Growing leaves

Graham Jones, www.indriid.com, July 2022

A WORK IN PROGRESS

1 Introduction

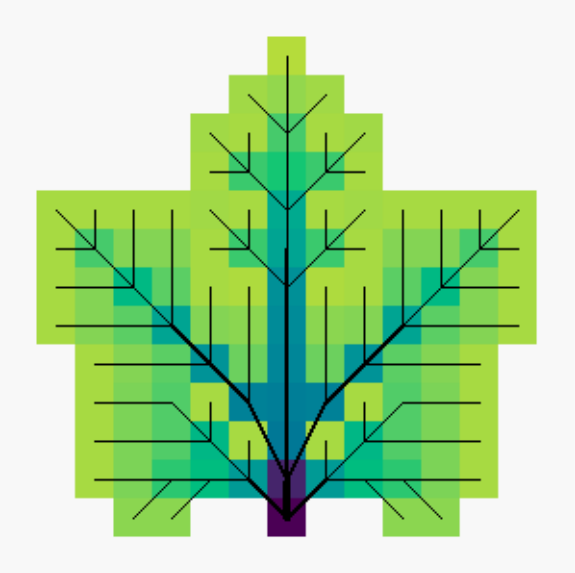

Figure 1: A mathematical leaf.

A leaf with a larger area collects more light, but requires a bigger network of veins to deliver water and nutrients to every part. A 'good' leaf maximizes area while minimizing transport costs. This is a simplified account of a paper by Qinglan Xia which describes a model in which these conflicting aims are balanced against each other:

The Formation of a Tree Leaf
https://pdodds.w3.uvm.edu/research/papers/others/2007/xia2007a.pdf

The leaf is made of square cells on a grid, together with a network of veins which supplies each cell from the base of the leaf.

The network of veins is a sort of mini-tree with its own 'root' at the base of the leaf. In graph theory terms, it is rooted tree (a tree with one distinguished vertex or node). There is a unique directed path from the root to every other node. We can talk about 'the' edge of a node, since only one edge leads to a node. Each node sits at the centre of a cell. I'll use the words 'cell' and 'node' interchangeably.

2 The algorithm

Here's the basic algorithm.

```
1. Start with a tree, G, containing one node, the root.
REPEAT FOREVER:
  2. Find the squares just outside G that can be added
     with the smallest possible increment in F(G). If
     the increment in F(G) is less than a threshold,
     add them as new cells, otherwise STOP.
  3. Optimize G, by repeatedly visiting every node n,
     and trying to find a better parent for it.
```

The tree G is a list of nodes. $F(G)$ is the total transport cost of G.

In step 2, 'just outside' means being an 8-neighbour of a node in G. The increment in $F(G)$ can be found by making a new tree G_2 with the candidate added, and calculating $F(G_2) - F(G)$. The threshold is called ϵ and it determines how big the leaf grows.

In step 3, the candidate new parents for n are those which are either an 8-neighbour of n or an 8-neighbour of n's current parent, but are not n itself, nor a descendant of n. For each candidate, make a new tree G_2 with the change in parent. If $F(G_2) < F(G)$, replace G with G_2. Since changing G in one place may make improvements elsewhere possible, the process is repeated until no node can be improved.

This is a simple but inefficient way to implement the algorithm. It is is spelled out in more details in section 6, and a more efficient implemention is described.

Along with ϵ, there are two parameters α and β which determine the shape and 'style' of the leaf.

3 The total transport cost $F(G)$

The nodes

Each node n has the following information which together define the topology (branching pattern) and geometry of the tree.

- The parent node parent(n). The root does not have a parent.
- A list children(n) of child nodes, which may be empty.

- Integers $x(n)$ and $y(n)$ giving the 2D coordinates of n. The root is at (0,0) and is assumed to be supplied from the plant by an invisible edge coming from (0,-1).

Each node n also has values which are used to calculate the transport cost.

- The length $d(n)$ of the edge of n.
- Weight $w(n)$, which is the total amount of fluid required at n, for n itself and all its descendants.
- The local 'bending cost' $b(n)$.
- The total bending cost $m(n)$ of a node n is the product of $b()$ values along the path leading from the root to n.
- The local transport cost $f(n)$ which is equal to $m(n)^\beta w(n)^\alpha d(n)$.

The total transport cost $F(G)$ is the sum of all the $f(n)$.

The weights

Each cell needs the same amount of fluid, which is assumed to be 1. This means $w(n)$ is equal to the number of cells that are descendants of n, plus 1. It is proportional to the cross-sectional area of a vein. Fig 2.

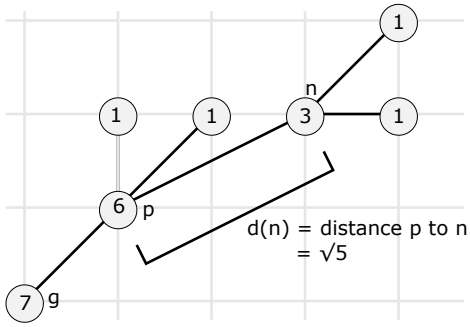

Figure 2: A part of a tree with weights and distances. Node n has parent p and grandparent g. The weight of n and other nodes are in the circles. The distance of n to p is $d(n) = \sqrt{2^2 + 1^2} = \sqrt{5}$.

The bending costs

For most nodes, the local bending cost $b(n)$ is derived from the angle between the edge of parent(n) and the edge of n. This requires n to have a parent and a grandparent parent(parent(n)), so it doesn't work near the root. If parent(n) is the

root, the invisible edge coming from (0,-1) mentioned above is used. For the root itself, we set $b(\text{root}) = 1$.

If the angle θ of the bend is less than a right angle,

$$b(n) = 1/\cos\theta.$$

Otherwise $b(n) = \infty$, which means that sharp bends never appear in the tree. Fig 3.

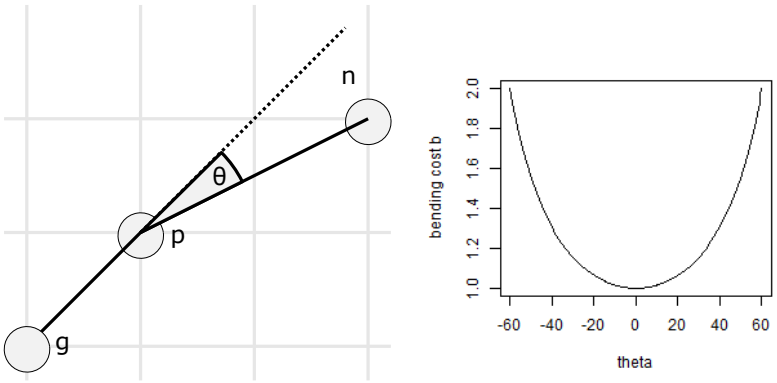

Figure 3: on the left, node n and the bend angleθ. Here, $\cos\theta = (2\times1+1\times1)/(\sqrt{5}\sqrt{2}) = 3/\sqrt{10}$and $b(n) = \sqrt{10}/3 = 1.054$. On the right, a graph of $b(n)$.

Starting with $m(\text{root}) = 1$, other nodes can be given $m()$ values by repeatedly applying $m(n) = b(n)m(\text{parent}(n))$.

3.1 The parameters α and β

Assume the veins have the same shaped cross-section, for example circular. If the cost of a vein was proportional to the circumference, it would be proportional to $w(n)^{1/2}$. On the other hand, if there was no saving in using one big vein instead of several little ones, the cost would be $w(n)$. The formula $w(n)^\alpha$ as α ranges from 0.5 to 1 interpolates between these possiblities. Values of α outside this range seem harder to justify phyysically.

Bends presumably incur some cost, requiring extra material for rigidity as well as containing the fluid, but I don't know any explanation for Xia's particular formula for $b(n)$.

3.2 When does it stop growing?

When some cells are added, the veins are optimized, and this may mean that the next cells can be added at a smller cost than the previous ones. It's not obvious

that the cost will ever reach ϵ and the leaf might keep growing forever. Xia proved a theorem that ensures that it does stop whenever when $\alpha > 1/2$.

My proof is based on Xia's, but I hope simpler. For convenience I assume that $\alpha \leq 1$ as well. We start with a lemma and a definition.

Lemma 1 *Suppose $x \geq 1$. The function $f(x) = (x+1)^\eta - x^\eta$, is decreasing if $\eta \leq 1$ and increases without limit if $\eta > 1$.*

Proof Exercise!

Let path(n) be the set of nodes in the path from root to n, including n, but not the root. The potential function $P(n, x)$ provides the cost of adding an extra weight x along the path from root to a node n. This is

$$P(n,x) = \sum_{u \in \text{path}(n)} m(u)^\beta [(w(u)+x)^\alpha - w(u)^\alpha] d(u). \tag{1}$$

(This potential function is also useful for an efficient implementation in section 6.)

Theorem 1 *The algorithm for growing a leaf stops if $1/2 < \alpha \leq 1$*

Proof Let $R(G)$ be the Euclidean distance from the root to a node in G which is furthest away from the root. When a node is added, $R(G)$ may or may not increase, but there will be an infinite number of new nodes which do increase $R(G)$ as G grows. Let r be one of them, so it is a square just outside the G at least as far away as $R(G)$. Assume that b is the chosen parent of r inside G. The cost of adding r is at least

$$\Delta(r) = d(r) m(u)^\beta + P(b, 1)$$

Here $d(r)$ is the distance from b to r. Since $m(u)^\beta$ is always at least 1,

$$\Delta(r) \geq d(r) + \sum_{u \in \text{path}(n)} \big[(w(u)+1)^\alpha - w(u)^\alpha\big] d(u).$$

We're assuming $\alpha \leq 1$, so by the Lemma, each of the terms $[(w(u)+x)^\alpha - w(u)^\alpha]$ gets smaller as $w(u)$ gets bigger. The biggest any $w(u)$ could be is the number of nodes $|G|$ in G, which is equal to the area covered by G. So

$$\Delta(r) \geq d(r) + \sum_{u \in \text{path}(n)} \big[(|G|)+x)^\alpha - |G|^\alpha\big] d(u).$$

Using $(|G|)+1)^\alpha - |G|^\alpha \le 1$, we get

$$\Delta(r) \ge \left[(|G|)+1)^\alpha - |G|^\alpha\right] \left(d(r) + \sum_{u \in \text{path}(n)} d(u) \right).$$

The term in big brackets is the length of the path from r to the root, so it must be at least $R(G)$.

$$\Delta(r) \ge \left[(|G|)+1)^\alpha - |G|^\alpha\right] R(G)$$

Now $|G|$ cannot be any bigger than the circle centered at the root and with radius $R(G)+1 \le 2R(G)$. (We add 1 because parts of squares might stick out of the circle.) So $|G| \le 4\pi R(G)^2 \le (4R(G))^2$, so $R(G) \ge (1/4)|G|^{1/2}$, and

$$\Delta(r) \ge (1/4)\left[(|G|)+1)^\alpha - |G|^\alpha\right] |G|^{1/2}$$

$$\ge (1/4)\left[(|G|)+1)^{\alpha+1/2} - |G|^{\alpha+1/2}\right]$$

Now we can use the Lemma with $\eta = \alpha + 1/2$. Since $\alpha > 1/2$, $\eta > 1$, which shows $\Delta(r)$ will eventually get bigger than ϵ as $|G|$ gets bigger.

4 Examples

In the pictures, the veins are shown with width proportional to the square root of $w(n)$, or the diameter of the vein. The colours are based on the transport cost for a cell and all its descendants, with darker colours for bigger costs.

TODO. explanations, other examples....

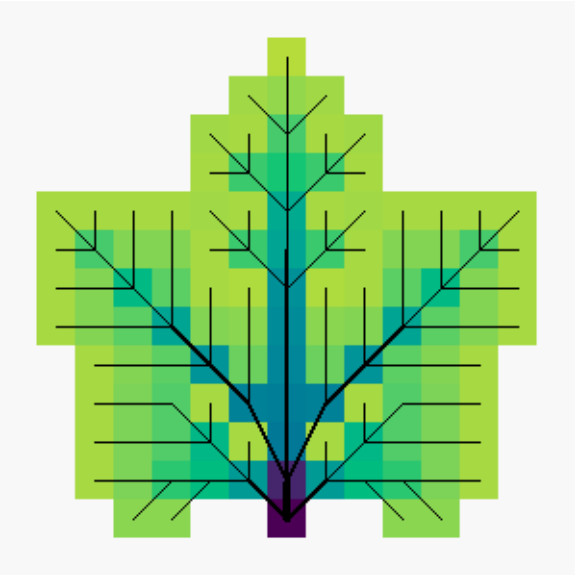

Figure 4: $\alpha = 0.70, \beta = 0.70, \epsilon = 5.00$

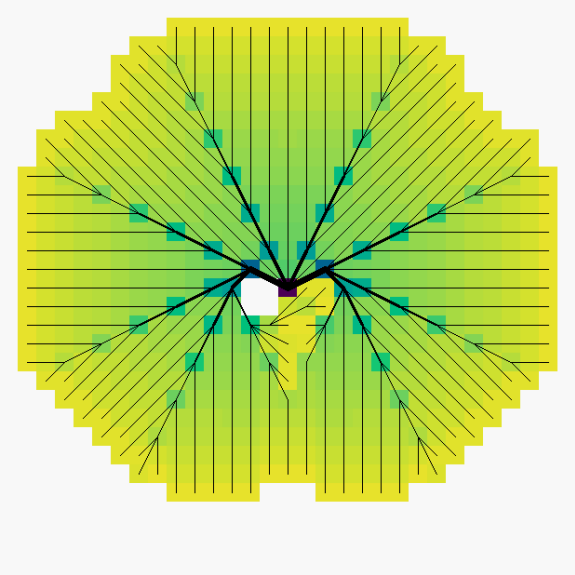

Figure 5: 95-1-1300

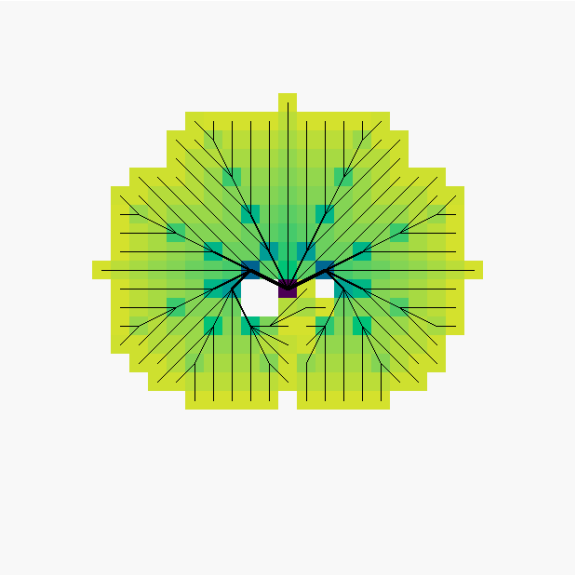

Figure 6: 95-1-900

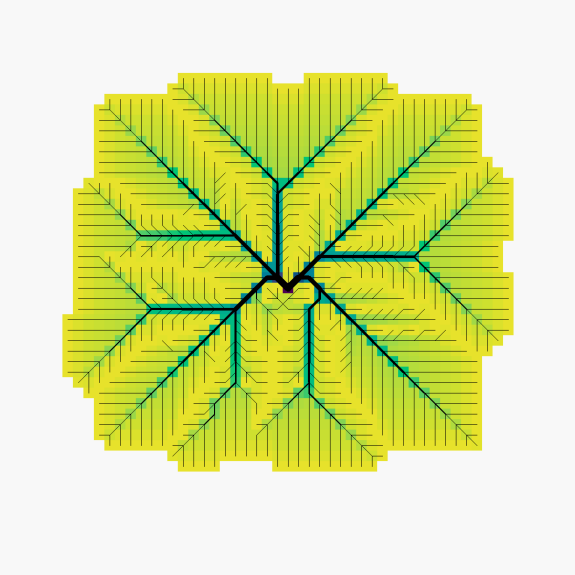

Figure 7: 70-1-650

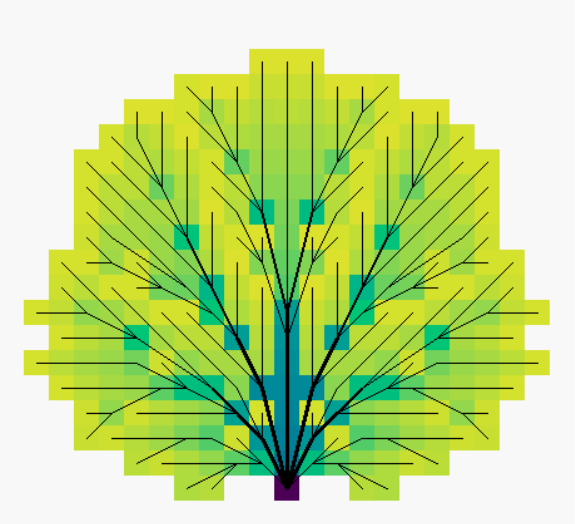

Figure 8: 95-150-1500

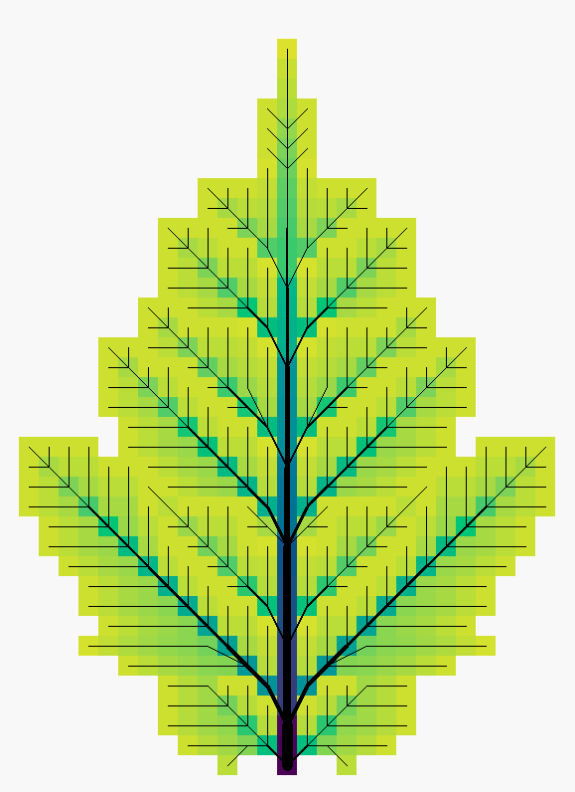

Figure 9: 55-100-500

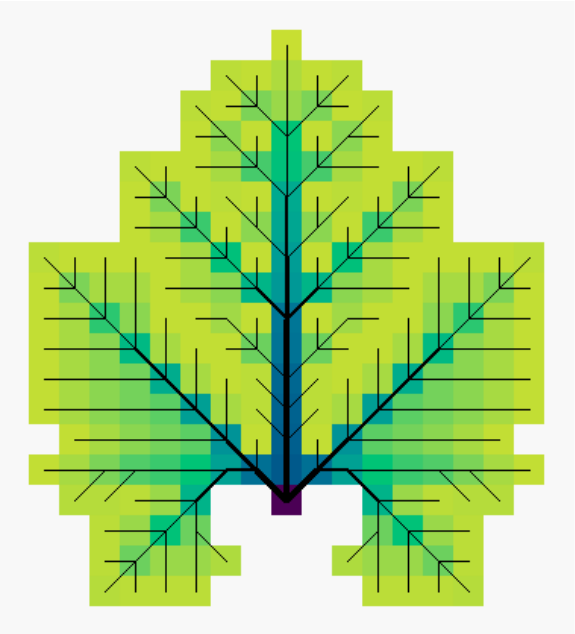

Figure 10: 68-38-500

5 What's the purpose?

TODO. caraciture.

TODO lily, nasturtium, holly, grass

6 Appendix for programmers

This is aimed at people who want to write their own code. I recommend starting with the simple version.

6.1 Pseudo-code for simple version

```
transport_cost(G) {
  fill in w(n) for every node, from tips to root (postorder)
  fill in b(n) for every node
  fill in m(n) for every node, from root to tips (preorder)
  find d(n) and fill in f(n) for every node
  return the sum of the f(n)'s
}
```

For w(n), we visit every child of a node before the node itself. At a tip x, $w(x) = 1$. Otherwise, we add:

$$w(n) = 1 + \sum_{c \in \text{children}(n)} w(c).$$

For $m(n)$ we visit parents before children. At the root, $m(n) = 1$. Otherwise, we multiply:

$$m(n) = b(n)m(\text{parent}(n)).$$

```
add_cells(G, epsilon) {
  A = empty list
  for (every cell c outside the tree) {
    for (every 8-nbr p of c which is inside the tree) {
      make a new tree G2 with c as child to p
      calculate cost(c,p) = transport_cost(G2)-transport_cost(G)
      Add triple (c,p,cost(c,p)) to A
    }
  }
  find the minimum cost minc of all the cost(c,p)'s in A
  if (minc < epsilon) {
    add all (c,p) with minimum cost equal to minc to G
  }
  return G
}
```

This adds new cells 'as slowly as possible'. Xia's paper uses a sequence of thresholds ϵ.... It is not clear how these are chosen. TODO.

`simple-xia.R` uses a sequence of thresholds.

`faster-xia.R` uses the 'as slowly as possible' method.

```
optimize_veins(G) {
  for (every nonroot node n) {
    p = parent(p)
    Q = list of every 8-nbr of n and every 8-nbr of p
    for (every q in Q which is not n nor a descendant of n) {
      make a new tree G2 with n having parent q not p
      if (transport_cost(G2) < transport_cost(G)) {
        G = G2
      }
    }
  }
  return G
}
```

The order in which nodes are visited make a slight difference to the way the leaf develops.

```
Set alpha, beta, epsilon
Make a one-node tree G for the root.
repeat:
  G2 = add.cells(G, epsilon)
  if (G2 no bigger than G) {
    break
  }
  F = transport_cost(G)
  repeat:
    G2 = optimize_veins(G)
    F2 = transport_cost(G2)
  until (F2 >= F)
until false
```

6.2 More efficient implementation

The rest of this section is about how to write a more efficient version, and The main inefficiency in the simple version is in the large number of times $F(G)$ is calculated, especially during vein optimization. To reduce the amount of recalculation, it is useful to store a transport cost for subtrees at nodes.

Let subtree(n) be the subtree at n, containing n and all its descendants. Then the

transport cost $F(\text{subtree}(n))$ is the cost of supplying n and all its descendants, and $F(G) = F(\text{subtree}(\text{root}))$.

Adding new cells

The cost of adding a child c to n uses the potential function from equation (1). It is

$$P(n,1) + f(c) = P(n,1) + m(c)^\beta d(c) = P(n,1) + [m(n)b(c)]^\beta d(c). \tag{2}$$

Using this, we can just look along the path, instead of recalculating $F(G_2)$ for a whole new tree G_2.

Reparenting

We consider the effect on several regions of the tree: the path leading to n; n itself; and the subtrees of each child of n. Suppose n is reparented from $p = \text{parent}(n)$ to q.

The weights along path(p) and path(q) change, but there is no change to bending costs b or distances d in these paths. So the change in $F(G)$ due to changes here is due to removing weight $w(n)$ from path(p) and adding it to the path to path(q). Set

$$\Delta_{\text{path}} = P(q, w(n)) - P(p, w(n)). \tag{3}$$

At n, there is no change in $w(n)$, but both $d(n)$ and $b(n)$ may change. I'll use subscripts p and q to distinguish the old and new values of $b(n)$, $d(n)$, etc. Then

$$f_q(n) = \left(\frac{b_q(n)}{b_p(n)}\right)^\beta \frac{d_q(n)}{d_p(n)} f_p(n)$$

. Set

$$\Delta_n = f_q(n) - f_p(n) = \left[\left(\frac{b_q(n)}{b_p(n)}\right)^\beta \frac{d_q(n)}{d_p(n)} - 1\right] f_p(n). \tag{4}$$

For a child c of n, there is no change in $w(c)$ or $d(c)$, but $b(c)$ may change, since it depends on parent(parent(c)) = parent(n). This means that for any $u \in \text{subtree}(c)$, the value of $m(u)$ gets multiplied by the factor

$$B(c) = \frac{b_q(n)}{b_p(n)} \frac{b_q(c)}{b_p(c)},$$

to account for the changes in the bends before reaching u. So we have $f_q(u) = B(c)^\beta f_p(u)$ for all u in the subtree at c, and so

$F_q(\text{subtree}(c)) = B(c)^\beta F_p(\text{subtree}(c))$ for each child c of n. Set

$$\Delta_{\text{subtrees}} = \sum_{c \in \text{children}(n)} \left[\left(\frac{b_q(n)}{b_p(n)} \frac{b_q(c)}{b_p(c)} \right)^\beta - 1 \right] F_p(\text{subtree}(c)) \tag{5}$$

Putting this together, the change in $F(G) = F(\text{subtree(root)})$ is

$$\Delta_{\text{root}} = \Delta_{\text{path}} + \Delta_n + \Delta_{\text{subtrees}}. \tag{6}$$

Note that the values of f and F that appear in equations (3, 4, 5) are all old values, and the w's, d's and b's are all local to nodes. This means we can compare the cost of reparenting from p to q by only visiting n and its children, and the two alternative paths back to the root.