**Glowworm Swarm Optimization**:

It deals with

**Proposed Apache Spark GSO Algorithm (Spark-GSO)**

Initially, a swarm of glowworms of a specific size is created. In the swarm, each glowworm is associated with a random positional vector (Xi) in the given search space is generated using uniform randomization. For each Xi vector, objective function J is calculated. Using Equation (1), luciferin level (Li) is evaluated for each glow worm with provided default luciferin value *L0*, *J(Xi)* and other constants. Initial local decision range r0 is used as local decision range *rd* for the first iteration. Once the entire swarm is initialized with updated information, the glowworms are added to a list. This list is used, broadcasted and updated in each iteration of the algorithm.

In the next phase of Spark-GSO, the iterative process of RDD operations are performed. Each iteration (RDD action) updates the glowworm swarm and the updated swarm is used as the input for the next iteration for processing.

Before the transformations are applied the entire swarm is sent to each task using a broadcast variable, a feature provided by spark to send and cache an object on each node before starting the tasks. The broadcast variable is initialized and broadcasted a s a list of glowworms for the processing in mappers. GSO constants such as s, rs, nt which are used in process of movement of glowworm swarms are retrieved from a java bean.

There are two mapper transformations used in the architecture. The first transformation is used to find the best neighbor from all the glowworms swarm ~~while the second transformation is used to update the luciferin level in the glowworm~~. To find the neighbor, an O(n2) algorithm is used. The algorithm involves in calculating the Euclidian distance and luciferin level comparisons between the given glowworm and all the other glowworms in space to locate a neighbor group as mentioned in the equation 2. Once the neighbor group is found, Equation 3 is used to find the best neighbor in that group. A technique called roulette wheel selection method is used to find the best neighbor. At the end of the first transformation, the best neighbor is attached to the original glowworm. Finally, the glowworm with an attached neighbor glowworm is emitted (returned) for further processing in the second transformation. The first transformation algorithm is outlined in the Algorithm 1 below.

The second transformation picks up the glowworms swarm with each glowworm attached with a best neighbor glowworm. This transformation mapper is used to update the luciferin level Li for each glowworm by evaluating the objective function for the new glowworm position. In this phase, the glowworm and its best neighbor position (Xj) is extracted at the start. Using Equation 4, the next step is to update the glowworm positional vector. Then the objective function is evaluated for the new positional vector for luciferin level calculation using Equation 1. The last step before emiting the new glowworm rdi is calculated using equation 5. Finally the glowworm with the updated information is emitted. The second transformation algorithms is outlined in the Algorithm 2 below.

Then an Apache Spark action Collect is implemented in the driver class. As spark transformations are lazy, no transformation is applied until the action is implemented. The collect supplies the actual updated glowworm swarm to the driver program. At the end of each iteration, the updated glowworm swarm is collected and broadcasted for the next iteration processing. Also, this updated glowworm is used for RDD operations in the next iteration.

|  |  |  |
| --- | --- | --- |
| Algorithm 1: Transormation Mapper1 |  | Algorithm 2: Transformation Mapper2 |
| **function** Call(Glowworm)  read (BroadcastSwarm)  //Copy the broadcasted swarm into a local variable  For each glowworm in BroadcastSwarm do  Xj=extractPosition(glowworm)  Lj=extractluciferin(glowworm)  EDist=returnEDistance(Xi,Xj )  if (EDist <rdi and Lj >Li) then  NeighborsGroup:add(j)  end if  end for  if (NeighborsGroup:size() <0) then  for each glowworm j in NeighborsGroup do  //calculate the probabilities from  the NeighborsGroup using Equation 2  prob[j]=calculateProbability(i,j)  end for  end if  nj=selectBestNeighbor(prob) //using the roulette wheel  Glowworm.setNeighborSize(NeighborsGroup.size())  Glowworm.addNeighbor(nj)  Emit (Glowworm)  end function |  | **function** Call(Glowworm)  if(neighborSize != 0)  extractNeighbor(Glowworm)  //Extract the neighbor glowworm information from the attached glowworm  else  //Extract the information from the current glowworm  glowwormi=NULL  extractInfo(Xi,Ji,Li,rdi)  fill(glowwormi,Xi,Jxi,Li,rdi)  end if  //calculate the new position for glowworm i  using Equation 4  newX=calculateNewX(Xi,Xj )  //update luciferin level for glowworm i  using objective function formula J  newJx=calculateNewJx(newX)  //update luciferin level for glowworm i  using Equation 1  newL=calculateNewX(Li,newJx)  //calculate the new rd for glowworm i  using Equation 5  newrd=calculateNewrd(rdi,nbSize)  glowwormi.update(newX,newJx,newL,newrd)  Emit(glowwormi)  end function |

Experiment and Results

In this section, we provide the details about the computing environment and the benchmark functions used for the experiments. We also discuss about the optimization quality, running time of the measurements for MR-GSO and Spark-GSO algorithms.

Environment:

We executed the MR-GSO and Spark-GSO algorithms on Wrangler Hadoop cluster hosted by Texas Advanced Computing Center (TACC). Each node Wrangler cluster has 24 cores (Intel(R) Xeon(R) CPU E5-2680 v3 @ 2.50GHz), 128 GB of memory. The Hadoop environment which we have used is Hadoop 2.6.0-cdh5.7.1 to implement MR-GSO algorithm while Apache Spark version 2.1.0 to execute Spark-GSO algorithm.

Benchmark Functions:

We have used two multimodal benchmark functions to evaluate the MR-GSO and Spark-GSO algorithms.