Rankmaniac Report

Team: AlanAldaVista

Sean Dolan|Jan Van Bruggen|Brad Chattergoon

2015

Introduction:

One of the important applications of Network Science is found in the search engine domain. Network science can be used to determine what results are relevant to a specific query by ranking the pages related to that query.

This process can be simulated using a network graph. A network graph can be extracted from some data set and an adjacency matrix, P, constructed for the graph. One of the properties necessary for network analysis to determine page rankings is that the graph has to be strongly connected. To simulate this a new graph, G, is constructed by introducing an edge from every node to every other node with the existing graph being weight as α, and the simulated contributions being weighted as $1-\alpha$ as follows:

$$G = \alpha \times P + \frac{1-\alpha}{n} \times \begin{bmatrix} 1 & \cdots & 1 \\ \vdots & \ddots & \vdots \\ 1 & \cdots & 1 \end{bmatrix}$$

Page rank is then calculated using the adjacency matrix outlined above.

Algorithm:

Algorithm Overview:

The AlanAldaVista team opted to use an algorithm stemming directly from the page rank formula.

$For\ a\ node\ i, the\ page\ rank\ at\ time\ t, r_i(t), is\ given\ as\ follows:$

$$r_i(t) = \sum_{j \in N(i)} \frac{r_j(t-1)}{d_j out}$$

$where\ N(i) is\ the\ set\ of\ neighbors\ of\ node\ i\ with\ an\ edge\ leading\ into\ i.$

The algorithm makes use of the mapper function to determine the contribution from each neighbor to a specific node. The collector then aggregates the contribution from each neighbor into a list of the individual contributions with the node under consideration as a key. The reduce function is called to sum the contributions from each neighbor to the node in order to produce the page ranking of that node for the t'th iteration.

Algorithm Pseudocode:

$Begin\ PageRankMap(NodeID, r_{NodeID}(t-1), r_{NodeID}(t), ListofOutLinkedNodes)$
$pagerank = r_{NodeID}(t);$
$friends = ListofOutLinkedNodes;$
$for\ friend\ in\ friends:$
$contribution = \alpha \times \frac{pagerank}{len(friends)};$
$emit(friend, contribution);$
$End\ PageRankMap$

$Begin\ Collector(Emitted\ Data)$
$for\ friend\ in\ Emitted\ Data:$
$NodeID = friend;$
$output = (NodeID, ListofContributions);$
$End\ Collector$

Note: This step was done automatically by the map-reduce framework, but is included to help the reader conceptualize the process.

$Begin\ PageRankReduce(key, ListofContributions)$
$return(NodeID, Sum(ListofContributions) + 1 - \alpha)$
$End\ PageRankReduce$

$Begin\ ProcessReduce(ListofPageRanks)$
$If\ Num\ of\ Iterations < 15$
$Return\ New\ Pagerank\ Data\ for\ Processing$
$Else$
$Determine\ Top\ Twenty\ Nodes;$
$Return\ Top\ Twenty\ Nodes;$
$End\ Process\ Reduce$

Note: ProcessMap was not used in the implementation of the algorithm and functioned as an identity function.

Discussion of Algorithm:

The team adopted an Occam's Razor approach to the problem: more complicated solutions may ultimately prove correct, but—in the absence of certainty—the fewer assumptions that are made, the better. Because the data sets for processing were not known in advance, no assumptions about structure were possible and without any such knowledge it was difficult to optimize to take advantage of any specific characteristics of the network graph. The team believed that a general implementation of the page-ranking algorithm would be best positioned to handle any type of data set or network structure.

Additionally, the page-ranking computation and map-reduce functions scaled in operations linearly. This was a valuable characteristic that the team sought to preserve in the algorithm. For this reason complicated optimizations were omitted in preference for pythonic improvements in the code, which would give operational benefits instead of algorithmic benefits.

Tolerance checks to determine the number of iterations were also omitted by relying on a property of convergence defined by the degree of desired error, which will be discussed in the "Technical Considerations" section of the report. Avoiding a check for tolerance allowed the program to avoid costly computations in sorting which contributed to an optimized run-time.

Technical Considerations:

The true page rank of the network graph is given by a row vector, π, such that the equation $\pi = \pi P$ is satisfied where P is the transition/adjacency matrix for the network. The goal of the algorithm is to iterate with convergence to π.

One of the important questions under consideration was "how close to π does the algorithm have to iterate to before the top twenty nodes are determined in order?" In other words, what degree of error from the true page ranks is tolerable with respect to obtaining an accurate ranking?

To answer this question, the team turned to the heavy-tailed property of network graphs. According to this framework, the page ranks of the nodes would follow a heavy tailed distribution that would obey the "catastrophe principle"; the page rank of the graph would be distributed in much higher concentrations in the higher ranked nodes than would be found in the lower ranked nodes. Since the order of magnitude of the page ranks of the top twenty nodes would be on the order of 10^0 or larger, the team concluded that an error from π (denoted by τ) of 0.01 would be sufficient to correctly rank the top twenty nodes as this τ produces a negligible relative error in the page ranks of the top twenty nodes.

Since the number of iterations executed before declaring convergence partly determines the run time of the program, the natural follow-up question is, "how

many iterations are required to obtain a τ of 0.01?" Fortunately this question is answered by the following equation:

$$Number\ of\ Iterations = \frac{\log(\tau)}{\log(\alpha)}$$

where α is the factor used to simulate a strongly connected graph.

In the project guideline, α is specified as 0.85. According to this formula the number of iterations approximates to 15.

Summary of team member contributions:

Sean Dolan: Implemented algorithm in python script and debugged the written code. Provided the insight on the number of iterations required to obtain a τ distance from the true page rank. Implemented any optimizations determined during brainstorming sessions.

Jan Van Bruggen: Responsible for testing the program. Implemented several testing scripts and procedures in python. Executed the use of these procedures and scripts both locally and remotely to the Amazon Web Service. Participated in writing and debugging code.

Brad Chattergoon: Assisted in developing the algorithm implemented. Participated in brainstorming sessions. Researched potential optimization implementations for the algorithm. Took lead on constructing the project report.

Proposed Improvement on Algorithm:

One of the improvements considered was omitting nodes from the graph that were void of any in links and out links. This would have removed the extra computations involved with nodes which do not affect the page-rank of the nodes on the graph[1]. Due to the extra level of computation in using length analysis or string comparisons on the input data, this improvement was not implemented

[1] http://www.zdnet.com/article/3-simple-ways-to-optimize-for-google-pagerank/