Subscriber ranking based on their approach

Kashyap R Puranik
Arjun N Bharadwaj
Joseph Joseph

September 8, 2010

Introduction:

Operators need to rank their subscribers for different reasons based on various parameters. The expenditure alone cannot be used to decide the rank of a group of subscribers because the network is a complex concept and a lot of nodes in the network influence the behaviour of a given node. It is therefore advisable to view the problem in other ways. The data available is large, it consists of the CDR data for all subscribers using which the ranking has to be done. Thus a parallelisable approach has to be taken to deal with the data effectively. Also division of the data into clusters is very essential in case of non-linear algorithms.

Basic modeling

The whole telecom network can be modeled as a weighted undirected graph as follows. Each subscriber in the network can be modelled as a node. There are edges between two nodes if the cumulative revenue generated by the connection, crosses a certain treshold. The weight of the edge can be made inversely proportional to the revenued generated by the connection between the two subscribers. It is important to note here that the graph is a sparse graph and an incidence matrix will not be necessary for the storage of the graph.

G = (V, E, W) where

V $\longrightarrow$ The set of vertices, corresponding to the subscribers |V| = N, the number of subscribers

T $\longrightarrow$ The threshold, the minimum revenue required to be genrated from a connection

E $\longrightarrow$ The set of recognised connections is the set {e | e = (u, v)ˆu, v ∈V ˆConnectionValue (u, v) $\geqslant$ T }

W $\longrightarrow$The weight function E→R, e∈E, W(e) = $\frac{1}{ConnectionValue(e)}$

ConnectionValue() function, E→R is a function that will be defined later which will attempt to quantify the revenue generated by the calls made, messages sent between two users.

Note: We are modeling the network as an undirected graph because we treat calls made from 'A' to 'B' as just a call between 'A' and 'B' ignoring the caller and the callee. We can later see that this way of modelling is as good as modelling the network as a directed graph.

Note: The pruning of the graph based on a treshold is to avoid calls that were accidental or calls between two people that are extremely rare. Such calls or messages can be ignored. Also the treshold can be used to restrict the number of edges in the graph.

High level implementation

We provide the high-level implementation of the algorithm here and show that it is a parallelisable one. The graph is a sparse graph and the number of edges depends on the parameter T that is the threshold. We store the following two details for each node in the graph.

- List of neighbours for each node
- Weight of each edge in the graph

The number of nodes and edges in the graph are enormous. We can use distributed storage in hash tables residing on multiple computers to store the data. We need two hashtables, one for the list of neighbours and the other for storing the weight of each edge. This allows us to access the data in constant time.

hashVertex() function takes a vertex and returns the location where the list of neighbours of a given vertex is stored.

hashEdge() function takes an edge and returns the location where the details of an edge is stored.

Creation of the graph

The following algorithm goes through the CDR list and constructs the graph. Since the data is stored on multiple computers, the algorithm is also parallelisable.

The steps of the algorithm is as follows

Part1: Scan-stage (Input -> CDR_list)

```
for each CDR in CDR_list do:
    value := getValue(service, duration, cost)
    addNeighbour(caller, callee), addNeighbour(callee, caller)
    addValue(caller, callee, value)
```

[Note: addNeighbour uses the hashVertex function to get the location of the neighbours of a vertex and adds the second parameter to the list of neighbours

addValue function uses the hashEdge function to get the location of the edge corresponding to the two vertices and increments the value of the the given edge]

Part2: Prune edges (connectionValue < treshold)

```
for each CDR in CDR_list do:
```

```
value := getValue(caller, callee)
if value < T: dropEdge (caller, callee)
```

[Note: getValue function again uses hashEdge function and gets the value of the edge as calculated in Part1.
dropEdge prunes the edges whose values are less than a treshold.]

Since the iteration is order independent, the load can be distributed amongst different computers and parallelised.

Now that the graph corresponding to the network with the given treshold has been obtained, we can attempt to rank the nodes based on a few considerations.

In the following sections, we introduce a few concepts that will be useful in formulating a ranking scheme.

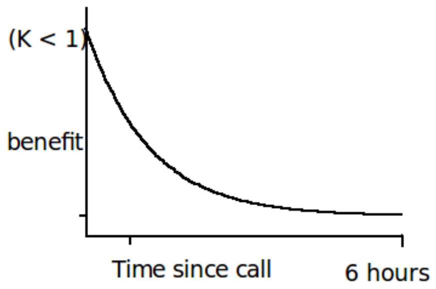
Call causality

A call from subscriber 'A' to subscriber 'B' and hence the flow of information may cause subscriber 'B' to call other people say for example 'C'. This may happen coincidentally or it may be a frequent pattern. If such a pattern is observed, then the relative importance of the connection 'A' to 'B' increases as it is responsible for more calls in the network.

The phenomenon can be quantified as follows:

Suppose there exist 2 CDRs as described in the table below, (details ommitted)

Num	Caller	Callee	Time	Cost
N_1	A	B	T_1	C_1
N_2	B	C	T_2	C_2



Call cause benefit should be attributed to the caller A assuming he caused the call.

$V = K_1 \ C_2(e^{k_2(T_2-T_1)})$

$V \rightarrow$ the benefit that caller 'A' should obtain because of causing the call

$K_1 \rightarrow$ A constant < 1, which decides the percentage of benifit that 'A' gets

$k_2 \rightarrow$ Decay factor, this should ensure the benefit decays if the time difference is more than a couple of hours.

From the above formulation, it is evident that the closer the calls are, more the benefit is.

This phenomenon may occur coincidentally but, such occurances won't contribute much to 'A'. Frequent occurances will however contribute because they keep getting added to the benefit of 'A'. Also, there is a low probability that this phenomenon occurs regularly but is not causal. Such cases are few in number and can be ignored in a random sample with large number of CDRs.

ConnectionValue

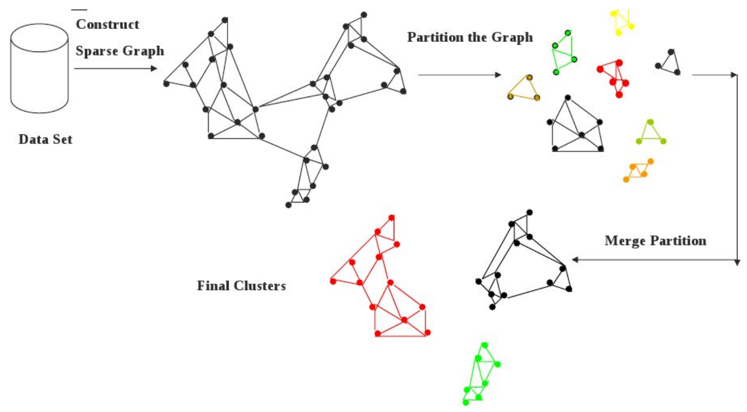
We haven't yet defined the connectionValue function. The function returns the expected value of a connection rather than the expenditure because of the connection. The concept of call causality can be used to boost the expenditure value and make it higher for subscribers who cause calls. The calculation of connectionValue is NP-hard and hence we use an approximate algorithm for calculation. The algorithm for ConnectionValue calculation is as follows.

```
Maintain a queue of CDRs consisting of CDRs in the past 6 hours. ⟶CDR_queue
k = DiminishingFactor
Repeat till convergence:
    for each CDR in CDR_list:
    ....enqueue the CDR_queue with CDR
    ....if there exists CDR$^1$in the queue for which transitivity holds
    ....CDR$^1$= (a⟶b) and CDR = (b⟶c)
    ....add k * (CDR.cost) to the connection (a ⟶b)
    k = k * DiminishingFactor
```

where k is a factor < 1. At the end of the iterations we will have an approximate value of the ConnectionValue.

Graph Clustering

CHAMELEONa widely used open graph clustering algorithm can be used to cluster the huge graph that we have to deal with it independently. The algorithm involves construction of a sparse graph as shown above, partitioning the graph and merging then close partitions to get final cluster.



Central nodes

In a naive way, we can call the central nodes as the nodes towards the centre in a visible cluster. To detect the centre of a cluster however we have to use a few algorithms that use the connection values. The central nodes are the ones that have a minimum sum of distances from all other nodes. So centrality can be defined as

C(u) = $\sum_{\forall v \in Cluster(u)} distance(u, v)$

The more the value of C(u), the farther away it is from the centre. Distances can be calculated using Fleury's algorithm as we know the ConnectionValue for each edge.

Random walks

Consider a walk on the graph in a particular cluster, starting from a random node. The next node on the random walk will be one of the neighbours of the given node with a probability that increases with the value of the connection.

The transition probability is given by.

T(u, v) = $\frac{ConnectionValue(u,v)}{\sum ConnectionValue(u,w), w \in Neighbourhood(u)}$

A random walk is simulated for sufficient amount of time and the number of times each node is visited, is counted. The count is usually high for the nodes in the central region for a cluster. The edges of the cluster are less frequently visited. This can be analysed as follows, the more the number of neighbours a node has, the more likely is its increment of count. More the value of a connection, more likely it is picked. Thus nodes around the centre will get a high score as opposed to the ones at the edge of a cluster.

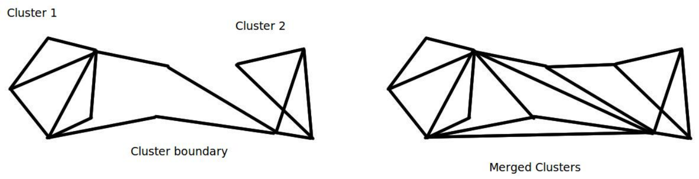
The random walk algorithm is as follows

Start at centre of cluster

```
Count(u) = 0 $\forall u \in V$
repeat N times, till convergence of values:
    Transit to neighbour 'n' with probability T(u,n)
    Count(n) = Count(n) + 1
```

Bridge nodes

Other than the central nodes, there are other nodes that are important such as bridge nodes, which form bridges between two clusters. They are important because they are the means by which information flow occurs from one cluster to another. Although they don't have direct economic significance, their importance can be seen when they cause cluster merging. This phenomenon occurs when say two groups of friends (two clusters) have a few mutual friends and the mutual friends (intra cluster bridges) bring the two groups together causing interaction and possible formation of new connections in the form of calls and messages. The above is a simple example of cluster merging but clusters merge on large scales in the world of telecom.



Ant Algorithms

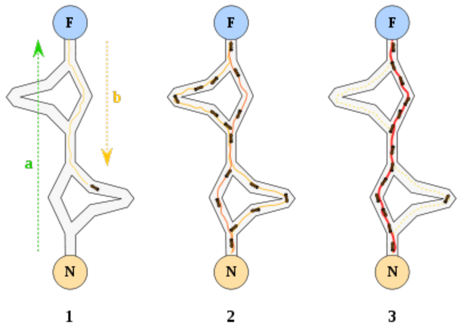
Ants follow a unique way of finding the shortest path to the food. They lay pheromones on their train to the path. If there is a short path of length 'l_1'and a large path of length 'l_2'. ($l_1 < l_2$) It takes time $t_1$and t_2 ($t_1 < t_2$) for the ants to go along the two paths. If there is a continuous flow of ants, more number of ants would have gone through the shorter path than the longer path and hence, the pheromone concentration would be higher on the shorter path than. Ants tend to stick to the path with higher pheromone concentration and stick to the shorter path.

This principle can be applied to mark pheromone on intracluster paths. An ant colony simulation where ants try to find paths between clusters, can be performed on the network. The information flow bridges will have a high pheromone concentration as they occur in all the short intra cluster paths.

The algorithm is as follows

```
for each cluster C1, C2: $\longrightarrow$parallelisable step
    /* start a flow of ants from the centre of C1 to the centre of C2 and back */
    while population is not saturated:
    ProduceAnt()

Ants are independent and work by themselves
Sample with probability p (explore / followTrail)
if explore, moving towards C2:
    transit to neighbour 'n' from current node u with probability $P = \frac{ConnectionValue(u,n)}{\sum_{\forall v \in Neighbours(u)} ConnectionValue(u,v)}$
else if followTrail, moving towards C2:
    choose the next neighbour 'n' with probability $P = \frac{PheromoneConcentration(n)}{\sum_{\forall v \in Neighbours(u)} PheromoneConcentration(v)}$
else if moving back to C1:
    chose the next neighbour 'n' with probability $P = \frac{PheromoneConcentration(n)}{\sum_{\forall v \in Neighbours(u)} PheromoneConcentration(v)}$
    PheromoneConcentration(n) = PheromoneConcentration(n) + 1
```



Ranking

Node Ranking for vertices in each cluster

Now that we have introduced the above ideas, we can formulate a ranking scheme using the ideas.

CentralityRank based on Centralness of the nodes in a cluster.

Lower the centrality index (viz. closer to the center), better the rank.(R_1)

ReachabilityRank based on the count in random walk model.

Higher the count, better the rank. (R_2)

Intra Cluster Connectivity Rank for clusters on the border, based on pheromone concentration.

Higher the pheromone concentration, better the rank (R_3)

Based on the ranks R_1, R_2 and R_3, the overall rank can be decided.

a_1* Centrality(u) - a_2* Reachability(u) - a_3BorderScore(u)

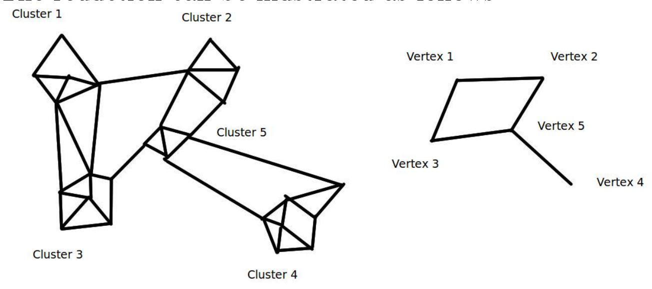
where a_1, a_2 and $a_3$are co-efficients that are decided according to requirements. We get the $R_v$for vertex v from that.

Ranking for Clusters

The huge graph can be shrunk and each cluster can be considered to be a vertex and the importance of the newly obtained vertex is the sum of the importance of all the vertex in the cluster.

We can now perform cluster ranking on similar lines of vertex ranking.

The reduction can be illustrated as follows



Overall Ranking

For each vertex we know the ranking within the cluster C, R_v for score S_v. We also know the score of the cluster C R_c and score S_c. We can obtain an overall rank of a given vertex R by combining the scores to get an overall score.

$S = S_c * B + S_v$

Page ranking (An alternate solution to the problem)

This section highlights the similarity between the telecom network and the world wide web. The documents in the web are ranked using page ranking. Documents are interconnected using hyperlinks in analogy with subscribers being connected to each other with connections in our model. The importance of a document is decided by the number of incoming links and the weights of the linking documents. In a directed graph version of the same problem, the importance of a subscriber is decided by the number of connections to the subscriber and the value associated with the subscriber in each connection. The algorithm has the element of circularity involved in it and hence the initial values for the subscribers can be equal to their expenditure. The bootstrapping iteration will update the values of each node based on the neighbours of the node and the iteration can end when convergence is seen.

The algorithm is as follows

```
for n in V:
    Value(n) = Expenditure(n)

factor = DiminishingFactor
```

```
loop till convergence of Value(n) $\forall$ n:
for n in V:
    Value(n) = Value(n) + factor* $\sum_{\forall u \in Neighbours(u)}$ (Value(u))
    factor = factor * diminishingFactor
```

Where the factor contributed by each neighbour diminishes with each iteration.

The overall algorithm:

1. Initialize the network in the form of a graph and remove edges below the threshold.
2. Divide the nodes of the graphs into clusters.
3. Algorithm 1:
    (a) Find the centrality index of each node. Nodes close to the centres of the cluster are better since they are connected to more nodes, through shorter distances.
    (b) Run random walk simulations and find out how often a node is visited - This will give us the reachability which is a measure of how much information flows to that node.
    (c) Run ant algorithms to find out which node is most frequently visited when information flows from on cluster to another. This is a measure of how much information flows through that node between two clusters.
    (d) Based on these three indices come up with the rank for the user
4. Algorithm 2:
    (a) Initialize each node's (user's) rank based on the expenditure he makes viz., his value.
    (b) Increment his value by multiplying his neighbour's values by a diminishing factor and adding it to his own.
    (c) Repeat (b) till convergence

Parallelisation of the implementation

Note that both algorithms involve dividing the entire network into clusters. Each step of both algorithms considers users of a particular cluster (or a pair of clusters in the case of the ant algorithm) and then iterated over all the clusters in such a way that the computations related to one (or pair of) cluster are independent of those relating to the others. Also, in the case of Algorithm

I, each step of the algorithm is independent of the others. This means that each step of the algorithm for each cluster (or pair of clusters) can be run on a separate processor. In this way, the huge graph is broken down into a large number of clusters each of which and at each step can be worked on separately.

Conclusion

In this report we have gone over two algorithms, one based on the idea that the information is quantized and everytime a call is made, there is a transfer of information and the other algorithm based on Page Ranking Algorithm. In the first algorithm, we try to identify the nodes having the major sway on the flow of information and have made the process both efficient and parallel. Unless real life experiments are conducted, it is difficult to say which algorithm works better.