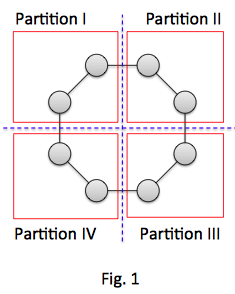
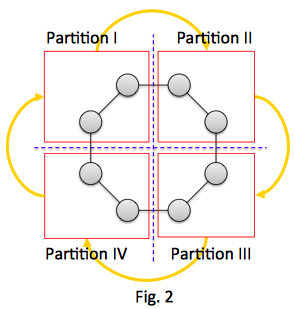
Vector Comparison Documentation

Algorithm Overview

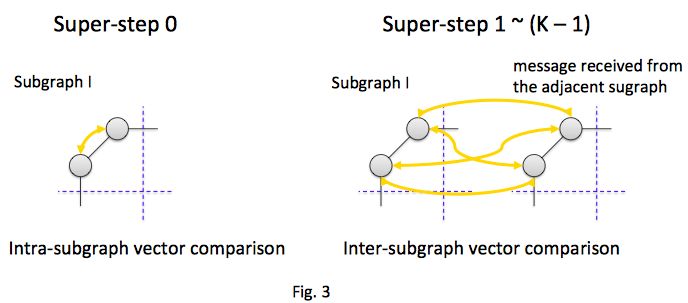
The goal of the vector comparison application is to perform Pearson Correlation on every pair of all the input vectors. Firstly, the vectors are translated into a graph-representation and stored in GoFS.

To be more specific, the input vectors are mapped into vertices, which form a circular graph. This graph is then partitioned into K pieces where each piece contains one **subgraph** and resides on one machine, as shown in Fig. 1.

As a Bulk Synchronous Processing (BSP) model, the entire computation is divided into a series of **super-steps**.  For the 0th super-step, each sub-graph would perform two operations, to read the vectors in its vertices and perform **intra-subgraph vector comparison**. Essentially, the intra-subgraph vector comparison performs comparisons on all pairs of vectors in a subgraph. After the 0th step is completed, each subgraph sends its own vectors to the successive subgrap, shown in Fig. 2.

After each subgraph receives the message from its predecessor, it performs **inter-subgraph vector comparison**. That is, each subgraph compares two sets of vectors, one being its own and the other being those it just received from its predecessor. Notice that since the comparison within each subgraph has been obtained in the 0th super-step, the inter-subgraph vector comparison only needs to evaluate the comparisons whose participant vectors are in different subgraphs. After this comparison, each sub-graph further passes the received vector set to the successor, forming a one-step circulation along the graph. The entire procedure is summarized in Fig. 3.



Visibly, all the vector sets would make a round-trip along the circle after (K – 1) super-steps, and finish the goal to compare every pair of all the input vectors. The pseudo-code is shown in the following.

Pseudo-code

public class VectorComparison extends GopherSubGraph {

      ArrayList<double[]> myVectors;

     @Override  
     public void compute(List<SubGraphMessage> subGraphMessages) {   
          myVectors = extractMyVectors();  
  
          if(getSuperStep() == 0) {   
               // 0th Super-Step : Calculate the intra-subgraph vector comparisons  
               int numOfMyVectors = myVectors.size();   
  
               // Intra-subgraph vector comparison   
               for(int i = 0; i < numOfMyVectors; ++i) {  
                    for(int j = i + 1; j < numOfMyVectors; ++j) {  
                         double pCorr = pearsonCorrelation(myVectors.get(i), myVectors.get(j));   
                         output(pCorr);

                    }  
               }  
  
               // Send the vector-set to the next subgraph  
               SubGraphMessage<String> msg = new SubGraphMessage<String>(myVectors.getBytes());   
               sentMessage(successiveSubgraph, msg);  
     } else {  
               // 1st ~ (K-1)th Super-step   
               SubGraphMessage<String> msg = subGraphMessages.get(0);  
  
               // Convert the message to ArrayList<double[]>  
               ArrayList<double[]> otherVectors = convert(msg.getData());  
  
               // Start to do the calculation  
               int numOfMyVectors = myVectors.size();  
               int numOfOtherVectors = otherVectors.size();  
  
               // Inter-subgraph vector comparison   
               for(int i = 0; i < numOfMyVectors; ++i) {  
                    for(int j = 0; j < numOfOtherVectors; ++j) {  
                         double pCorr = pearsonCorrelation(myVectors.get(i), otherVectors.get(j));   
                         output(pCorr);  
                    }  
               }  
  
               if(getSuperStep() == (numberOfPartitions - 1)) {  
                    voteToHalt();  
               } else {  
                    // Send the received vector-set to the next subgraph  
                    SubGraphMessage<String> msg = new SubGraphMessage<String>(otherVectors.getBytes());   
                    sentMessage(successiveSubgraph, msg);  
               }  
          }  
     }  
}

How to Install and Run Vector Comparison

In this section, we describe how to install and run the vector comparison application we provided with the code.

In the code, it is assumed that the graph has four partitions, and they are distributed on four different machines where containers are deployed.

Building Application

* You can build the application using $mvn clean install command which will create vect-comp-0.9-SNAPSHOT.jar
* Install application in floe (see Readme.pdf)

Deploying the workflow

* Once floe is running, you can deploy the workflow defined in vector-comparison/graph/vect-comp.xml in floe by running edu.usc.pgroup.goffish.gopher.sample.client.GraphStart in the vector-comparison source folder we have provided. Pass command line arguments : coordinator Host name and vect-comp.xml path
* For a sample output, please refer to Readme.pdf
* Then run the edu.usc.pgroup.goffish.gopher.sample.client.Client with command line arguments HostName to connect , data port and control port which will initiate the workflow.
* After the computation, some .txt files would be generated at containers. For each line in these .txt files should have the following format

X,Y Z, where X and Y refer to the compared vectors and Z stands for the Pearson Correlation between these two vectors