Network Value Calculations For The River Barrier Problem

Sarah Read

11/9/2015

1 Introduction

River networks contain barriers, like dams and culverts, that make it difficult for fish to swim from one habitat to another. These barriers decrease the overall value of the river network because they restrict the movement of fish within the system. As conservationists begin to work towards removing barriers from the river network it is important for them to take into account the increase in network value per dollar spent so that barriers are removed in order to open up the most habitat for the fish. This document will outline the algorithms for calculating the total network value for a given river network.

The river network is modeled as a rooted bidirected tree. Each node in the tree represents a habitat in the river network. The bigger the node value the larger the habitat. Each edge has an associated probability which represents the probability that a fish could swim from one habitat to the other (see figure 1). If the probability between 2 nodes is 1 in both directions this represents the case when fish can swim freely between both habitats, which occurs only if

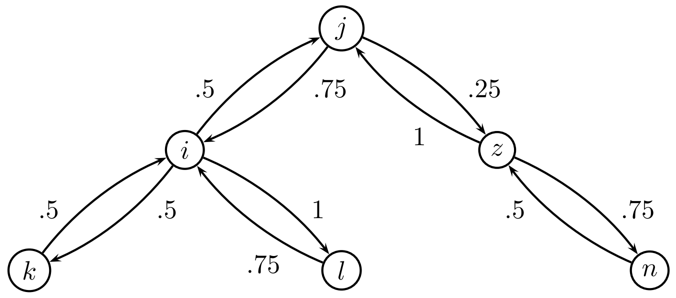


Figure 1: Example of a bidirected tree representing a river network. Each edge is labeled with the probability of moving from the node on one side to the node on the other.

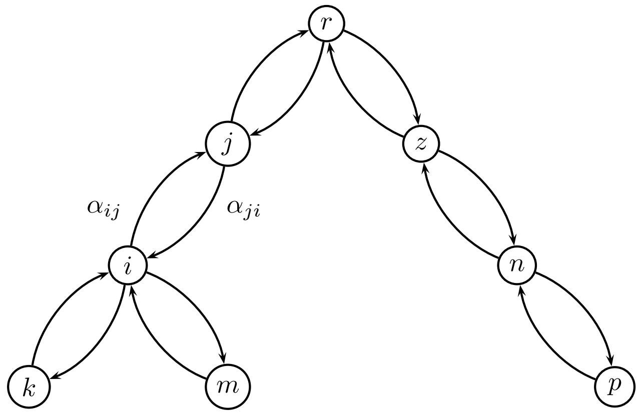


Figure 2: Example river network with α messages between i and j labeled

the river forks. Otherwise the probability will be less than 1, representing the case when two habitats are divided by a barrier (such as a dam or culvert). Note that even though we treat the tree as being rooted the node that is selected as the root is arbitrary. No matter which node is treated as the root the total network value will be the same.

In order to compute the total network value the calculation is broken up into 3 pieces:

- α_{ij} Value: The sum of all paths from nodes in the subtree rooted at i that end at i, where j is the parent.
- β_{ij} Value: The sum of all paths starting at i and ending at a node in the subtree rooted at i, where j is the parent.
- γ_{ij} Value: The sum of all paths between any 2 nodes in the subtree rooted at i, where j is the parent.

For each value we must calculate the value each node passes to it's parent and what value it will receive from it's parent. Therefore for each node we will calculate α_{ij} (alpha to parent of i), α_{ji} (alpha from parent of i), β_{ij}, β_{ji}, γ_{ij}, and γ_{ji}.

We do not calculate these values for the root because using the message passing model there is one set of messages per edge, so we have n-1 messages and n nodes. We are mapping the edge values onto the nodes for the calculation, so we choose to exclude 1 node, and for convenience we exclude the root.

Figure 2 depicts a tree rooted at r. Nodes i and j are labeled and the edges connecting i and j are labeled to show the direction of α_{ij} and α_{ji}, the α value from i to its parent and the value of α from i's parent to i respectively. Message passing for β_{ij} and γ_{ij} is the same.

2 α Calculations

α_{ij} finds the value of all paths starting in the subtree rooted at i and ending at i, where neighbor j is the only neighbor of i not in the subtree.

$$\alpha_{ij} = v_i + \sum_{k \in N(i) \setminus j} \alpha_{ki} p_{ki}$$

3 β Calculations

β_{ij} finds the value of all paths starting at i and ending in the subtree rooted at i, where neighbor j is the only neighbor of i not in the subtree.

$$\beta_{ij} = v_i + \sum_{k \in N(i) \setminus j} p_{ik} \beta_{ki}$$

4 γ Calculations

γ_{ij} finds all of the paths between any pair of nodes in the subtree rooted at i, including those that start and end at i, where neighbor j is the only neighbor of i not in the subtree.

$$\gamma_{ij} = \sum_{k \in N(i) \setminus j} \gamma_{ki} + \sum_{k,l \in N(i) \setminus j, k \neq l} \alpha_{ki} p_{ki} p_{il} \beta_{li} + v_i * \beta_{ij} + v_i * \alpha_{ij} - v_i^2$$

The first term finds all γ_{ki} values for all children of i, which takes care of all paths that start and end in the tree rooted at one of i's children. The second term finds all paths that start in a subtree rooted by one child of i and end in a subtree rooted by a different child of i (see figure 3). The third term finds all paths starting at i and ending in the subtree of i (including the path that starts and ends at i). The fourth term finds all paths that start in the subtree of i and end at i (including the path that starts and ends at i). The last term subtracts the value of the path starting and ending at i since we counted it twice, once in the third and once in the fourth term.

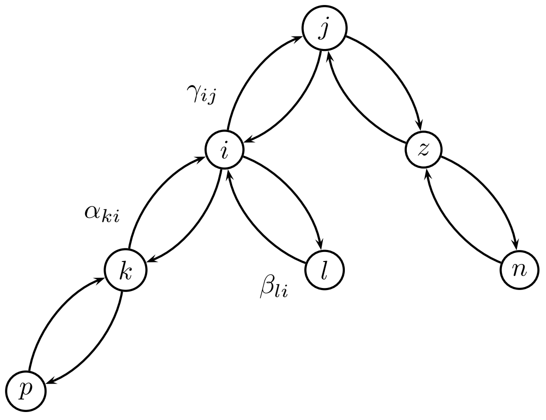


Figure 3: Depiction of the second term of the γ_{ij} calculation

5 Total Network Value Calculation

The total value of the network is a sum of all paths between all pairs of nodes in the network.

$$NetworkValue = \gamma_{ij} + \alpha_{ij}p_{ij}\beta_{ji} + \alpha_{ji}p_{ji}\beta_{ij} + \gamma_{ji}$$

The first and last terms sum the value of all paths within the subtrees of i and j. The second and third terms find the value of all paths starting in one subtree and ending in the other. This accounts for all possible paths between any 2 nodes in the tree.

6 Implementation Overview

All algorithms were implemented using recursive functions and dynamic programming. For each node i we calculate α_{ij}, α_{ji}, β_{ij}, β_{ji}, γ_{ij}, and γ_{ji}. We store this information in 6 different arrays; α, β, and γ to parent and α, β, and γ from parent. Since each node is assigned an id 0 through n-1 (where n is the number of nodes in the tree) the value of each calculation is stored in the index of the array corresponding to it's id.

As noted earlier, the to parent and from parent calculations are done based on an arbitrary root. The root is not included in calculations, and therefore is given the id n-1 and our arrays have length n-1. The selection of the root does not matter, and the same network value will be found for any root choice.

7 Data Types

The 2 main data types used in our implementation are the Node and Tree classes.

The Node class contains the following properties:

- nodeId: Unique identifier for the node between 0 and n - 1.
- nodeLabel: Label on the node, does not have to be unique.
- val: The value of the habitat represented by the node.
- coordinates: The coordinates of where the node is on the map.
- neighbors: A list of all neighbors of the node. No parent-child relationship is defined here.

The Tree class contains the following properties:

- numNodes: The total number of nodes in the tree.
- nodeList: A list of all of the nodes in the tree.
- root: An arbitrary node in the tree.
- edgeProbs: A 2 dimensional array that holds the probability of traversing from one node to another. Each row in the array is formatted as [startNode, endNode, probabilty].

The data comes in as JSON and is then transformed into Nodes and Trees. The JSON file lists:

- numNodes: Total number of nodes in the tree.
- nodeLabels: Array of labels for all nodes.
- vals: Array of values for all nodes.
- coords: Array of number pairs corresponding the the x and y coordinates of each node.
- probBtwNodes: 2 dimensional array where each index is an array formatted as [startNode, endNode, probabilty].

Each node can be created by getting the information from the i^{th} index of the JSON object. See Figure 4 for an example JSON object.

```
var example_tree = {
    numNodes : 3,
    nodeLabels : ["a", "b", "c"],
    vals : [3, 1, 2],
    coords : [
                [3,1],
                [2,3],
                [4,3]
             ],
    probBtwNodes : [
                    ["a", "b", 1],
                    ["b", "a", .5],
                    ["a", "c", .5],
                    ["c", "a", .25]
                   ]
};
```

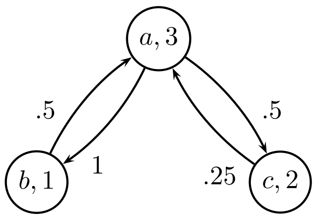


Figure 4: Example JSON representation of a basic river network and the corresponding tree. Each node in the tree shows the label and the value of the node. The edges are labeled with the corresponding probability.

8 Data Transformation

Data was originally saved with the habitats being represented as edges and nodes being barriers or splits in the network. The data was transformed to switch these values and resemble the above JSON.

9 α Implementation

First we find all of the to parent values. Since each α_{ij} value depends on the α value of i's children, we implemented a function that starts at the root and recursively calls the children of the given node. The base case for a leaf node is:

$$\alpha_{ij} = v_i$$

Then after the recursive call to node i's children the value is computed as:

$$\alpha_{ij} = v_i + \sum_k \alpha_{ki} p_{ki}$$

Where v_i is the value of the node i and the sum is over all children of i.

Figure 5 shows and example of the final alphaToParent array that would be calculated. In this example α_{ba} is found the following way:

$$\alpha_{ba} = v_b + \alpha_{db} p_{db} + \alpha_{eb} p_{eb}$$

Or using the alphaToParentArray:

$$alphaToParent[0] = v_b + alphaToParent[2] * p_{db} + alphaToParent[3] * p_{eb}$$

Next we have to find all of the from parent values. These values will be stored in an alphaFromParent array, similar to what we did for alphaToParent. We start by calculating all of the from parent values for the children of the root:

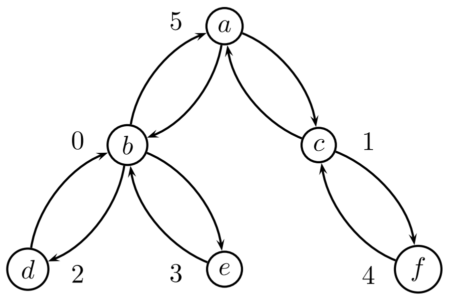
$$\alpha_{ji} = v_r + \sum_{k \in N(r) \setminus i} \alpha_{kr} p_{kr}$$

Where v_r is the value of the root and the sum is over all children of r excluding i.

Using the tree from figure 5 we can find alphaFromParent for node b:

$$alphaFromParent[0] = v_a + alphaToParent[1] * p_{ca}$$

If the root had 3 children the right hand side of the equation would have 2 terms that get values from alphaToParent.



alphaToParent:

α_{ba}	α_{ca}	α_{db}	α_{eb}	α_{fc}

Figure 5: alphaToParent array example based on the given graph. Node labels are shown inside the nodes, the node ids are shown next to the nodes.

Finally we have to find the from parent values for all other nodes in the tree. The general formula is:

$$\alpha_{ji} = v_j + \alpha_{lj} * p_{lj} + \sum_{k \in N(j) \setminus i} \alpha_{kj} p_{kj}$$

Where l is the parent of j and the sum is over all children of j excluding i. The first term is the value of j. The second is α from parent of j time the probability of going from j's parent to j. The sum finds all children of j that are not i and adds their α to parent value times the probability of going from the child k to j. This is because when finding α we must find all paths in the subtree rooted at j that start in the subtree and end at j. The children of j that are not i are in j's subtree and therefore must be included. We use the α to parent value for the children of j because of the direction of the message passing.

Again, looking at figure 5 we can find alphaFromParent for node d:

$$alphaFromParent[2] = v_b + alphaFromParent[0] * p_{ab} + alphaToParent[3] * p_{eb}$$

10 β Implementation

The β_{ij} calculation is very similar to α_{ij} except for that the probabilities go the other direction because we are calculating paths starting at the root of the subtree instead of ending at the root of the subtree.

11 γ Implementation

The γ_{ij} calculation relies on both α_{ij} and β_{ij}, so it must be calculated after these two arrays. Since γ_{ij} depends on the children on i we write the base case as:

$$\gamma_{ij} = v_i * v_i$$

Then after the recursive call to the children of i the value of the gammaToParent array is computed as the following for node b from figure 5:

Values related to node b:

$$gammaToParent[0] = alphaToParent[0]*v[0]+betaToParent[0]*v[0]-v[0]*v[0]$$

Values related to the children of b:

$$gammaToParent[0]+ = alphaToParent[2]*p_{db}*p_{be}*betaFromParent[3]+gammaToParent[2]$$

$$gammaToParent[0]+ = alphaToParent[3]*p_{eb}*p_{bd}*betaFromParent[2]+gammaToParent[3]$$

After doing this for all nodes i we must compute the gammaFromParent array. Again for node b:

$$gammaFromParent[0] = gammaToParent[5] - v[5] * v[5]$$

$$gammaFromParent[0]+ = v[5]*alphaFromParent[0]+v[5]*betaFromParent[0]$$

Again, like with the alphaFromParent value for d we have to take e into account and get the following value:

Values related to node d:

$$gammaFromParent[2] = gammaFromParent[0] - v[0] * v[0]$$

$$gammaFromParent[2]+ = v[0]*betaFromParent[2]+v[0]*alphaFromParent[2]$$

Values related to the neighbors of d:

$$gammaFromParent[2]+ = gammaToParent[3]+alphaToParent[2]*p_{eb}*p_{ba}*betaFromParent[0]$$

$$gammaFromParent[2]+ = alphaFromParent[0] * p_{ab} * p_{be} * betaToParent[3]$$