether any of them gives a valid solution.

## Algorithmic significance: "convex character programming"

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given a set of trees on the same taxa *X*, can you partition *X* into size-4 subsets such that in each tree the induced quartets are disjoint?**

- I don't actually know whether this is NP-hard or not. Probably it is. But in any case I can build an algorithm by looping through all $g_4$ characters in one of the trees and checking whether any of them gives a valid solution.

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given a set of trees on the same taxa *X*, can you partition *X* into size-4 subsets such that in each tree the induced quartets are disjoint?**

- Can easily be generalised to "and the quartets have the same topology", "such that in at least one of trees the quartets are disjoint" and so on.

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given two trees on the same set of taxa, can you find character that maximizes the absolute difference in parsimony scores of that character on the two trees?**

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given two trees on the same set of taxa, can you find character that maximizes the absolute difference in parsimony scores of that character on the two trees?**

- It can be proven that $g_2$ contains such an optimal character!

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given two trees on the same set of taxa, can you find character that maximizes the absolute difference in parsimony scores of that character on the two trees?**

- It can be proven that $g_2$ contains such an optimal character! Both listing and (especially) uniformly sampling work surprisingly well.

## Algorithmic significance: "convex character programming"

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given two trees on the same set of taxa, can you find character that maximizes the absolute difference in parsimony scores of that character on the two trees?**

- This has been leveraged to produce very strong lower bounds on the TBR distance between large trees (Wersch, K., Linz and Stamoulis 2022).

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given a set of trees, find me a *maximum agreement forest* of the trees in which each component has at least *k* taxa.**

- Agreement forests are convex characters in which, for each state, all trees induce the same topology on that state.

## Algorithmic significance: "convex character programming"

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

- **Q: Given a set of trees, find me a *maximum agreement forest* of the trees in which each component has at least *k* taxa.**

- Loop through all $g_k$ characters one of the trees. Can be easily generalized. Yields an easy $O(2.618^n)$ algorithm for the classical problem (*k=1*).

## Algorithmic significance: "convex character programming"

| | *minimum* | *maximum* |
|---|---|---|
| $g_1$ | 2.618 | 2.618 |
| $g_2$ | 1.618 | 1.618 |
| $g_3$ | 1.272 | 1.466 |
| $g_4$ | 1.174 | 1.380 |
| $g_5$ | 1.128 | 1.325 |
| $g_6$ | 1.101 | 1.285 |

| | *caterpillar* | | | *random trees* | | |
|---|---|---|---|---|---|---|
| | 1s | 10s | 100s | 1s | 10s | 100s |
| $g_1$ | 14 | 16 | 19 | 14 | 16 | 19 |
| $g_2$ | 27 | 32 | 37 | 27 | 32 | 37 |
| $g_3$ | 34 | 41 | 47 | 38 | 49 | 55 |
| $g_4$ | 40 | 48 | 56 | 56 | 66 | 74 |
| $g_5$ | 47 | 56 | 64 | 73 | 84 | 96 |
| $g_6$ | 52 | 63 | 72 | 83 | 101 | 116 |

Table 2: The numbers in row $g_k$ indicate the largest $n$ for which, on caterpillar (respectively, random) trees with $n$ taxa, all $g_k$ characters could be listed within $\{1, 10, 100\}$ seconds.

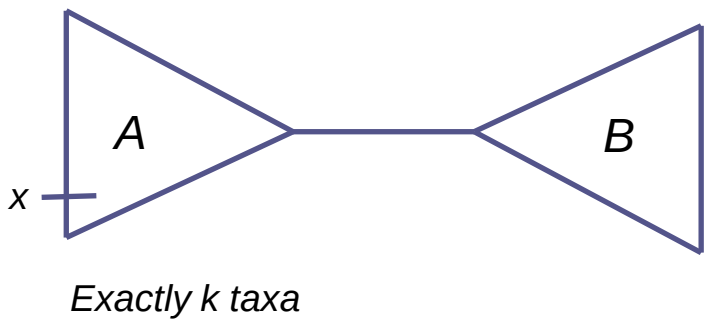
• Use your imagination ☺ The code for counting / listing / sampling $g_k$ characters can be downloaded from my website http://skelk.sdf-eu.org.

# Proof sketches

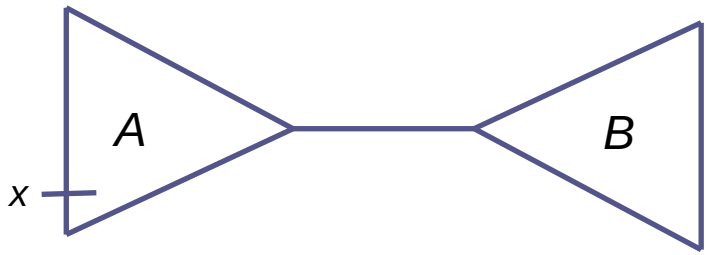
**Proof sketches:**

**1. That caterpillars maximize $g_k$**

## An easy (but very useful) recurrence for computing $g_k(T)$

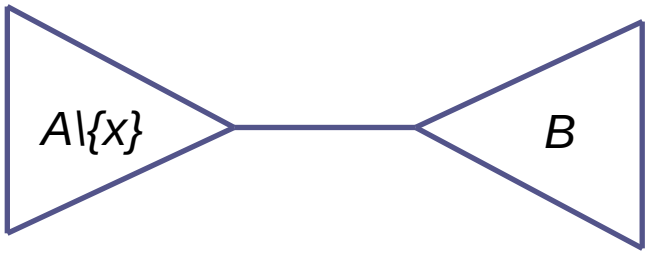


## An easy (but very useful) recurrence for computing $g_k(T)$



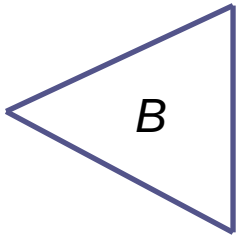

*Exactly k taxa*

=

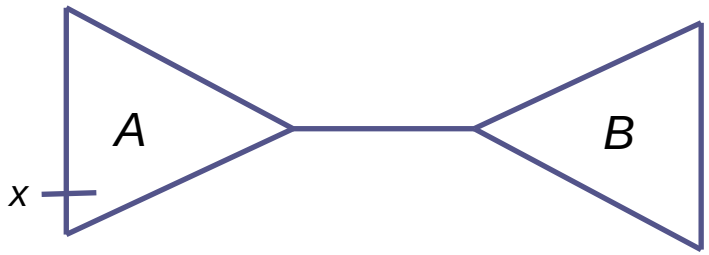



*Exactly k-1 taxa*

+

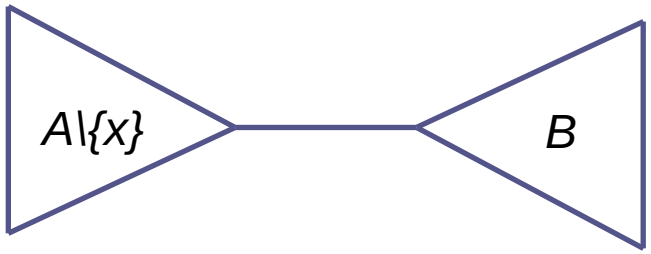


**Now let's apply it to caterpillars:**



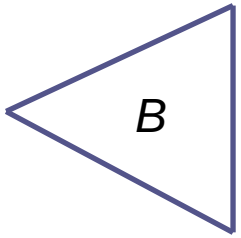

*Exactly k taxa*

=

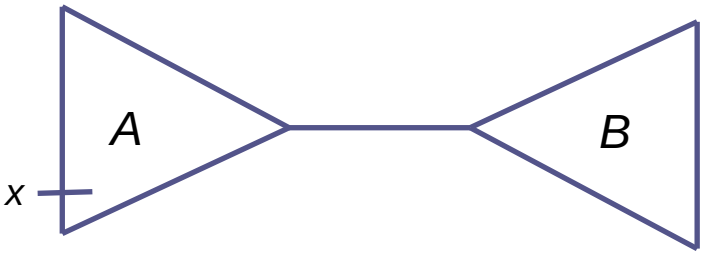



*Exactly k-1 taxa*

+



**Now let's apply it to caterpillars:**




*Exactly k taxa*

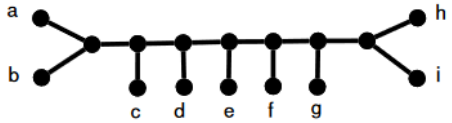
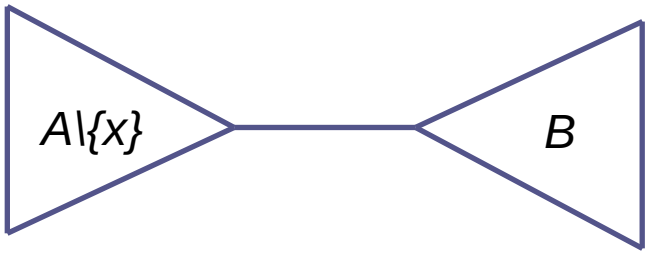


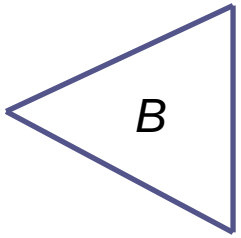

Figure 2: A caterpillar tree on 9 taxa.

=

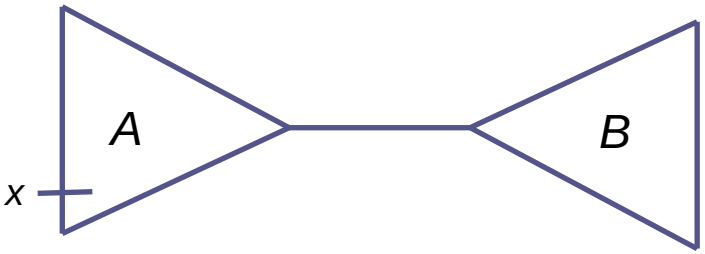



*Exactly k-1 taxa*

+



**Now let's apply it to caterpillars:**




*Exactly k taxa*

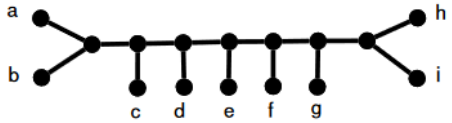
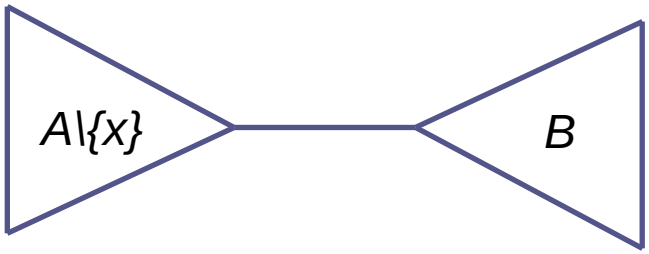


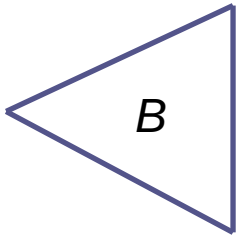

Figure 2: A caterpillar tree on 9 taxa.

=



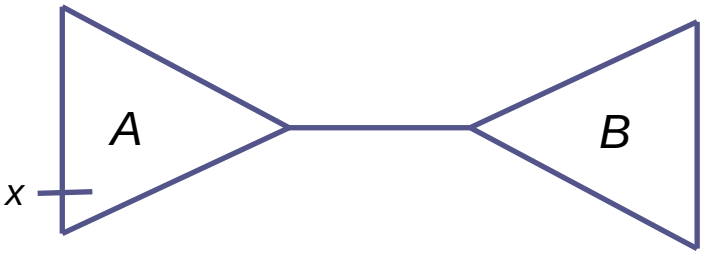

*Exactly k-1 taxa*

+




$g_k(Cat_n) = g_k(Cat_{n-1}) + g_k(Cat_{n-k})$

**Now let's apply it to caterpillars:**




*Exactly k taxa*

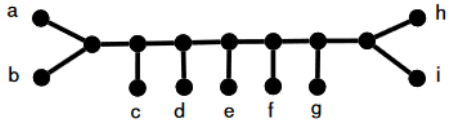
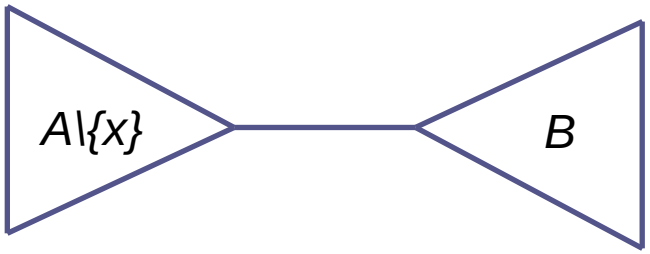


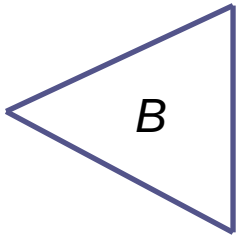

Figure 2: A caterpillar tree on 9 taxa.

=




*Exactly k-1 taxa*

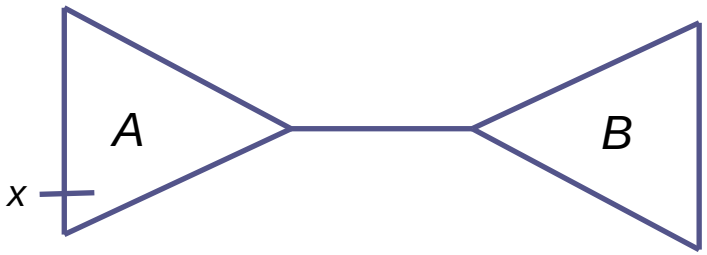
+




$g_k(Cat_n) = g_k(Cat_{n-1}) + g_k(Cat_{n-k})$

Homogenous linear recurrence
(easy to solve!)

**Now let's apply it to caterpillars:**




*Exactly k taxa*

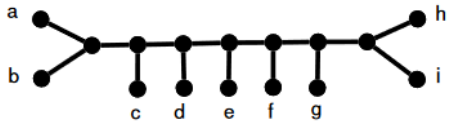
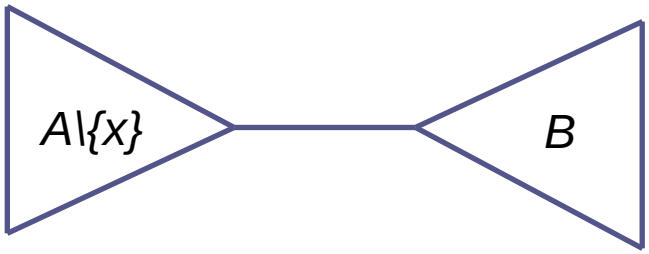


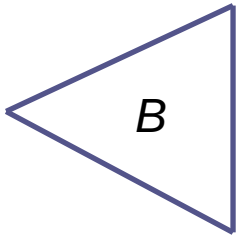

Figure 2: A caterpillar tree on 9 taxa.

=



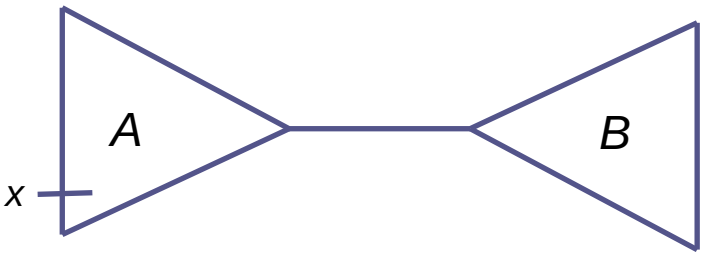

*Exactly k-1 taxa*

+




$g_k(Cat_n) = g_k(Cat_{n-1}) + g_k(Cat_{n-k})$

$$x^k - x^{k-1} - 1 = 0$$




Exactly k taxa

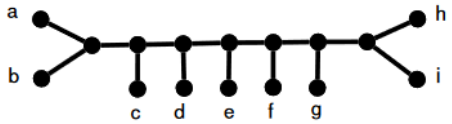
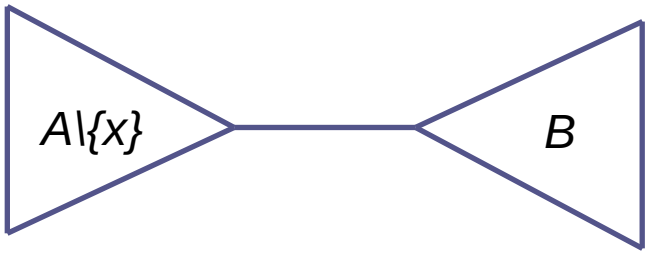



Figure 2: A caterpillar tree on 9 taxa.

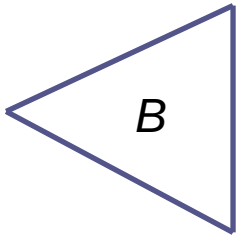
**But why are caterpillars maximizers?**

=




Exactly k-1 taxa

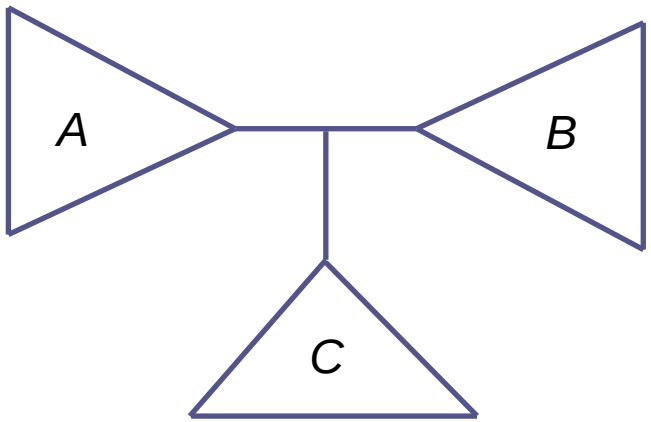
+




$g_k(Cat_n) = g_k(Cat_{n-1}) + g_k(Cat_{n-k})$

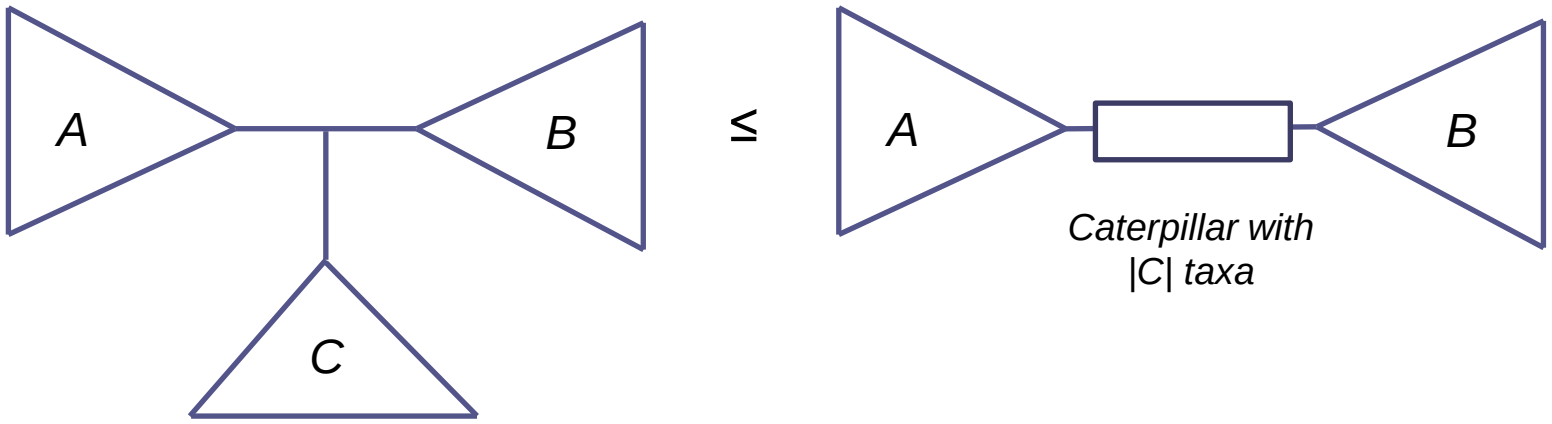
$$x^k - x^{k-1} - 1 = 0$$

**Another easy observation: linearization of small subtrees cannot cause $g_k$ to decrease**



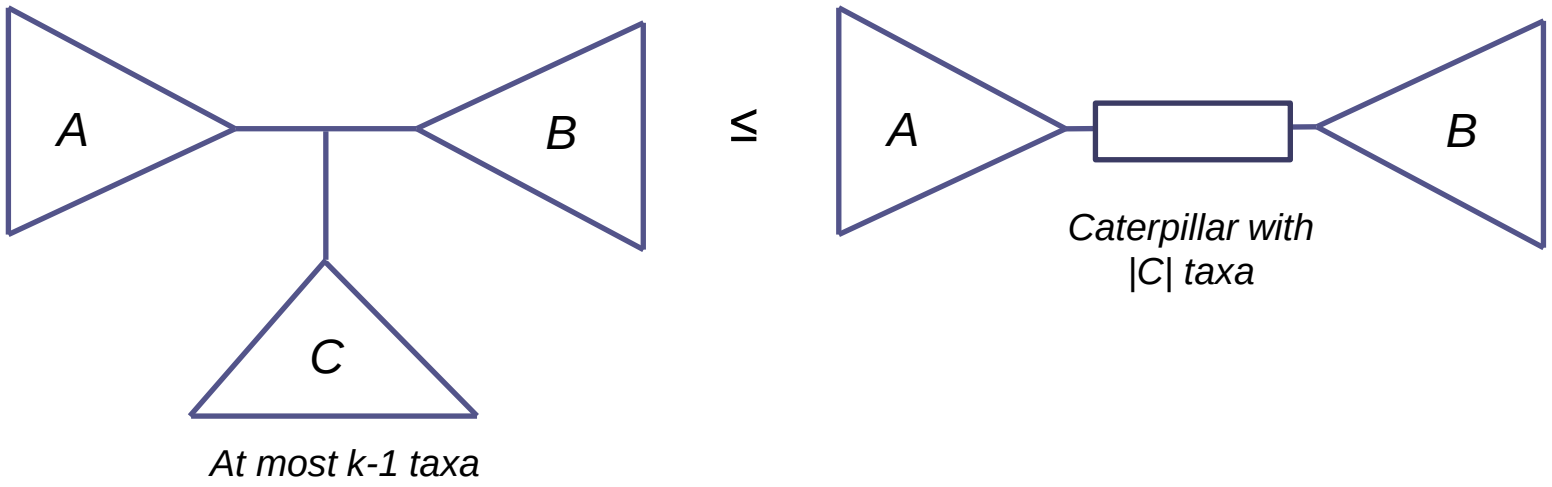

*At most k-1 taxa*

**Another easy observation: linearization of small subtrees cannot cause $g_k$ to decrease**



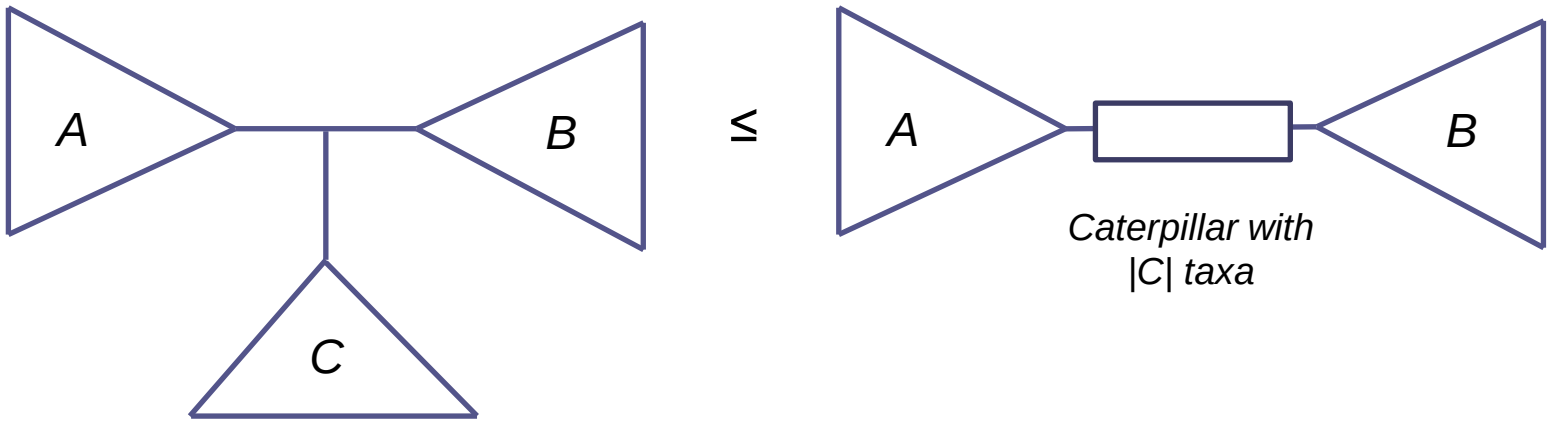

*At most k-1 taxa*

**Proof by induction on (#taxa + #cherries) that caterpillars maximize**




- Let $T$ be an arbitrary maximizer of $g_k(T)$.
- If such a tripartition exists where $|A|, |B| \geq 2$, linearize to get a new tree $T'$.
- We have $g_k(T) \leq g_k(T')$.
- $T'$ has fewer cherries than $T$.
- So by induction $g_k(T) \leq g_k(T') \leq g_k(Cat_n)$.
- Done!

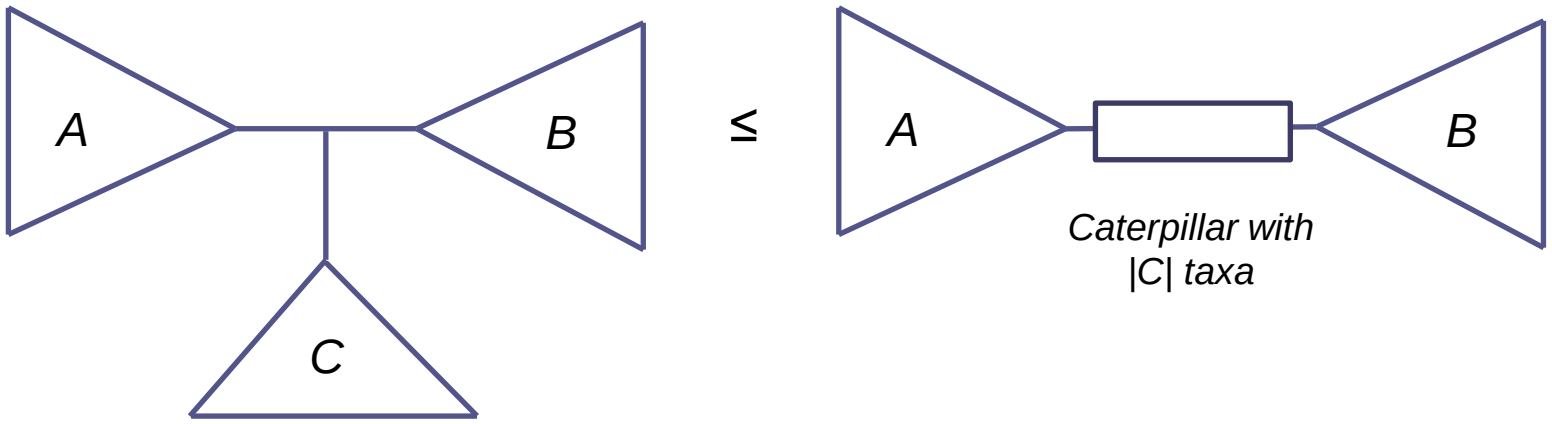
**Proof by induction on (#taxa + #cherries) that caterpillars maximize**




*At most k-1 taxa*

- Let *T* be an arbitrary maximizer of $g_k(T)$.
- If such a tripartition does not exist where $|A|, |B| \geq 2$ *and* $2 \leq |C| \leq k-1$, we can show that *T* must have a split *D*|*E* where $|D|=k$ (I won't prove this today).
- So $g_k(T) = g_k(T \setminus \{x\}) + g_k(T \setminus D)$ where x is an arbitrary taxon in *D*.
- Both terms of the summation have smaller (#taxa + #cherries).
- So $g_k(T) \leq g_k(Cat_{n-1}) + g_k(Cat_{n-k})$
  $= g_k(Cat_n)$

**Proof by induction on (#taxa + #cherries) that caterpillars maximize**




*At most k-1 taxa*

- Let *T* be an arbitrary maximizer of $g_k(T)$.
- If such a tripartition does not exist where $|A|, |B| \geq 2$ *and* $2 \leq |C| \leq k-1$, we can show that *T* must have a split *D*|*E* where $|D|=k$ (I won't prove this today).
- So $g_k(T) = g_k(T \setminus \{x\}) + g_k(T \setminus D)$ where x is an arbitrary taxon in *D*.
- Both terms of the summation have smaller (#taxa + #cherries).
- So $g_k(T) \leq g_k(Cat_{n-1}) + g_k(Cat_{n-k})$
  $= g_k(Cat_n)$ DONE!

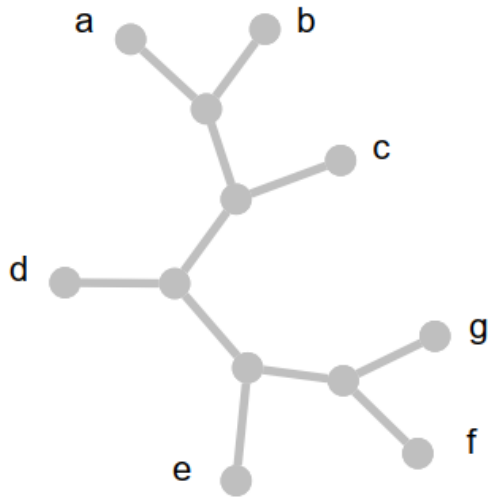
## Proof sketches:
## 2. That fully k-loaded trees minimize $g_k$

**What are fully k-loaded trees?**

•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).

**What are fully k-loaded trees?**

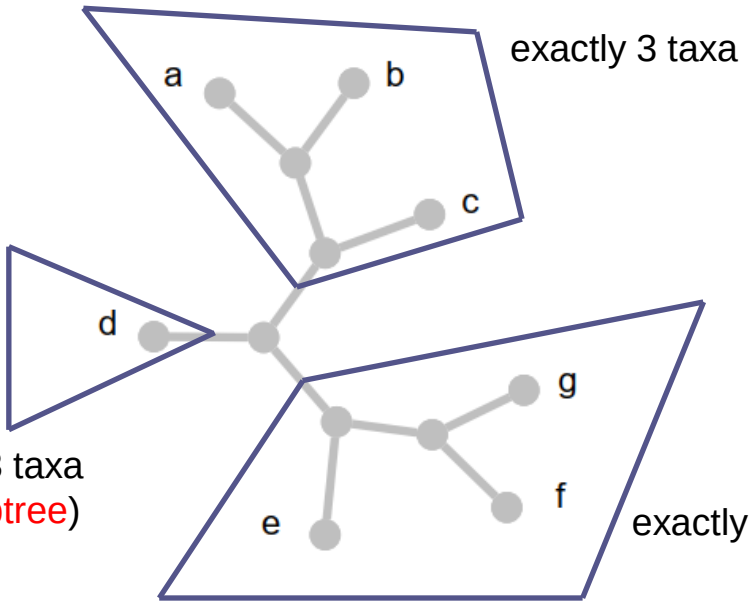
•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).




fully 4-loaded tree

**What are fully k-loaded trees?**

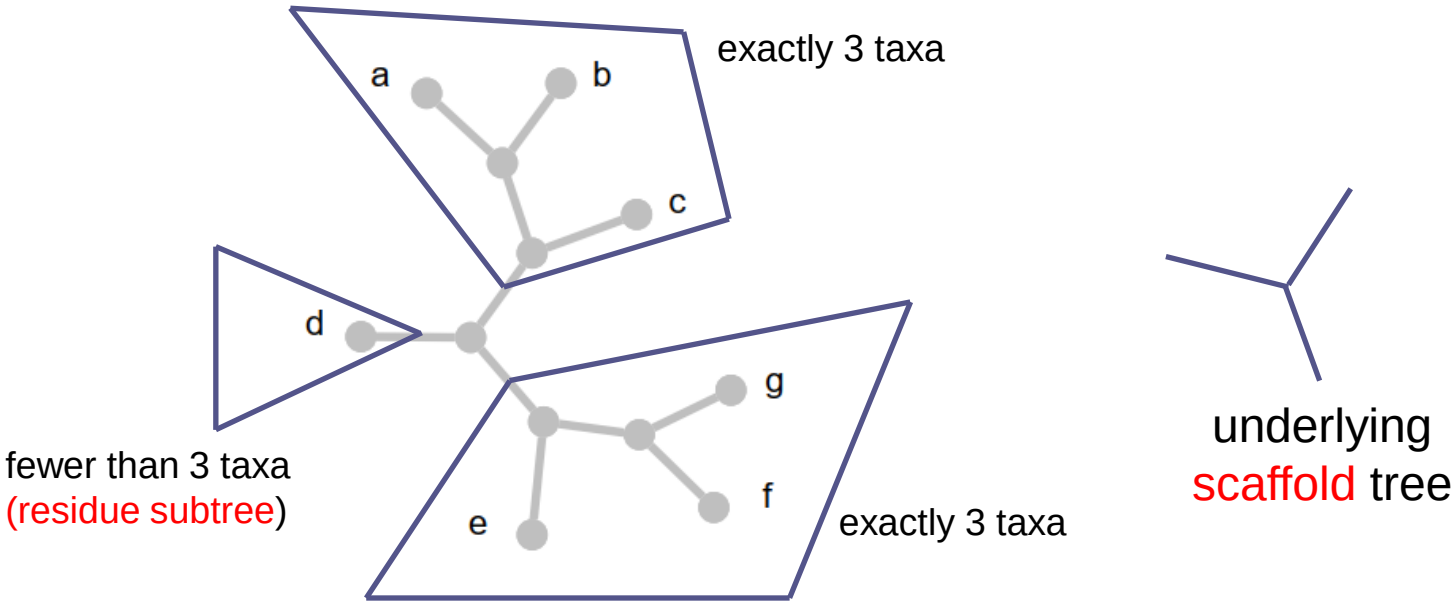
•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).




fully 4-loaded tree

**What are fully k-loaded trees?**

•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).



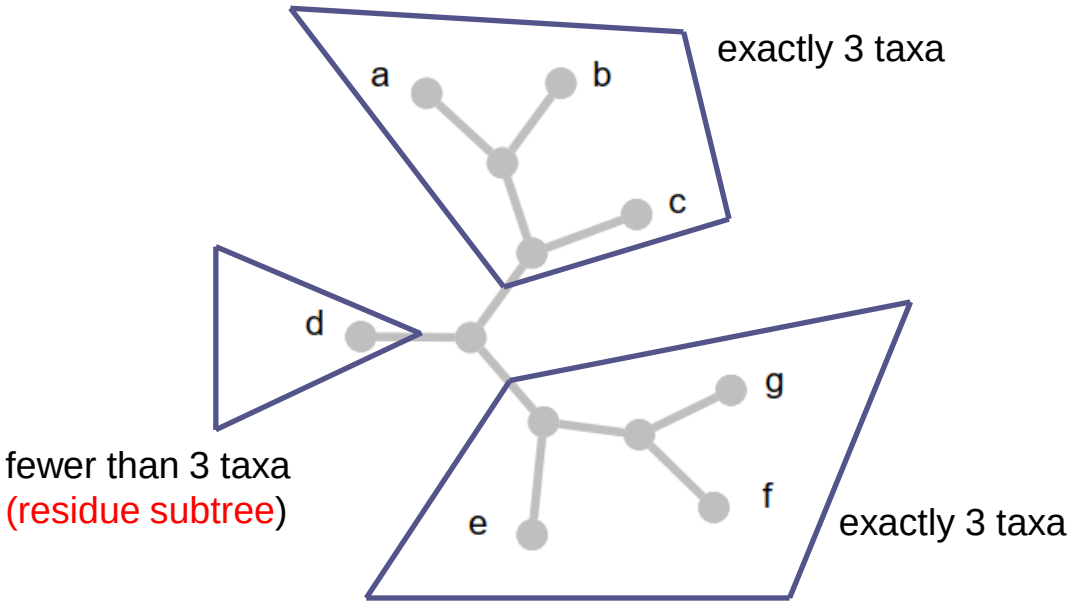

fully 4-loaded tree



underlying
scaffold tree

## What are fully k-loaded trees?

•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).



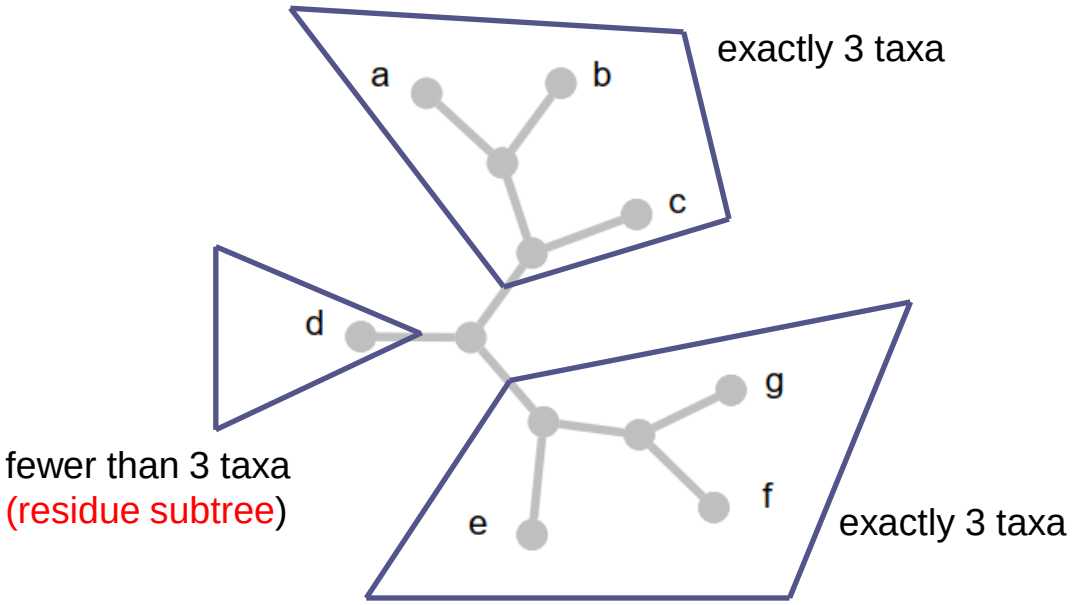

fully 4-loaded tree

underlying
scaffold tree

**If *T* is fully k-loaded, then**
**$g_k(T) = g_2(Scaffold(T))$**

## What are fully k-loaded trees?

•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).



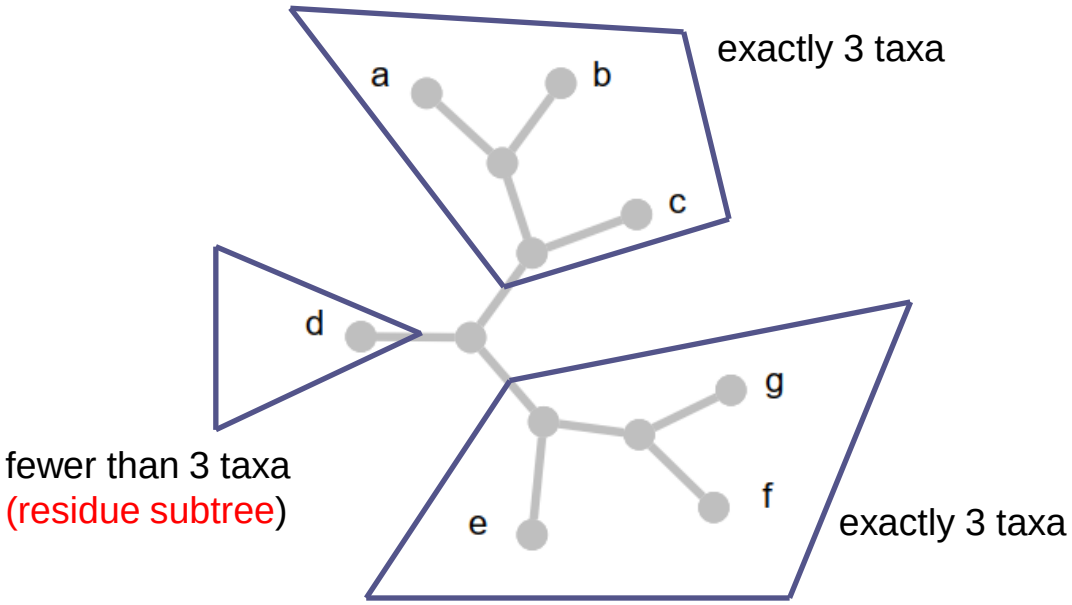

fully 4-loaded tree

underlying
scaffold tree

**So all fully k-loaded trees have the same $g_k$ value (because $g_2$ is topology invariant)**

## What are fully k-loaded trees?

•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).



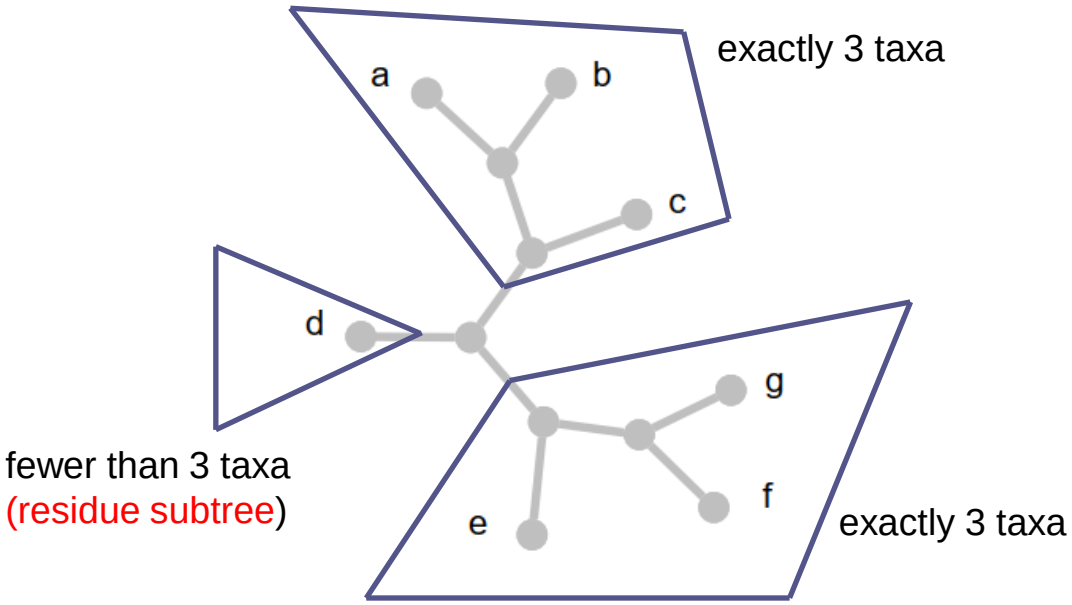

fully 4-loaded tree

underlying
scaffold tree

$$\left\lfloor \frac{\phi^{\lceil \frac{n}{k-1} \rceil - 1}}{\sqrt{5}} + \frac{1}{2} \right\rfloor.$$

**What are fully k-loaded trees?**

•A fully k-loaded tree is a tree where the taxa can be partitioned into pendant subtrees, such that all pendant subtrees have exactly k-1 taxa, except perhaps one, which will have at most k-1 taxa (the “residue” subtree).




fully 4-loaded tree

underlying
scaffold tree

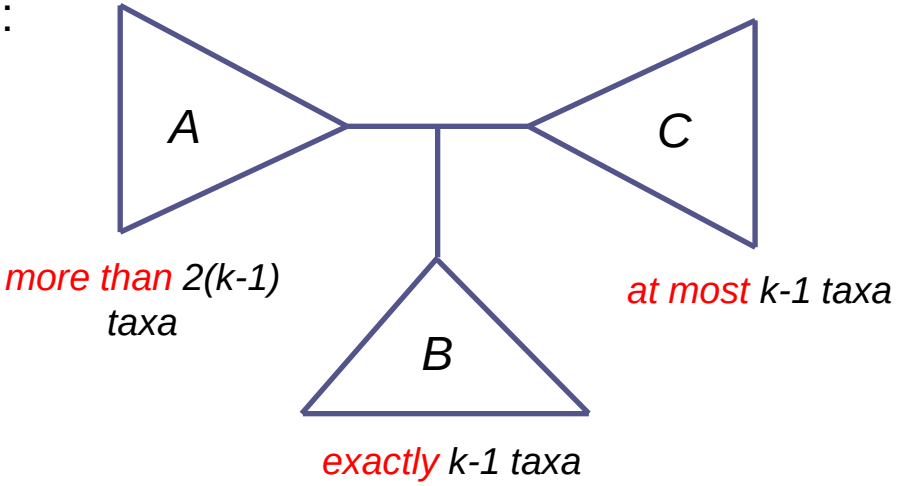
**But why is this minimum?**

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :

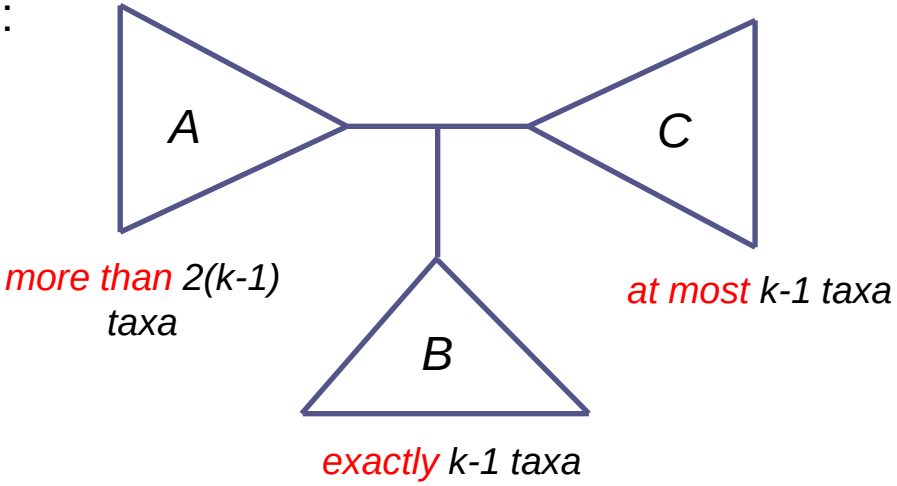
**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :



**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :

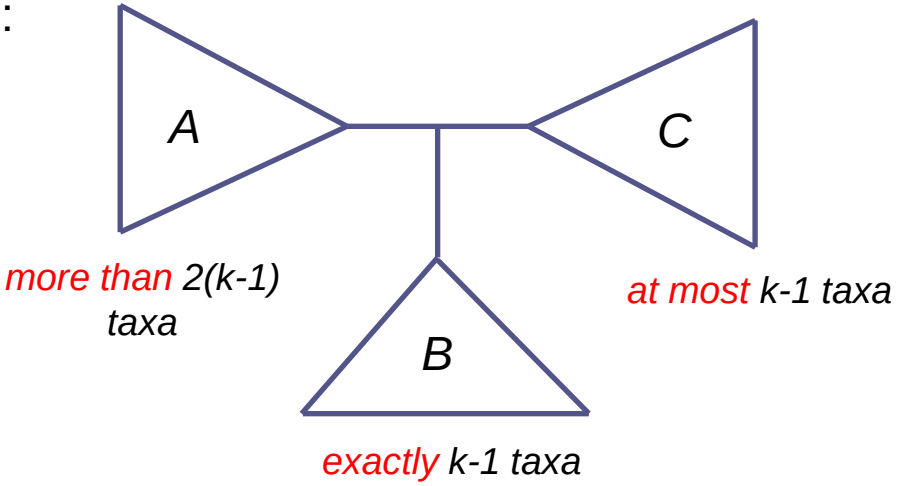



- $g_k(T) = g_k(A+B)\, g_k(C) \quad + \quad g_k(A)\, g_k(B+C) \quad + \quad g_k(A+B)\, g_k(B+C)$

- Note that |A|, |C|, |A+B|, |B+C| all have fewer than *n* leaves

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :

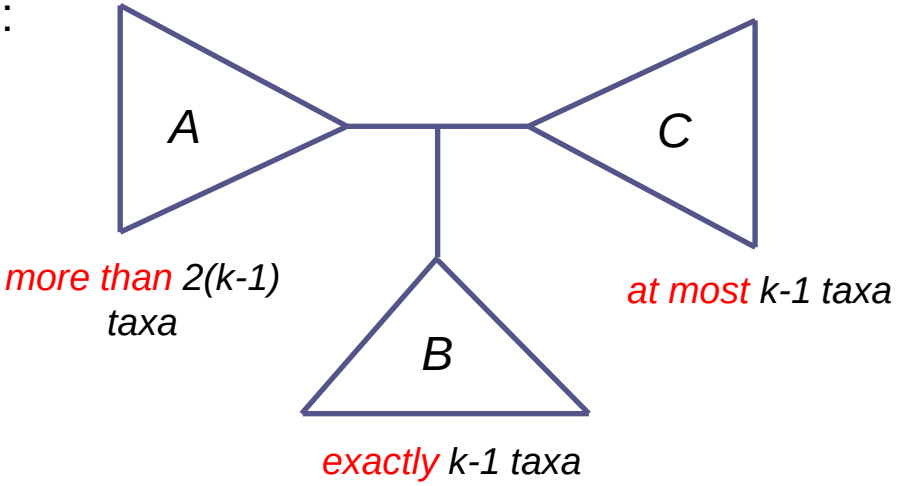



- $g_k(T) = g_k(A+B)\, g_k(C) \quad + \quad g_k(A)\, g_k(B+C) \quad + \quad g_k(A+B)\, g_k(B+C)$

- So by induction the terms $g_k(A+B)$, $g_k(C)$, $g_k(A)$, $g_k(B+C)$ are all larger than or equal to the $g_k$ values for corresponding fully k-loaded trees

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

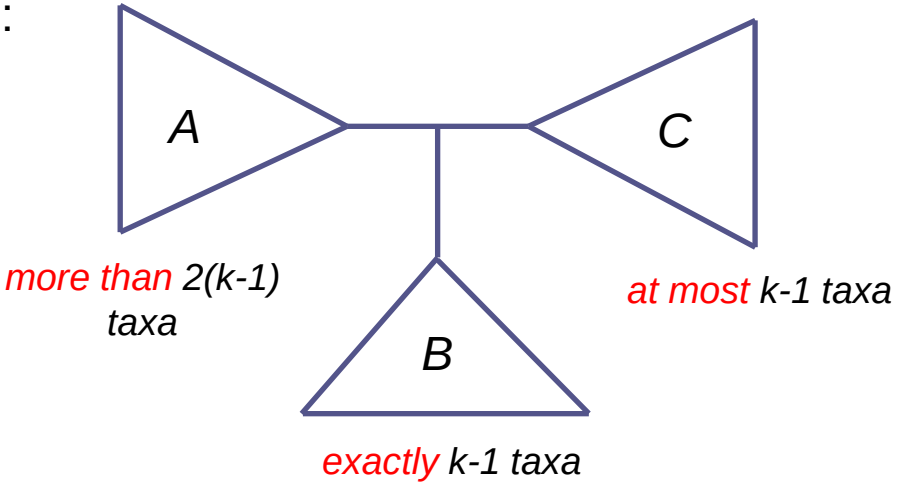
- Let $T$ be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate $T$ so it looks like this, without increasing $g_k$ :




- $g_k(T) \geq g_k^{fl}(|A|+|B|)\, g_k^{fl}(|C|) \quad + \quad g_k^{fl}(|A|)\, g_k^{fl}(|B+C|) \quad + \quad g_k^{fl}(|A+B|)\, g_k^{fl}(|B+C|)$

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let $T$ be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate $T$ so it looks like this, without increasing $g_k$ :

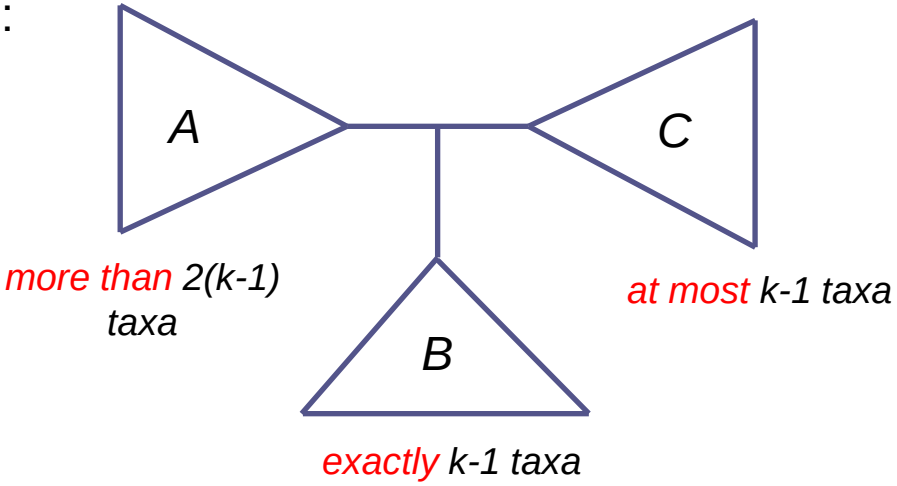



- $g_k(T) \geq g_k^{fl}(|A|+|B|)\, g_k^{fl}(|C|) \quad + \quad g_k^{fl}(|A|)\, g_k^{fl}(|B+C|) \quad + \quad g_k^{fl}(|A+B|)\, g_k^{fl}(|B+C|)$

- If we can show that there exists a fully k-loaded tree whose $g_k(T)$ value is equal to the RHS of the above inequality, we are done.

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :

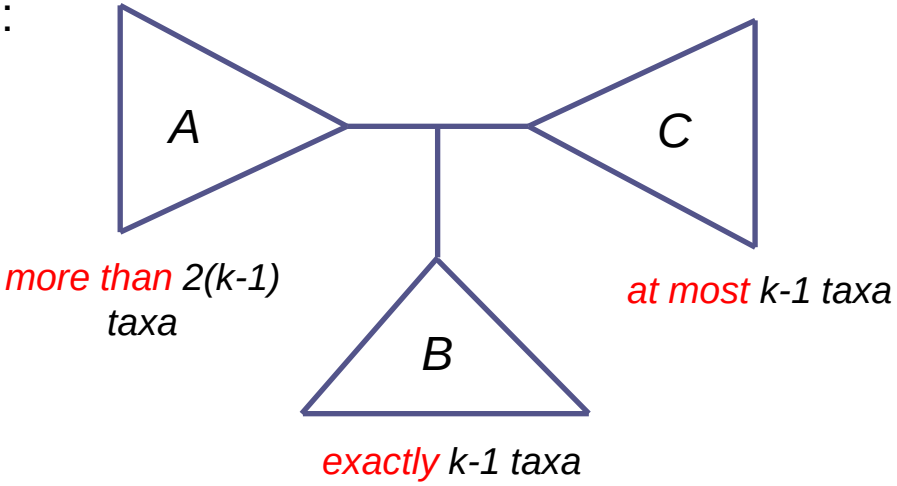



- $g_k(T) \geq g_k^{fl}(|A|+|B|)\, g_k^{fl}(|C|) \quad + \quad g_k^{fl}(|A|)\, g_k^{fl}(|B+C|) \quad + \quad g_k^{fl}(|A+B|)\, g_k^{fl}(|B+C|)$

- **Easier case:** If |A| or |C| is divisible by k-1. We replace subtree A with a fully k-loaded tree. This yields a fully k-loaded tree overall, because there is at most one residue subtree. Done!

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :

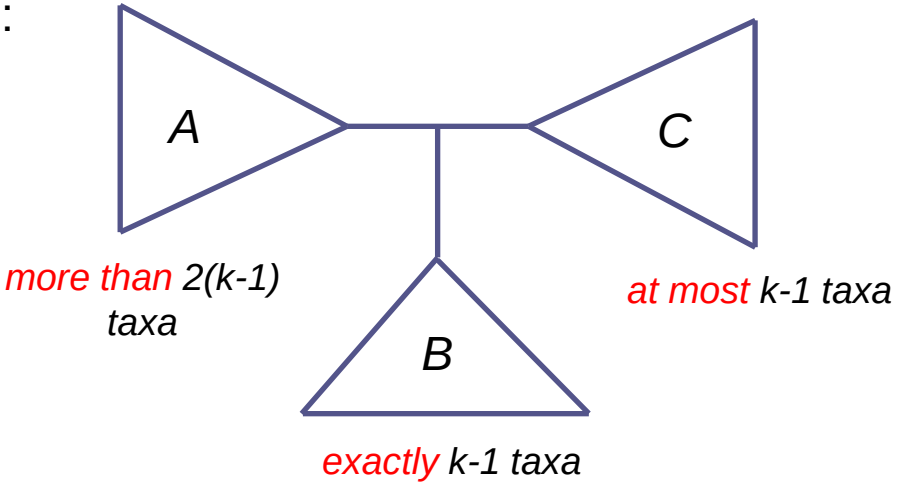



- $g_k(T) \geq g_k^{fl}(|A|+|B|)\, g_k^{fl}(|C|) \quad + \quad g_k^{fl}(|A|)\, g_k^{fl}(|B+C|) \quad + \quad g_k^{fl}(|A+B|)\, g_k^{fl}(|B+C|)$

- **Harder case:** Neither |A| or |C| is divisible by k-1. Problem is that replacing A might leave us with two residue subtrees, and this is not allowed ☹

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :

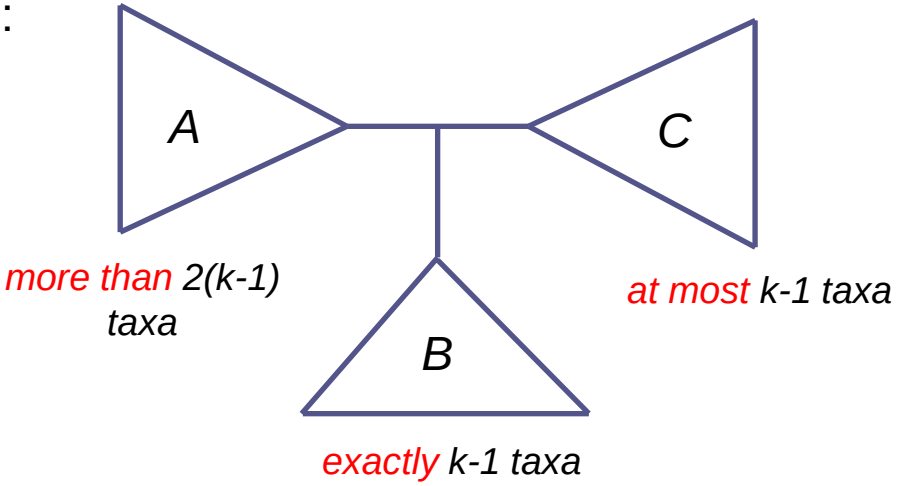



- $g_k(T) \geq g_k^{fl}(|A|+|B|)\, g_k^{fl}(|C|) \quad + \quad g_k^{fl}(|A|)\, g_k^{fl}(|B+C|) \quad + \quad g_k^{fl}(|A+B|)\, g_k^{fl}(|B+C|)$

- **Harder case:** Neither |A| or |C| is divisible by k-1. Problem is that replacing A might leave us with two residue subtrees, and this is not allowed ☹ Requires (tricky!) application of induction *twice.*

**Proof by induction on number of taxa that fully k-loaded trees are minimizers**

- Let *T* be an arbitrary minimizer of $g_k$.
- With a bit of manipulation we can manipulate *T* so it looks like this, without increasing $g_k$ :




- $g_k(T) \geq g_k^{fl}(|A|+|B|)\, g_k^{fl}(|C|) \quad + \quad g_k^{fl}(|A|)\, g_k^{fl}(|B+C|) \quad + \quad g_k^{fl}(|A+B|)\, g_k^{fl}(|B+C|)$

- **Harder case:** Neither |A| or |C| is divisible by k-1. Problem is that replacing A might leave us with two residue subtrees, and this is not allowed ☹ Requires (tricky!) application of induction *twice.* Done ☺

**Conclusions / reflections**

• Convex character programming – try it ☺ Happy to discuss possible applications.

• How do $g_k$ values vary between the lower and upper bounds? Can we parameterize this somehow as a function of tree topology?

• How do $g_k$ values change under the action of common tree rearrangement operations?

• What does the vector of $g_k$ values tell us about a tree? We know examples of non-isomorphic trees that have identical $g_k$ values. Can we characterize when this happens?

• Are there other natural restrictions of convex characters on trees that we could study (in order to further extend the modelling power of convex character programming?) Ideally allowing efficient counting/listing/sampling!

• Networks…?

**Thank you for listening!**