ID2222 – Data Mining

Assignment 5 -K-way Graph Partitioning Using JaBeJa

Course Coordinator:
Vladimar Vlassov & Sarunas Girdzijauskas

Teaching assistants:
Edward Tjörnhammer
Kambiz Ghoorchian
Mohamed Gabr

Group 2

Óttar Guðmundsson

Örn Arnar Karlsson

2018-12-07

Description

We were tasked with implementing and understanding the algorithm JaBeJa (means *swap*), which is a distributed graph partitioning algorithm that uses a gossip-based peep-to-peer technique.

First, we had to complete two missing parts of the algorithm that was downloaded from an open repository on Github.

Secondly, we had to tweak the configurations of the JaBeJa algorithm, run it and analyze it to find the smallest edge cuts for a selection of graphs.

Finally, for the bonus points, we had to define our own acceptance probability function or make changes to the Ja-Be-Ja algorithm to improve its performance. Then an evaluation was needed on how our changes affected the performance of graph partitioning.

How to run

Taken from the assignment.
You can run the program using the run.sh script. Run ./run.sh -help to see all the possible command line parameters. All the sample graphs are stored in the ./graphs directory; use the 3elt, add20, and Facebook/Twitter graphs in your experiments. After running the experiment, the results are stored in the ./output directory. Use the plot.sh to visualize the results. plot.sh generates a graph.png file in the current directory.

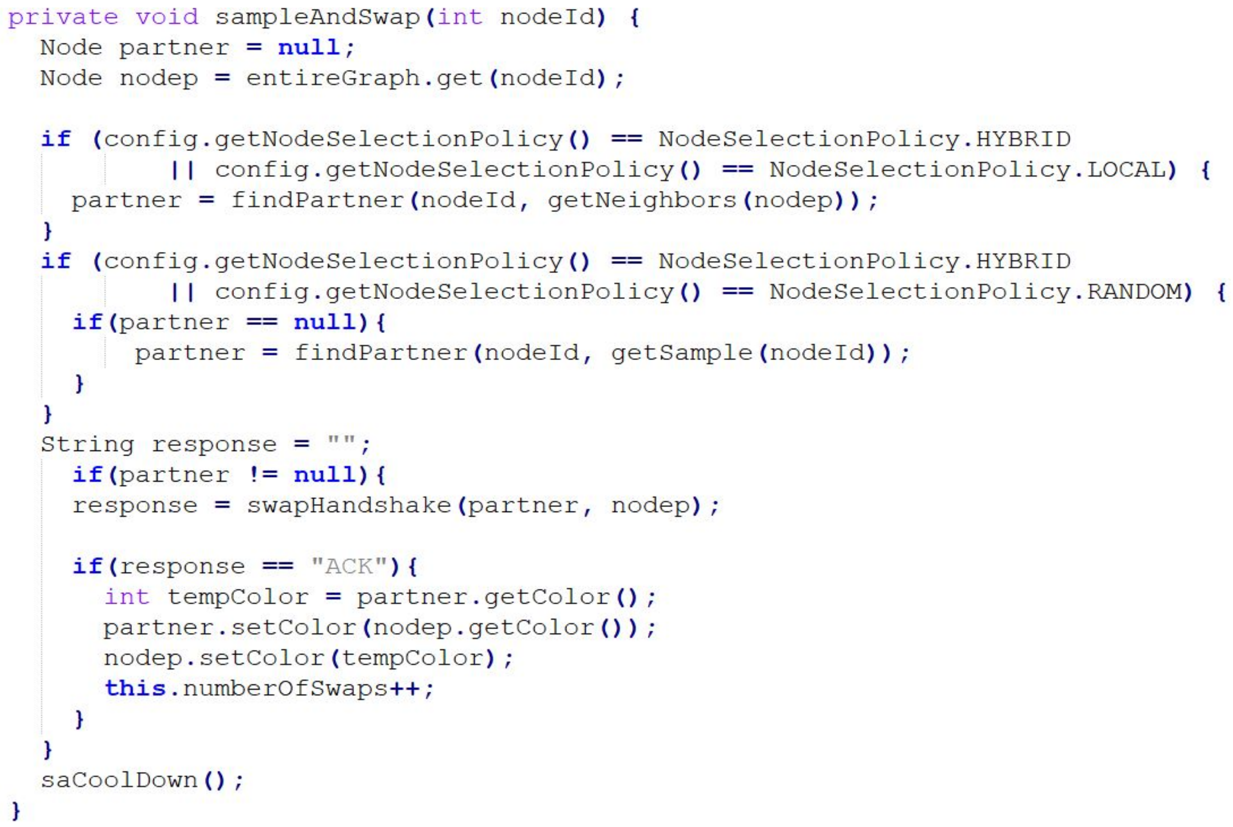
```
>> ./compile.sh
>> ./run -graph ./graphs/3elt.graph
>>./plot .sh output/result
```

Note that the **bold** part the file location for the graph.

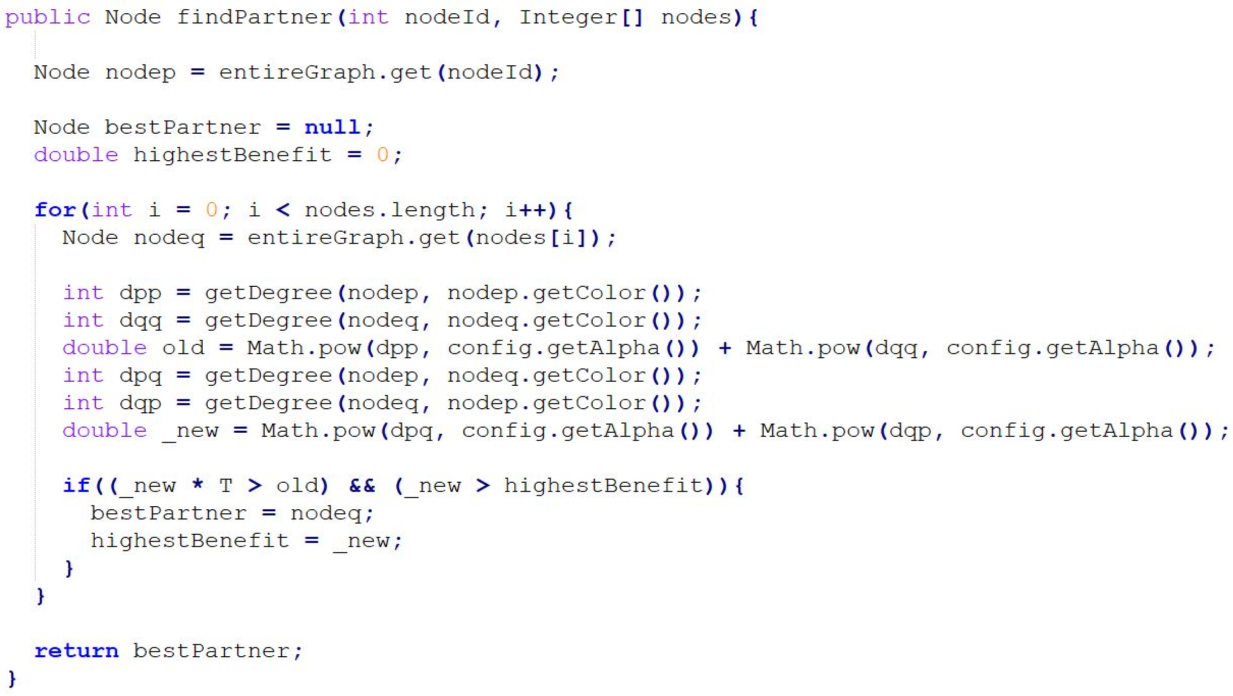
Solution

Task 1

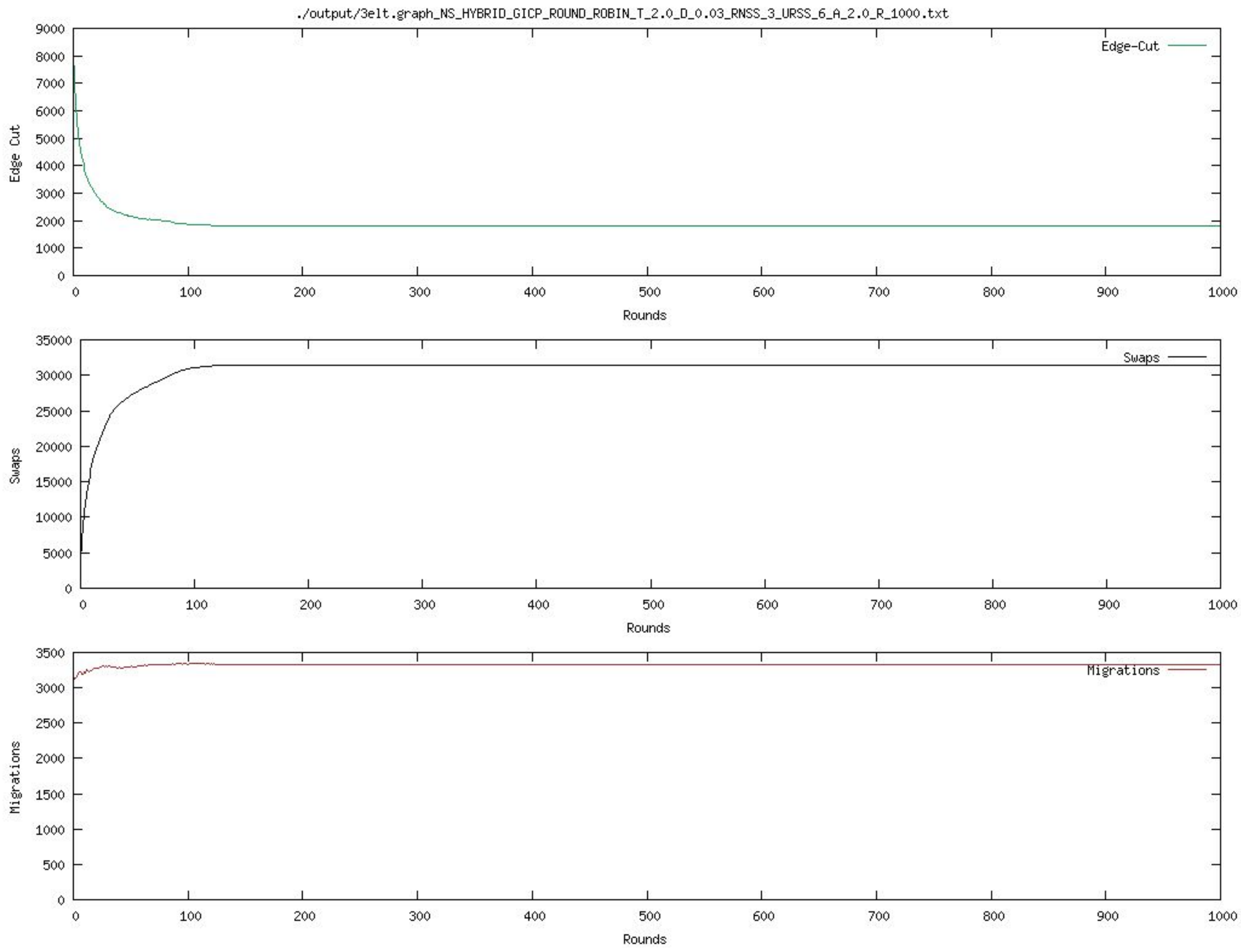
For this task, only the JaBeJa.java class was edited. The two methods that needed implementation were marked with a TODO tag, namely the *sampleAndSwap()* function, and the *findPartner()* method. This was pretty straightforward but did take us quite the time to understand since the commands given to run were not the same for the graph that was displayed. After we noticed that the graph had a different name than the command, we immediately knew that we had the correct solution and proceeded onwards.



```
private void sampleAndSwap(int nodeId) {
  Node partner = null;
  Node nodep = entireGraph.get(nodeId);

  if (config.getNodeSelectionPolicy() == NodeSelectionPolicy.HYBRID
          || config.getNodeSelectionPolicy() == NodeSelectionPolicy.LOCAL) {
    partner = findPartner(nodeId, getNeighbors(nodep));
  }
  if (config.getNodeSelectionPolicy() == NodeSelectionPolicy.HYBRID
          || config.getNodeSelectionPolicy() == NodeSelectionPolicy.RANDOM) {
    if(partner == null){
        partner = findPartner(nodeId, getSample(nodeId));
    }
  }
  String response = "";
    if(partner != null){
    response = swapHandshake(partner, nodep);

    if(response == "ACK"){
      int tempColor = partner.getColor();
      partner.setColor(nodep.getColor());
      nodep.setColor(tempColor);
      this.numberOfSwaps++;
    }
  }
  saCoolDown();
}
```

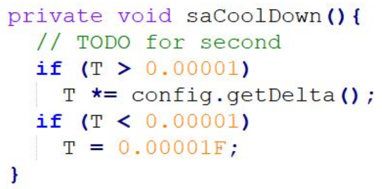


```
public Node findPartner(int nodeId, Integer[] nodes){

  Node nodep = entireGraph.get(nodeId);

  Node bestPartner = null;
  double highestBenefit = 0;

  for(int i = 0; i < nodes.length; i++){
    Node nodeq = entireGraph.get(nodes[i]);

    int dpp = getDegree(nodep, nodep.getColor());
    int dqq = getDegree(nodeq, nodeq.getColor());
    double old = Math.pow(dpp, config.getAlpha()) + Math.pow(dqq, config.getAlpha());
    int dpq = getDegree(nodep, nodeq.getColor());
    int dqp = getDegree(nodeq, nodep.getColor());
    double _new = Math.pow(dpq, config.getAlpha()) + Math.pow(dqp, config.getAlpha());

    if((_new * T > old) && (_new > highestBenefit)){
      bestPartner = nodeq;
      highestBenefit = _new;
    }
  }

  return bestPartner;
}
```

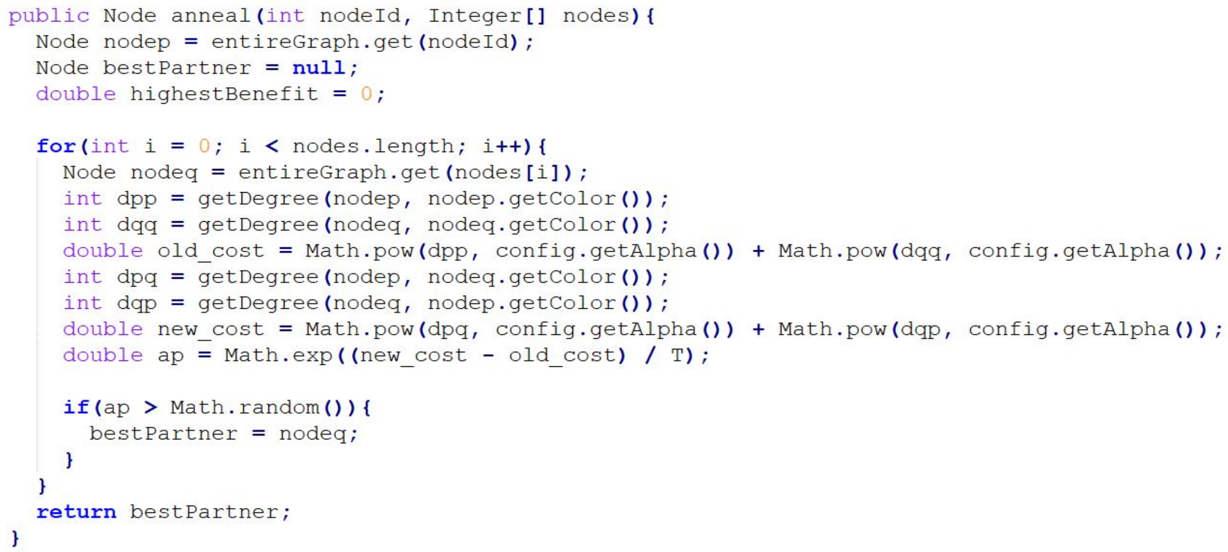


Task 2

1. For this task, we had to make some changes to the algorithm. The acceptance probability was linear and so was the temperature cooldown. Thus, changes were made to the saCoolDown() function and the acceptance probability function as well, which was also made exponential.

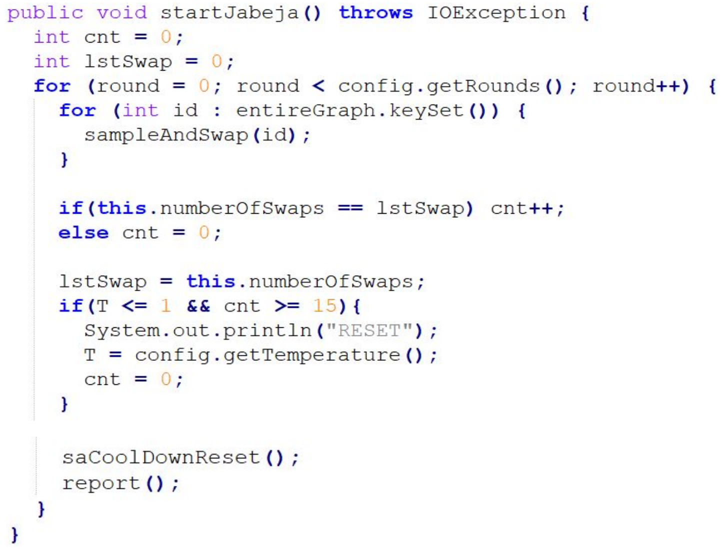


```
private void saCoolDown(){
  // TODO for second
  if (T > 0.00001)
    T *= config.getDelta();
  if (T < 0.00001)
    T = 0.00001F;
}
```

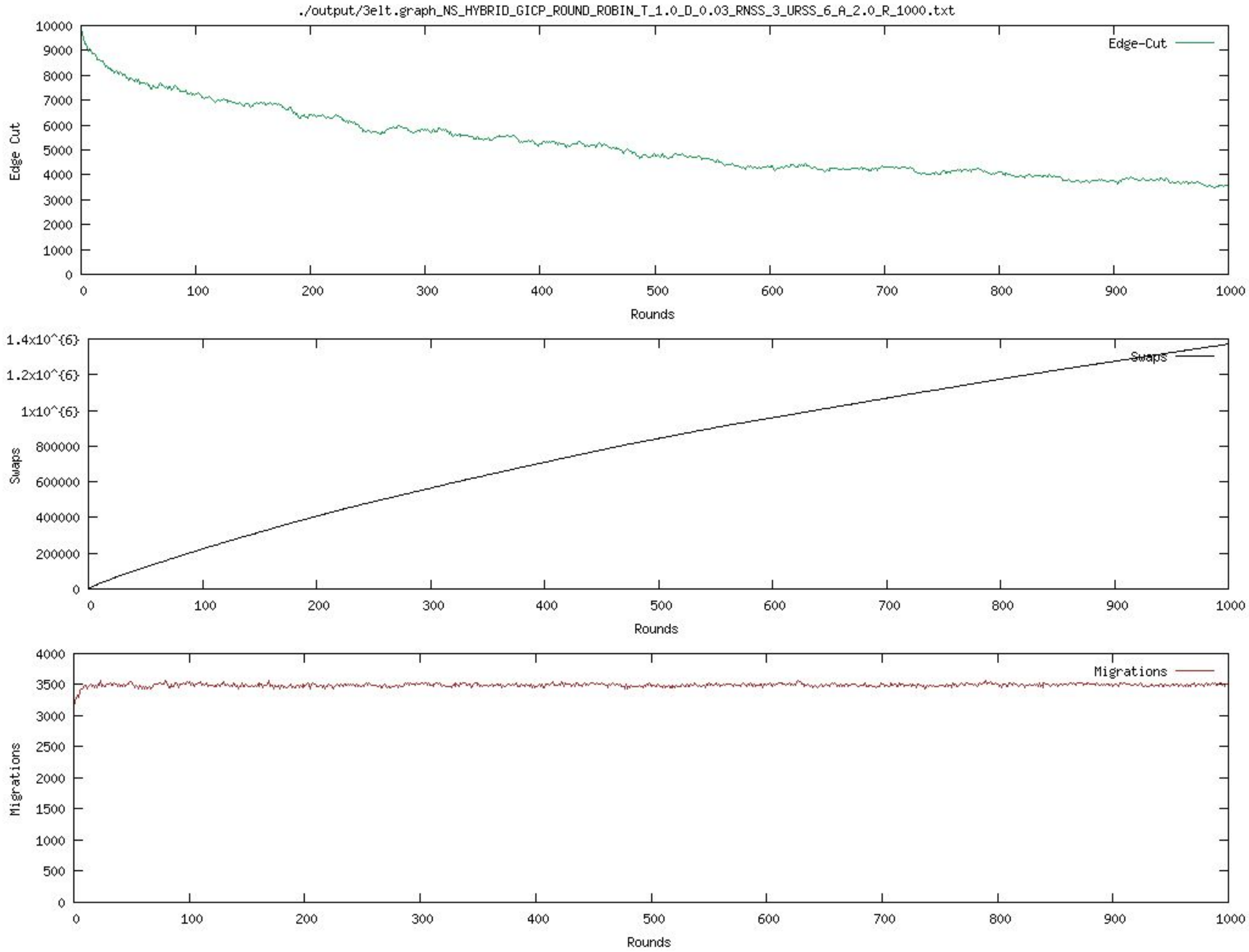


```java
public Node anneal(int nodeId, Integer[] nodes){
  Node nodep = entireGraph.get(nodeId);
  Node bestPartner = null;
  double highestBenefit = 0;

  for(int i = 0; i < nodes.length; i++){
    Node nodeq = entireGraph.get(nodes[i]);
    int dpp = getDegree(nodep, nodep.getColor());
    int dqq = getDegree(nodeq, nodeq.getColor());
    double old_cost = Math.pow(dpp, config.getAlpha()) + Math.pow(dqq, config.getAlpha());
    int dpq = getDegree(nodep, nodeq.getColor());
    int dqp = getDegree(nodeq, nodep.getColor());
    double new_cost = Math.pow(dpq, config.getAlpha()) + Math.pow(dqp, config.getAlpha());
    double ap = Math.exp((new_cost - old_cost) / T);

    if(ap > Math.random()){
      bestPartner = nodeq;
    }
  }
  return bestPartner;
}
```

2. For the second task, we needed to implement a reset function for the temperature. Our implementation was done in such a way that if the number of swaps had stayed the same for 15 rounds or more, we bumped the temperature back up to 2.



```java
public void startJabeja() throws IOException {
  int cnt = 0;
  int lstSwap = 0;
  for (round = 0; round < config.getRounds(); round++) {
    for (int id : entireGraph.keySet()) {
      sampleAndSwap(id);
    }

    if(this.numberOfSwaps == lstSwap) cnt++;
    else cnt = 0;

    lstSwap = this.numberOfSwaps;
    if(T <= 1 && cnt >= 15){
      System.out.println("RESET");
      T = config.getTemperature();
      cnt = 0;
    }

    saCoolDownReset();
    report();
  }
}
```

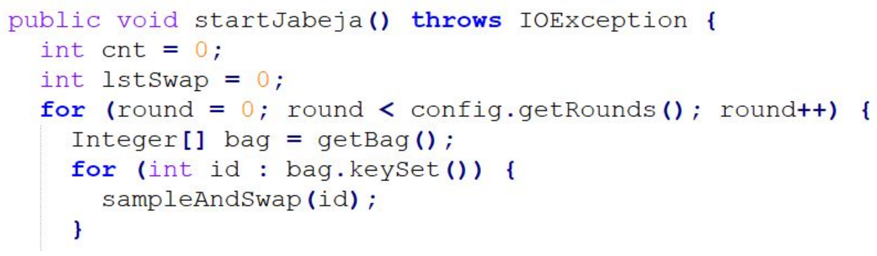


Bonus

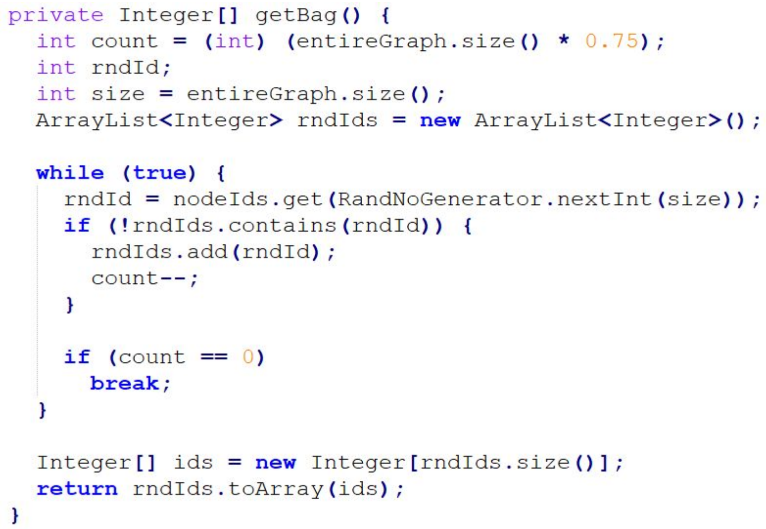
For the bonus task, we had to define our own acceptance probability function or make some sort of a change to the JaBeJa algorithm in order to improve its performance.

We decided to reduce the number of nodes that had to be processed in every round by less than 50%. The motivation for our improvement was that speeding up the algorithm by more than 50% could definitely be considered as an improvement. However, the intuiton behind more than 50% was based on quorums. By looping over the majority of nodes, we know that we can split up the nodes to a minimum of two partitions.

We implemented the method this way



```
public void startJabeja() throws IOException {
  int cnt = 0;
  int lstSwap = 0;
  for (round = 0; round < config.getRounds(); round++) {
    Integer[] bag = getBag();
    for (int id : bag.keySet()) {
      sampleAndSwap(id);
    }
```



```
private Integer[] getBag() {
  int count = (int) (entireGraph.size() * 0.75);
  int rndId;
  int size = entireGraph.size();
  ArrayList<Integer> rndIds = new ArrayList<Integer>();

  while (true) {
    rndId = nodeIds.get(RandNoGenerator.nextInt(size));
    if (!rndIds.contains(rndId)) {
      rndIds.add(rndId);
      count--;
    }

    if (count == 0)
      break;
  }

  Integer[] ids = new Integer[rndIds.size()];
  return rndIds.toArray(ids);
}
```

Without bagging

Time: 39:58

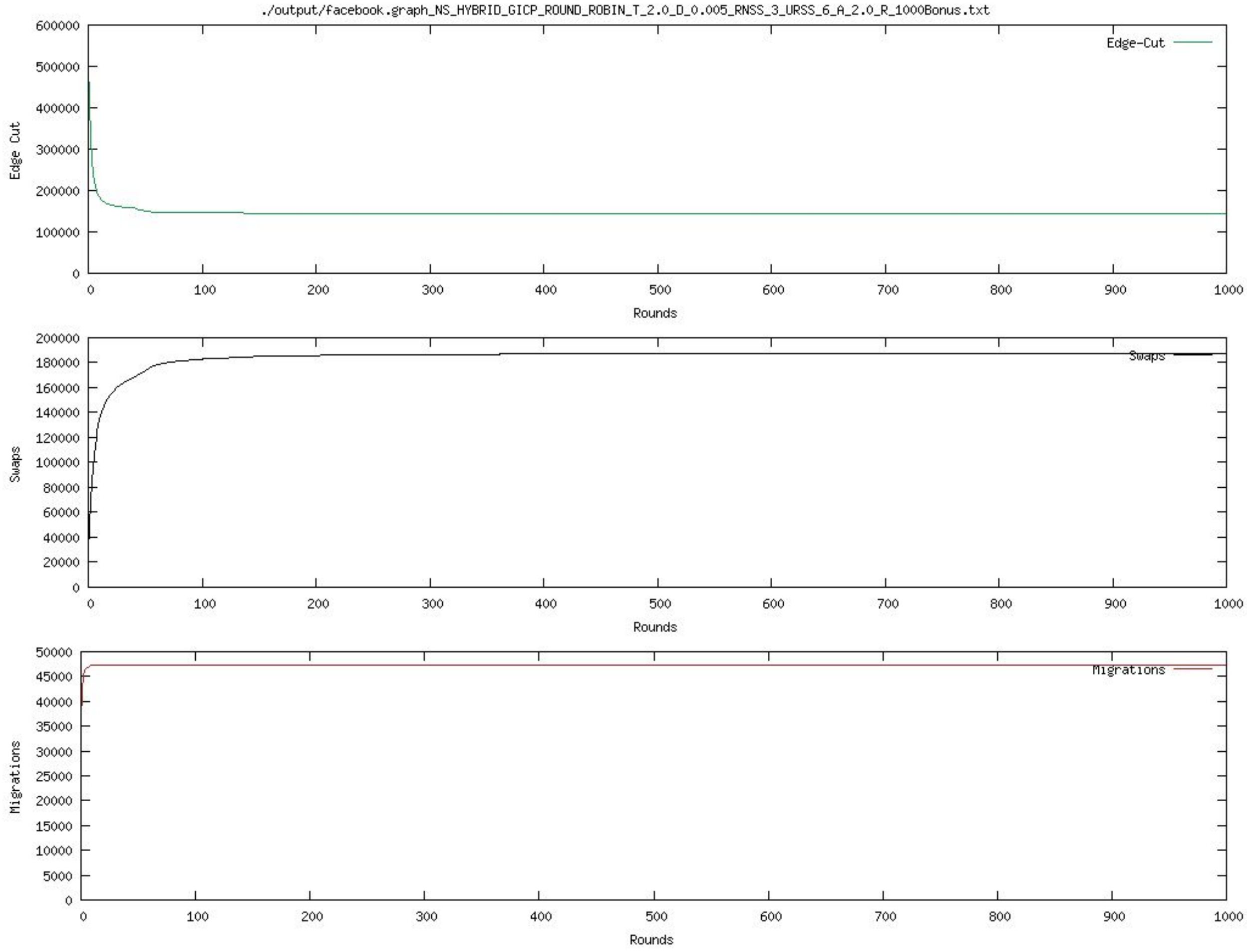
With bagging

Edge 145.105

Swaps 187.252

Migrations 47.385

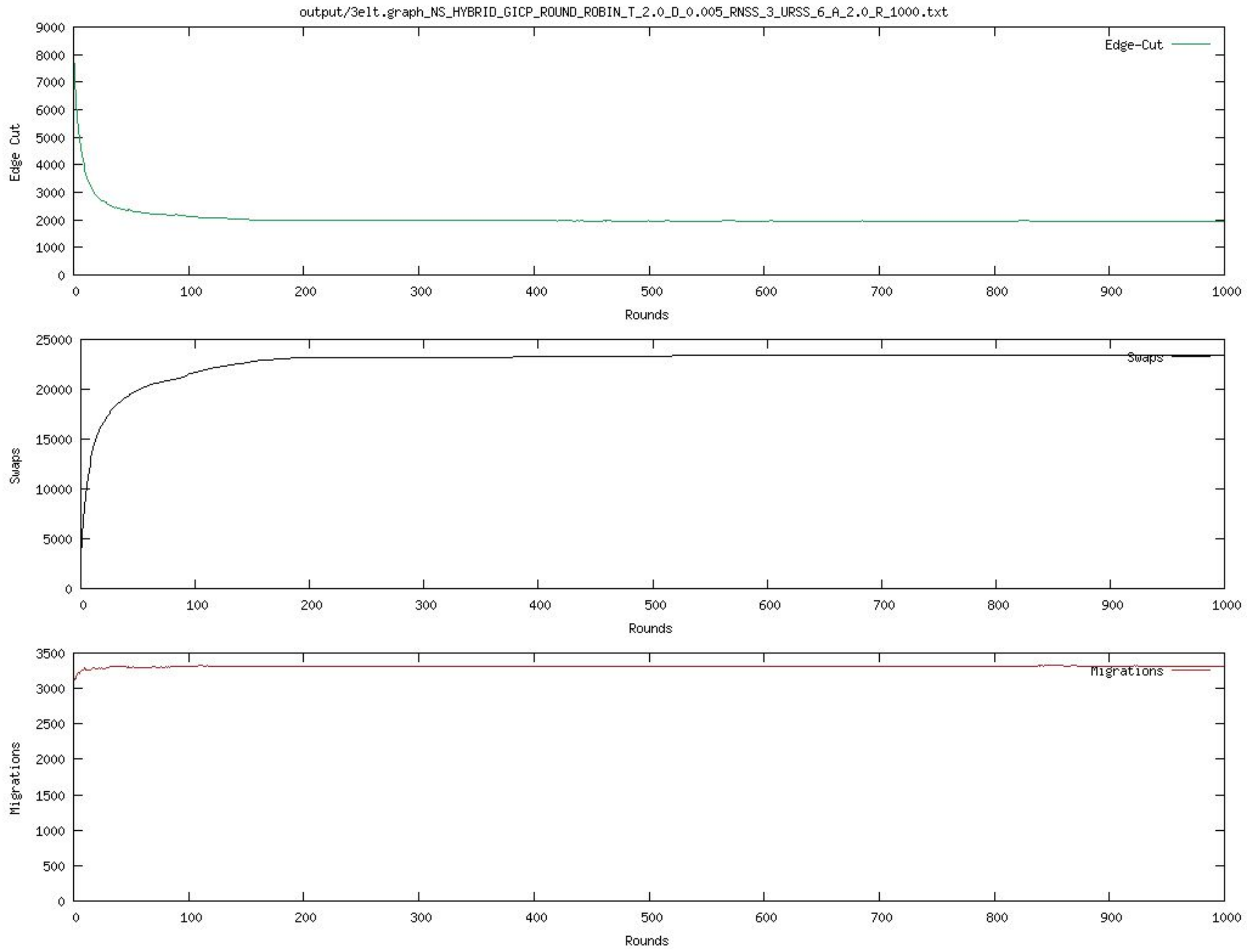
Time: *35:13*



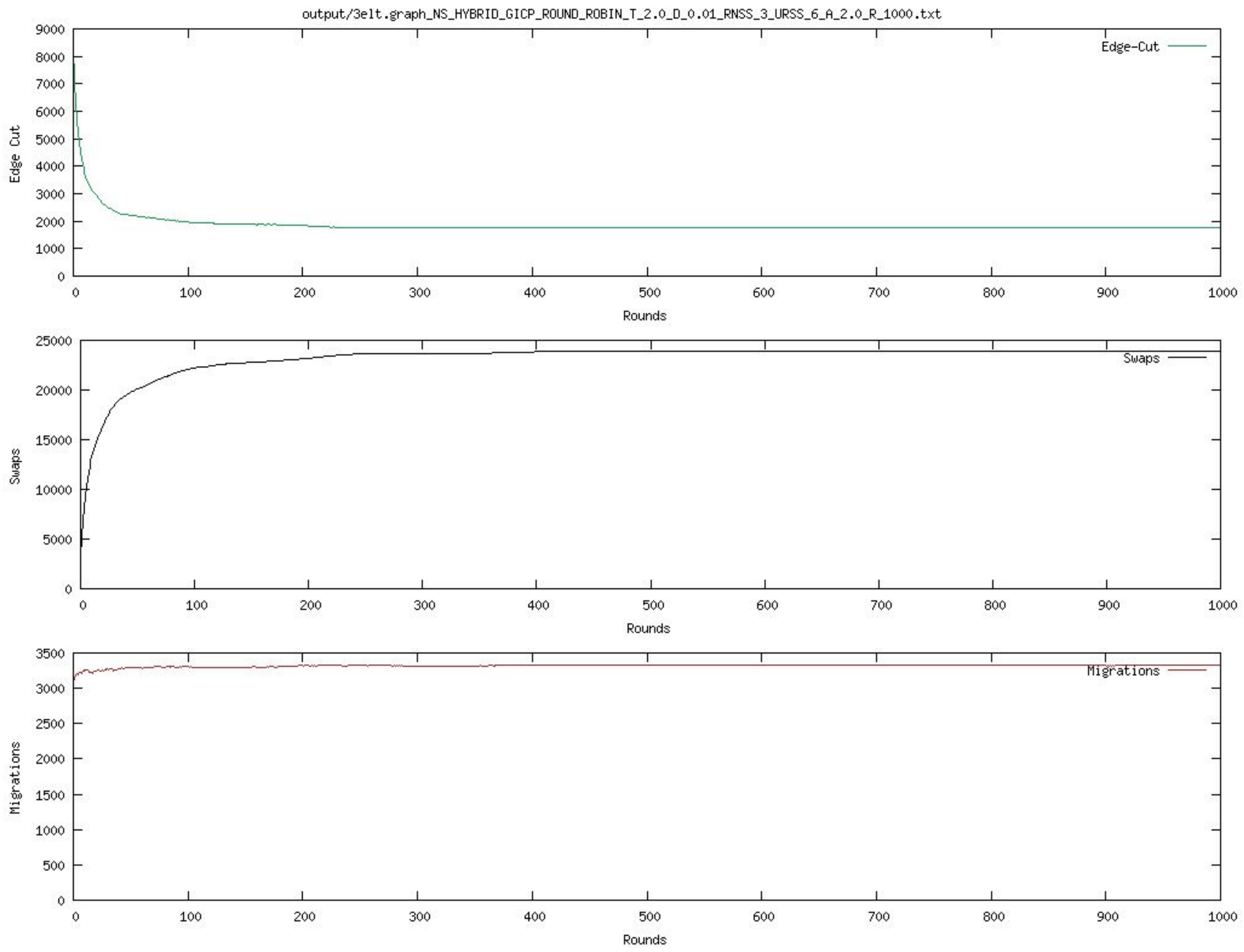
Results

In your report discuss how your changes affect the performance of the algorithm in terms

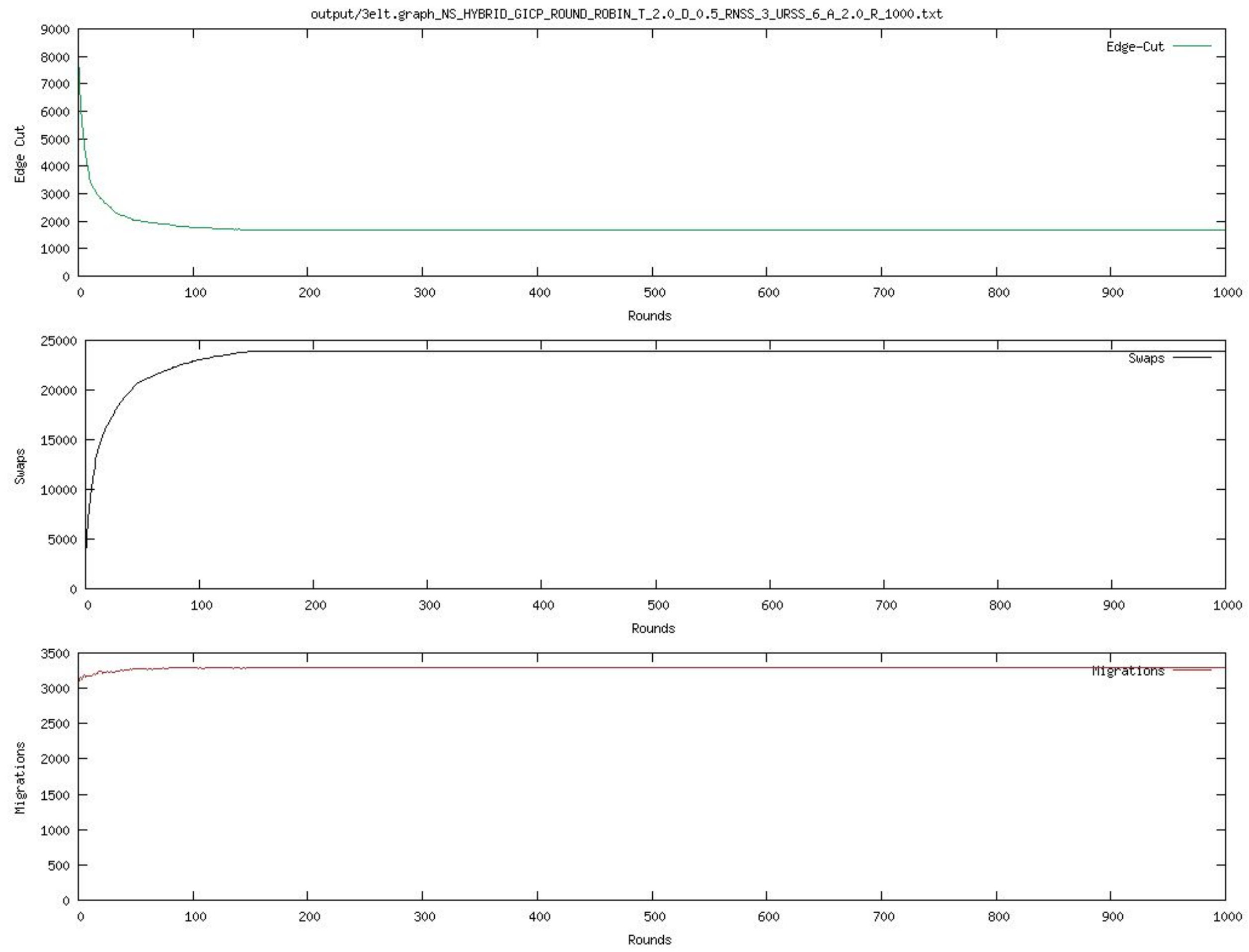
***3elt** Delta=0.005*
edge 1950
swaps 23453
migrations 3319



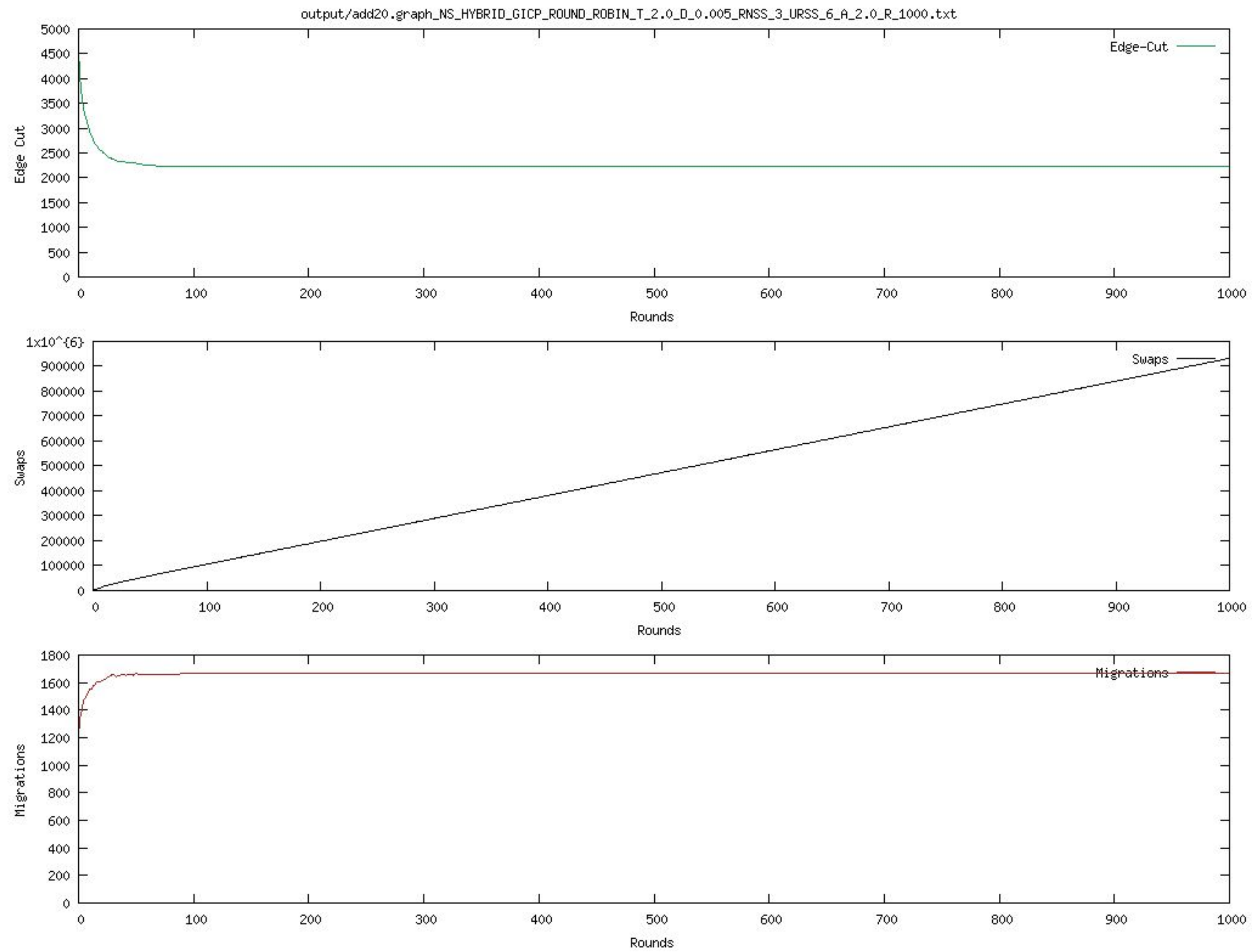
***3elt** Delta=0.01*
edge 1755
swaps 23879
migrations 3320



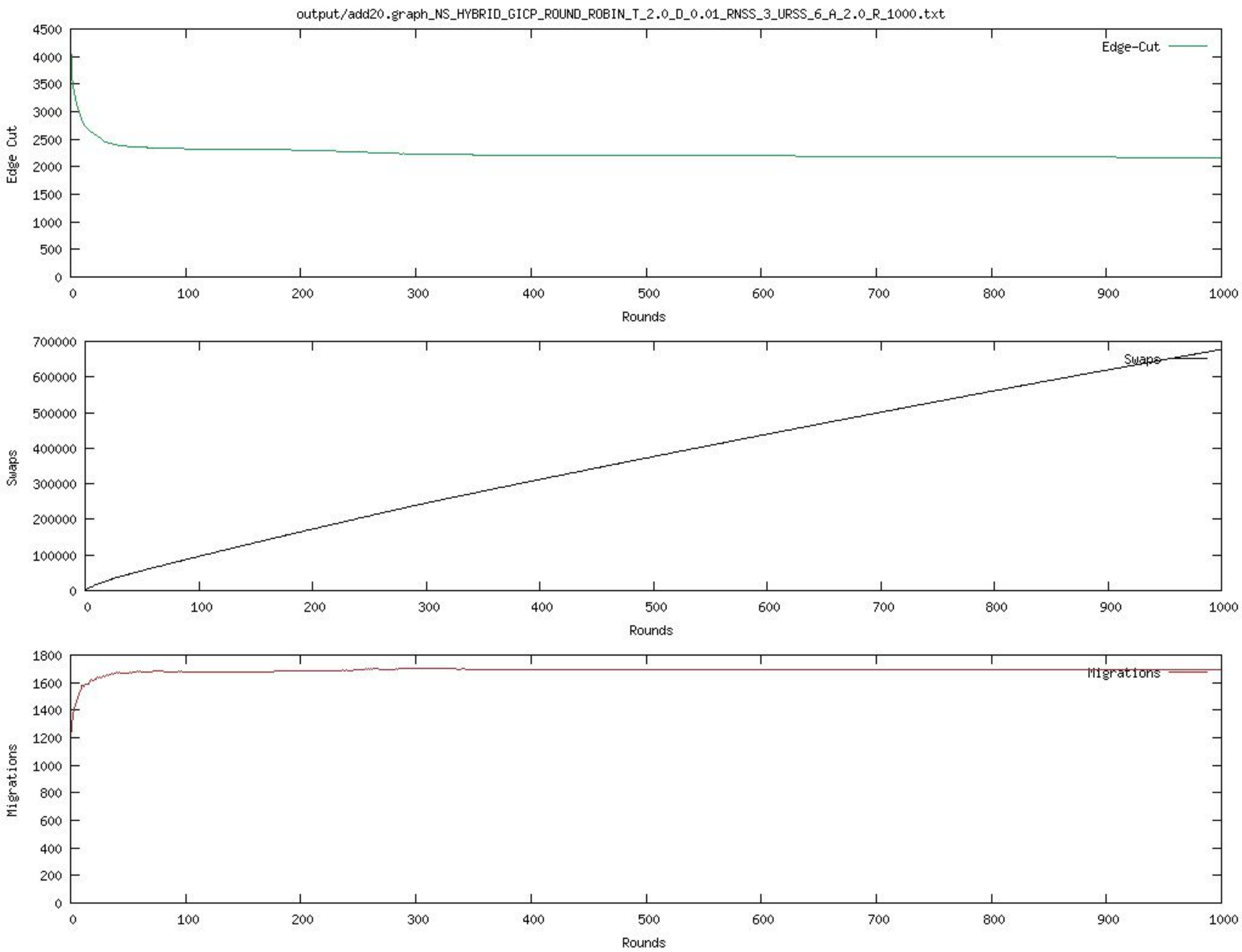
3elt *Delta=0.5*
edge 1673
swaps 23917
migrations 3387



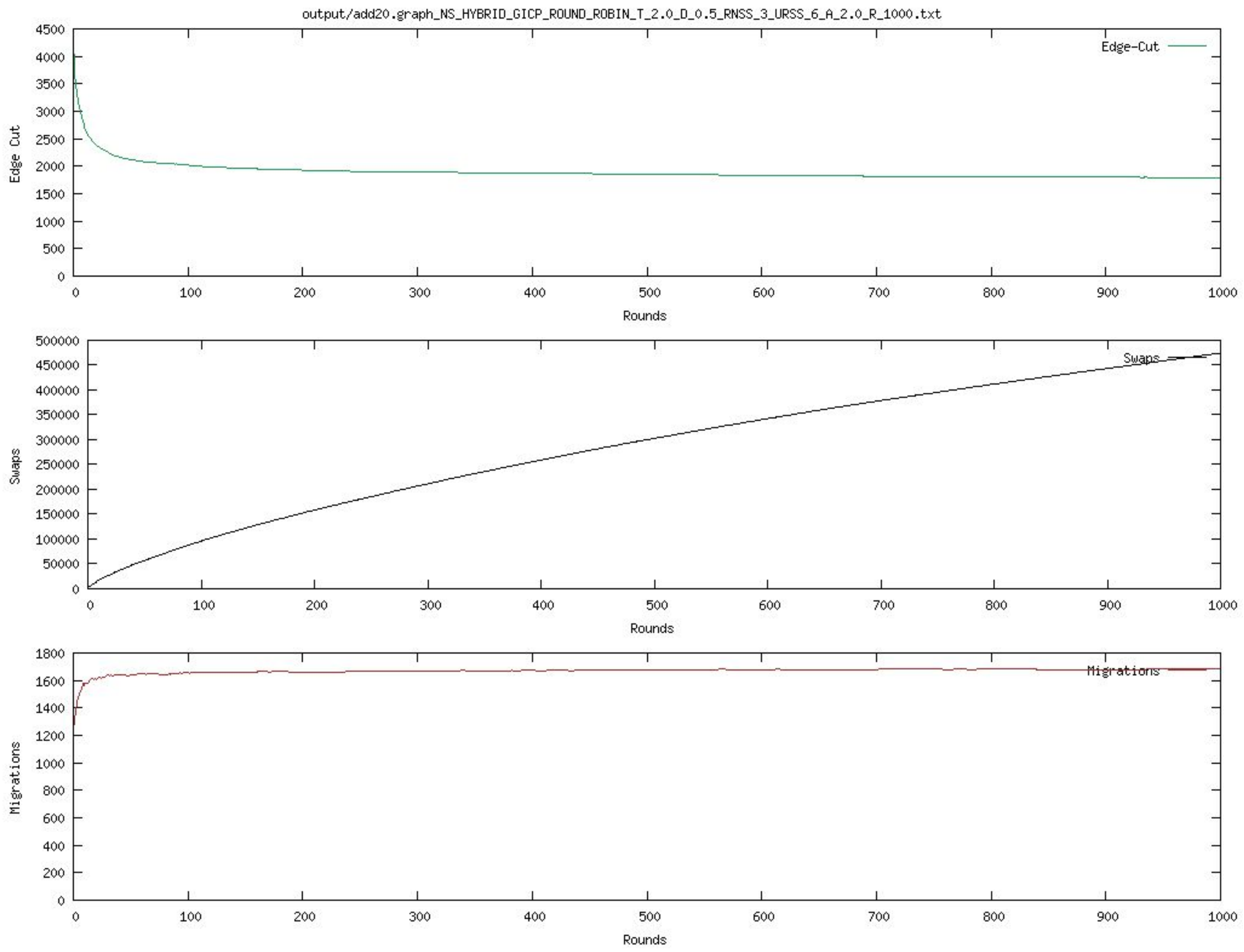
add20 *Delta=0.005*
edge 2240
swaps 931300
migrations 1664



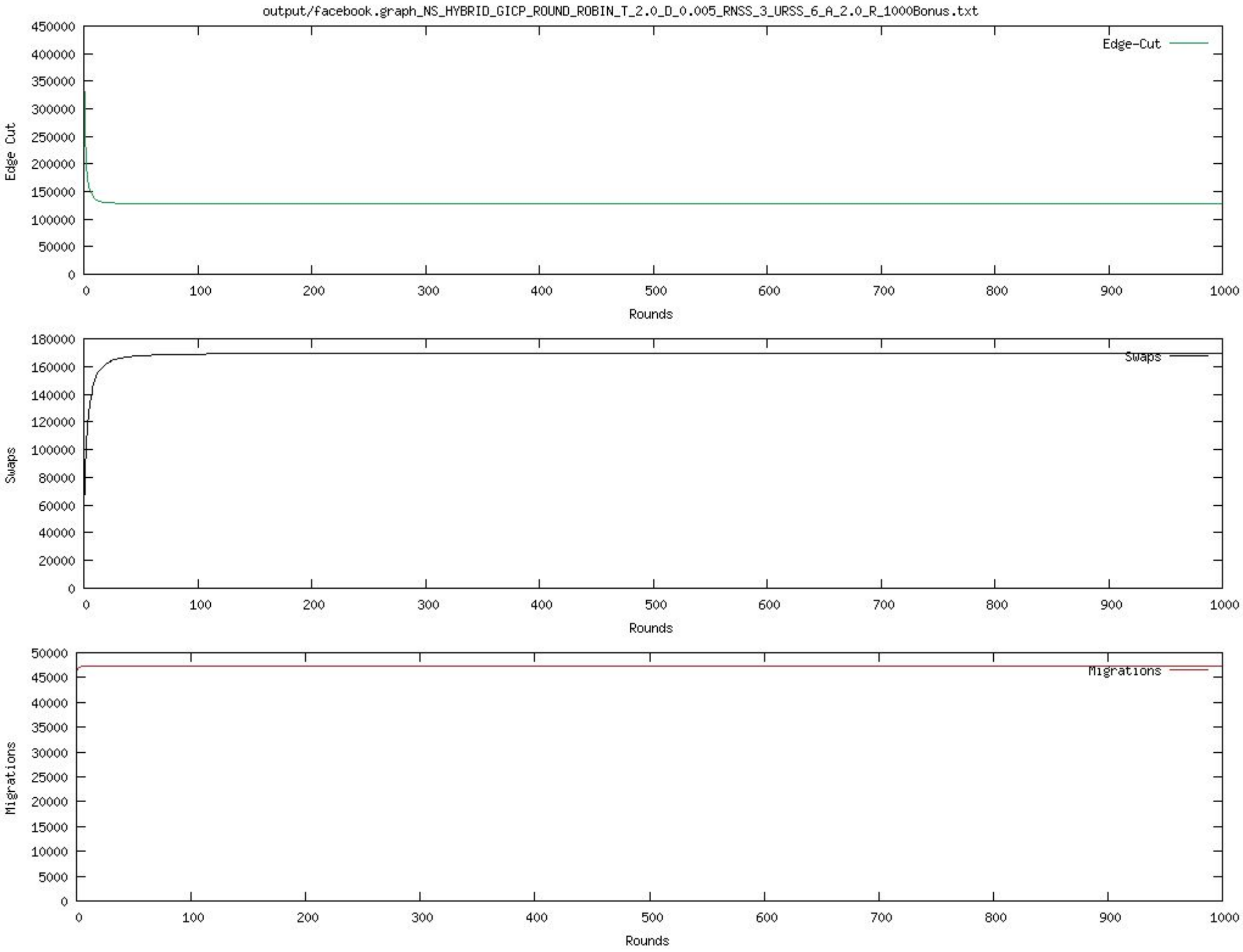
add20 *Delta=0.01*
edge 2171
swaps 676982
migrations 1696



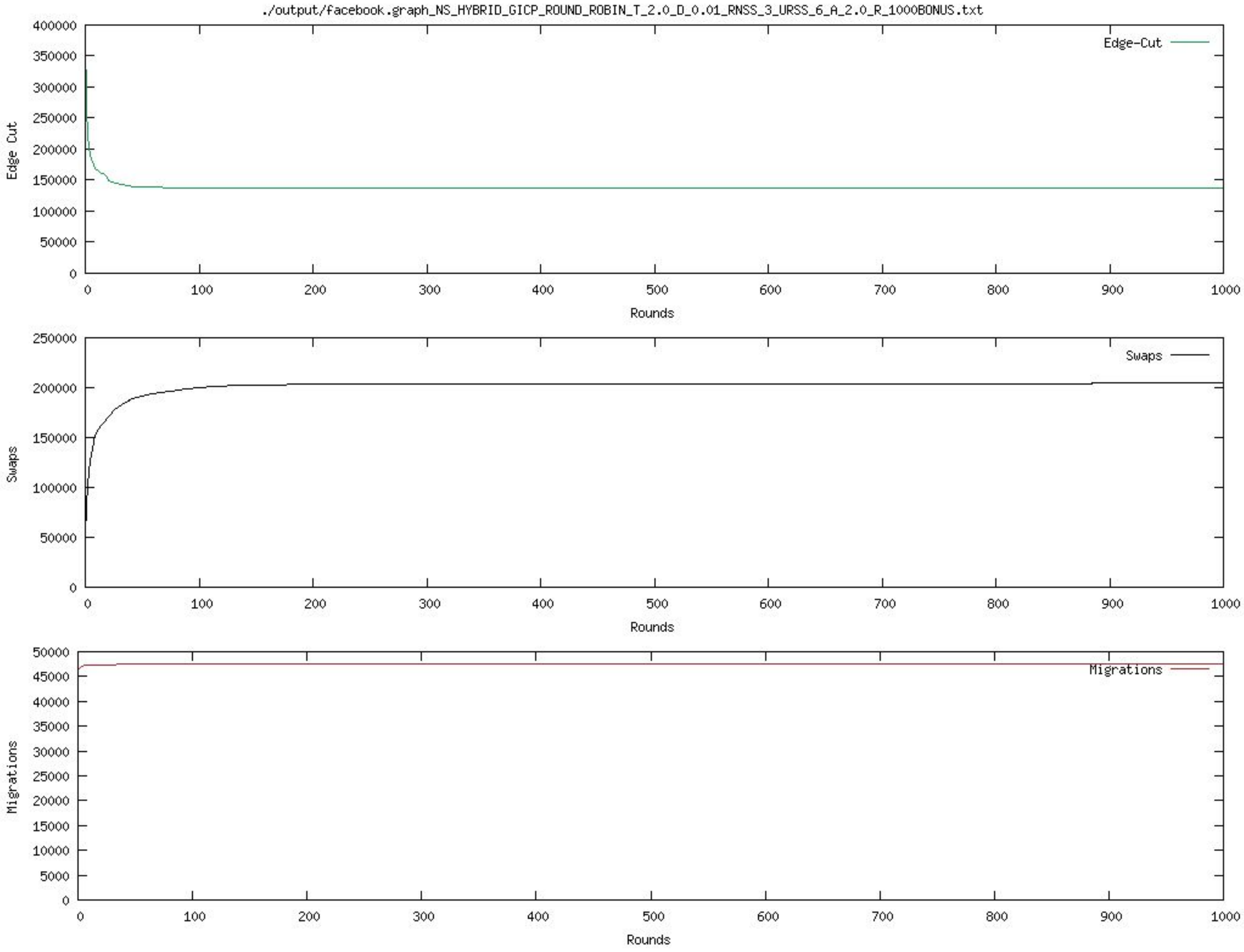
add20 *Delta=0.5*
edge 1781
swaps 474256
migrations 1683



***facebook** Delta=0.005*
edge 128.126
swaps 169.578
migrations 47.280



***facebook** Delta=0.01*
edge 136.325
swaps 204.288
migrations 47.479


./output/facebook.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_2.0_D_0.01_RNSS_3_URSS_6_A_2.0_R_1000BONUS.txt
Edge Cut
400000
350000
300000
250000
200000
150000
100000
50000
0
Edge-Cut
0
100
200
300
400
500
600
700
800
900
1000
Rounds
Swaps
250000
200000
150000
100000
50000
0
Swaps
0
100
200
300
400
500
600
700
800
900
1000
Rounds
Migrations
50000
45000
40000
35000
30000
25000
20000
15000
10000
5000
0
Migrations
0
100
200
300
400
500
600
700
800
900
1000
Rounds