

## Programming the Wagner Uppass Algorithm

Peter Coxhead

School of Computer Science, University of Birmingham, Birmingham, UK  
p.coxhead@cs.bham.ac.uk

The source for the algorithm is Swofford, DL & Maddison, WP, "Reconstructing Ancestral Character States Under Wagner Parsimony", *Mathematical Biosciences* 87:199-229 (1987). The key information is at the bottom of page 214.

In the notation used here, Swofford & Maddison's  $\circ$  is replaced by  $\sqcap$ , but with the same meaning.

My formulation of their algorithm is as follows.

- Start with a rooted tree resulting from the application of the downpass algorithm.
- At each interior node of the tree  $i$  we will initially store three quantities:  $S_i$ ,  $I_i$  and  $U_i$ . At the start of the uppass algorithm,  $S_i$  holds the value resulting from the downpass algorithm.  $S_i$ ,  $I_i$  and  $U_i$  will be updated as the algorithm proceeds.  $U_i$  will hold the final uppass values.
- Begin by initializing the root node by setting  $I_{\text{root}}$  to  $S_{\text{root}}$ .
- Now perform a pre-order tree walk (i.e. process a node followed by a recursive call of the algorithm to process its left and right children), starting from the root node, visiting only interior nodes (i.e. ignoring leaves).
- Suppose we are at an interior node  $p$  which has children  $q$  and  $r$ . Then perform the following calculations.

### Version 1

$$T_q \leftarrow S_r \sqcap S_p$$

Note that  $S_p$  will have been updated in a previous step, or does not need to be updated if  $p$  is the root.  $T_q$  and  $T_r$  are 'working values' and are not stored at a node.

$$T_r \leftarrow S_q \sqcap S_p$$

$$J_q \leftarrow S_q \sqcap T_q$$

$J_q$  and  $J_r$  are 'working values' and are not stored at a node.

$$J_r \leftarrow S_r \sqcap T_r$$

$$U_p \leftarrow J_r \cap J_q \cap I_p$$

Note that  $I_p$  will have been stored in a previous step when  $p$ 's parent was processed.  $U_p$  is stored at the node  $p$  and is its final uppass value. See below for an improvement to this line.

if (node  $q$  is not a leaf) Update the children of node  $p$  ready for subsequent steps.

$$S_q \leftarrow T_q$$

$$I_q \leftarrow J_q$$

if (node  $r$  is not a leaf)

$$S_r \leftarrow T_r$$

$$I_r \leftarrow J_r$$

- It can be seen that actually it is not necessary to store the  $U_i$  values separately. Since  $I_p$  is not needed once  $U_p$  has been calculated, the final uppass value can be stored in  $I_p$ .

Subsequent processing of the uppass tree is likely to expect the resultant values to be stored in  $S_i$  rather than  $I_i$ . Hence a final pass over the tree is needed with this version of the algorithm, copying  $I_i$  to  $S_i$ .

Alternatively, a pre-pass could be used to copy  $S_i$  into  $I_i$ , and then  $I$  and  $S$  interchanged in the formulae above. With this change, the algorithm becomes:

- Start with a rooted tree resulting from the application of the downpass algorithm.

- At each interior node of the tree  $i$  we need to store two quantities:  $S_i$  and  $I_i$ . At the start of the uppass algorithm,  $S_i$  holds the value resulting from the downpass algorithm; at the end of the algorithm,  $S_i$  will hold the final uppass values.
- Begin by passing through all nodes of the tree (in either pre- or post-order), copying  $S_i$  into  $I_i$ .
- Now perform a pre-order tree walk (i.e. process a node followed by a recursive call of the algorithm to process its left and right children), starting from the root node, visiting only interior nodes (i.e. ignoring leaves).
- Suppose we are at an interior node  $p$  which has children  $q$  and  $r$ . Then perform the following calculations.  $T_i$  and  $J_i$ , as before, are local variables for the calculation and are not stored at node  $i$ .

### Version 2

$$T_q \leftarrow I_r \sqcap I_p$$

$$T_r \leftarrow I_q \sqcap I_p$$

$$J_q \leftarrow I_q \sqcap T_q$$

$$J_r \leftarrow I_r \sqcap T_r$$

$$S_p \leftarrow J_r \cap J_q \cap S_p$$

if (node  $q$  is not a leaf)

$$I_q \leftarrow T_q$$

$$S_q \leftarrow J_q$$

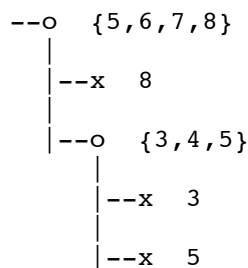
if (node  $r$  is not a leaf)

$$I_r \leftarrow T_r$$

$$S_r \leftarrow J_r$$

### Example 1

Applying the Wagner downpass algorithm to the tree  $(8, (3, 5))$  gives:



Apply the formulae given in Version 1 of the algorithm above to the root of the tree which resulted from the Wagner downpass algorithm. Initially  $I_p = \{5,6,7,8\}$ .

$$T_q \leftarrow S_r \sqcap S_p = \{3,4,5\} \sqcap \{5,6,7,8\} = \{5\}$$

$$T_r \leftarrow S_q \sqcap S_p = \{8\} \sqcap \{5,6,7,8\} = \{8\}$$

$$J_q \leftarrow S_q \sqcap T_q = \{8\} \sqcap \{8\} = \{8\}$$

$$J_r \leftarrow S_r \sqcap T_r = \{3,4,5\} \sqcap \{8\} = \{5,6,7,8\}$$

$$I_p \leftarrow J_q \cap J_r \cap I_p = \{8\} \cap \{5,6,7,8\} \cap \{5,6,7,8\} = \{5,6,7,8\}$$

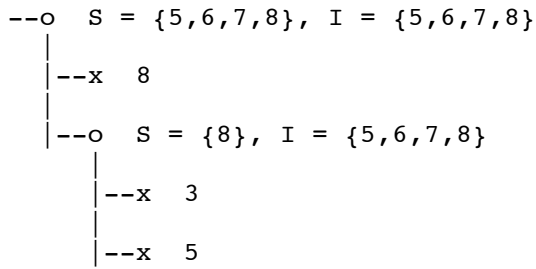
As node  $q$  is a leaf no changes are made.

As node  $r$  is not a leaf

$$S_r \leftarrow T_r = \{8\}$$

$$I_r \leftarrow J_r = \{5,6,7,8\}$$

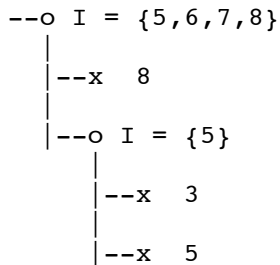
At this stage, the tree has the values:



Now repeat the calculations from the second interior node.

$$\begin{aligned}
 T_q &\leftarrow S_r \cap S_p = \{5\} \cap \{5,6,7,8\} = \{5\} \\
 T_r &\leftarrow S_q \cap S_p = \{3\} \cap \{5,6,7,8\} = \{3,4,5\} \\
 J_q &\leftarrow S_q \cap T_q = \{3\} \cap \{5\} = \{3,4,5\} \\
 J_r &\leftarrow S_r \cap T_r = \{5\} \cap \{3,4,5\} = \{5\} \\
 I_p &\leftarrow J_q \cap J_r \cap I_p = \{3,4,5\} \cap \{5\} \cap \{5,6,7,8\} = \{5\} \\
 &\text{As nodes } q \text{ and } r \text{ are leaves no changes are made.}
 \end{aligned}$$

The final tree is thus:



In this case it is easy to verify the algorithm by calculating the correct uppass values of the two interior nodes. Let these be  $a$  and  $b$  (reading down the diagram above).

We need to minimize  $|a - 8| + |a - b| + |b - 3| + |b - 5|$ .

In general, if  $l \leq h$ , the minimum value of an expression of the form  $|x - l| + |x - h|$  occurs when  $l \leq x \leq h$ . The minimum is then  $h - l$ .

Hence the minimum of  $|a - 8| + |a - b| + |b - 3| + |b - 5|$  occurs when  $3 \leq b \leq 5$  and  $b \leq a \leq 8$ , and will have the value  $2 + (8 - b)$ . Clearly this is when  $b$  is as large as possible, namely 5; we then have  $5 \leq a \leq 8$ .

So in this particular example, the algorithm has given the correct answer.

## Example 2

Consider the tree  $a(1, b(c(d(2, 4), e(5, 6)), f(0, 3)))$ , where  $a$ - $f$  are the interior node values.

Applying the Wagner downpass algorithm gives the tree on the next page.

```

--o = {1,2,3}
|
|--x = 1
|
|--o = {3,4}
|
|--o = {4,5}
|
|--o = {2,3,4}
|
|--x = 2
|
|--x = 4
|
|--o = {5,6}
|
|--x = 5
|
|--x = 6
|
|--o = {0,1,2,3}
|
|--x = 0
|
|--x = 3

```

Hence the minimum number of unit changes is 10.

A computer program based on the the uppsass algorithm given above produces:

```

--o = {1,2,3}
|
|--x = 1
|
|--o = {1,2,3}
|
|--o = {2,3,4}
|
|--o = {2,3,4}
|
|--x = 2
|
|--x = 4
|
|--o = 5
|
|--x = 5
|
|--x = 6
|
|--o = {1,2,3}
|
|--x = 0
|
|--x = 3

```

Is this output correct? It would be extremely tedious to write down the expression for the total number of changes and then try to minimize it by considering all possible cases. A simpler

approach is to write a program to evaluate the changes for all possible values of the interior node values a-f, ranging in each case from 0 to 6, printing out only those sets of values which produce a total change of 10. Such a program results in the output shown below, where a-f are the interior nodes in the order shown in the diagram above.

```
abcdef  
112251  
113351  
114451  
122252  
123352  
124452  
133353  
134453  
222252  
223352  
224452  
233353  
234453  
333353  
334453
```

Hence we have  $a = \{1,2,3\}$ ,  $b = \{1,2,3\}$ ,  $c = \{2,3,4\}$ ,  $d = \{2,3,4\}$ ,  $e = 5$ ,  $f = \{1,2,3\}$ ; precisely the values produced by the program implementing the algorithm I have given above.

### **Demonstration**

A Javascript demonstration of the Fitch and Wagner Parsimony algorithms will be found at [http://www.cs.bham.ac.uk/~pxc/evo/FW\\_Parsimony\\_Demo.html](http://www.cs.bham.ac.uk/~pxc/evo/FW_Parsimony_Demo.html).