DataMining ID2222 - Homework 5
K-way Graph Partitioning using JaBeJa

Sherly Sherly and Anna Martignano

December 10, 2019

1 Introduction

In this project, we will modify and test a Distributed Algorithm for Balanced Graph Partitioning: JA-BE-JA described in the paper written by Fatemeh Rahimian, Amir H. Payberah, Sarunas Girdzijauskas, Mark Jelasity and Seif Haridi. [1]. There are two tasks in this project which will be described in **Section 2** and **Section 3**.

2 Task 1

The first task performed for this assignment is the completion of the code provided at this link by implementing the 'sampleAndSwap()' and 'findPartner()' method of the Jabeja class as described in the pseudo-algorithm in the paper. The steps will be described in the next paragraph.

sampleAndSwap(). *Step 1.* We start by looking at the neighborhood of the given node p. If there exists in its neighborhood a node q which can decrease the energy of the system if swapped with node p, such a node q becomes the new partner of node p. *Step 2.* If no partner is found among node p's neighborhood, we evaluate the energy of a potential swap with a random node in the network, and eventually such random node can become a partner if the graph edge-cut, i.e. system energy decreases with the swap. *Step 3.* The next step to complete 'sampleAndSwap' is to select selects the best partner of a node.

The selection depends on the evaluation of the energy system of a potential swap and return the node q, if exists, which satisfy the following formula:

$$(d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha) \times T > d_p(\pi_p)^\alpha + d_q\pi_q)^\alpha$$

and maximizes:

$$(d_p(\pi_q)^\alpha + d_q(\pi_p)^\alpha)$$

The detailed algorithm is described below.

Algorithm 1 JA-BE-JA Algorithm

Require: Any node p in the graph has the following methods:
- getNeighbors(): returns p's neighbors
- getSample(): returns a uniform sample of all the nodes
- getDegree(c): returns the number of p's neighbors that have color c

1: **procedure** SAMPLE AND SWAP
2: $partner \leftarrow FindPartner(p.getNeighbors(), T_r)$
3: **if** $partner = null$ **then**
4: $partner \leftarrow FindPartner(p.getSample(), T_r)$
5: **end if**
6: **if** $partner \neq null$ **then**
7: color exchange handshake between p and partner
8: **end if**
9: $T_r \leftarrow T_r - \delta$
10: **if** $T_r < 1$ **then**
11: $T_r \leftarrow 1$
12: **end if**
13: **end procedure**
14: **function** FIND PARTNER(Node [] nodes, float T_r)
15: $highest \leftarrow 0$
16: $bestPartner \leftarrow null$
17: **for** $q \in nodes$ **do**
18: $d_{pp} \leftarrow p.getDegree(p.color)$
19: $d_{qq} \leftarrow q.getDegree(q.color)$
20: $old \leftarrow d_{pp}^\alpha + d_{qq}^\alpha$
21: $d_{pq} \leftarrow p.getDegree(q.color)$
22: $d_{qp} \leftarrow q.getDegree(p.color)$
23: $new \leftarrow d_{pq}^\alpha + d_{qp}^\alpha$
24: **if** $(new \times T_r > old) \wedge (new > highest)$ **then**
25: $bestPartner \leftarrow q$
26: $highest \leftarrow new$
27: **end if**
28: **end for**
29: **return** $bestPartner$
30: **end function**

3 Task 2

There are two parts to this task. The first part involves implementing a different simulated annealing mechanism described here. In the second part, we will investigate how the Ja-Be-Ja algorithm behaves when the simulated annealing is restarted after Ja-Be-Ja has converged.

First Part. The new proposed solution differs from the previous one in the definition of the function which control how the temperature parameter T change over iterations. We will be modifying the function 'saCoolDown()' given in the original setup. In the previous task 1, we have considered an initial temperature of 2 and we have decrease it iteration by iteration by $\delta = 0.003$. In this alternative solution, the initial temperature will be set at 1.0 and decreasing it at the end of each iteration by multiplying it by a constant called α. α is a hyper parameter which needs to be tuned and typical choices for it is between 0.8 and 0.99. Another way to improve simulated annealing to repeat the process of finding a neigbor and comparing its cost multiple times (usually between 100 to 1000) to find the best move at each temperature. In our case, we will not be implementing this part.

Second Part. We will be re-initializing the value of T to the initial value when Ja-Be-Ja has converged.

4 Dataset

The graph datasets used to assess the performance of the JaBeJa algorithm are the following:

- Facebook Graph
- Twitter Graph
- Add20
- 3elt

5 Results

In order to evaluate the results of the distributed algorithm three metrics are considered:

- Number of swaps

- Convergence time: the iteration at which the algorithm stops to perform swaps
- Minimum edge-cut: the energy of the system as the number of edges between nodes with different colors defined by the following formula $E(G, \pi) = \frac{1}{2} \sum_{p \in V} (d_p - d_p(\pi_p))$

The results for each tasks will be presented in the respective subsections below.

5.1 Task1

Dataset	T_{min}	δ	α	Swaps	Convergence Time	Min edge-cut
3elt	2	0.003	2	1580209	472	2604
add20	2	0.003	2	1090263	987	2095
facebook	2	0.003	2	21200364	850	134246
twitter	2	0.003	2	899515	813	41156

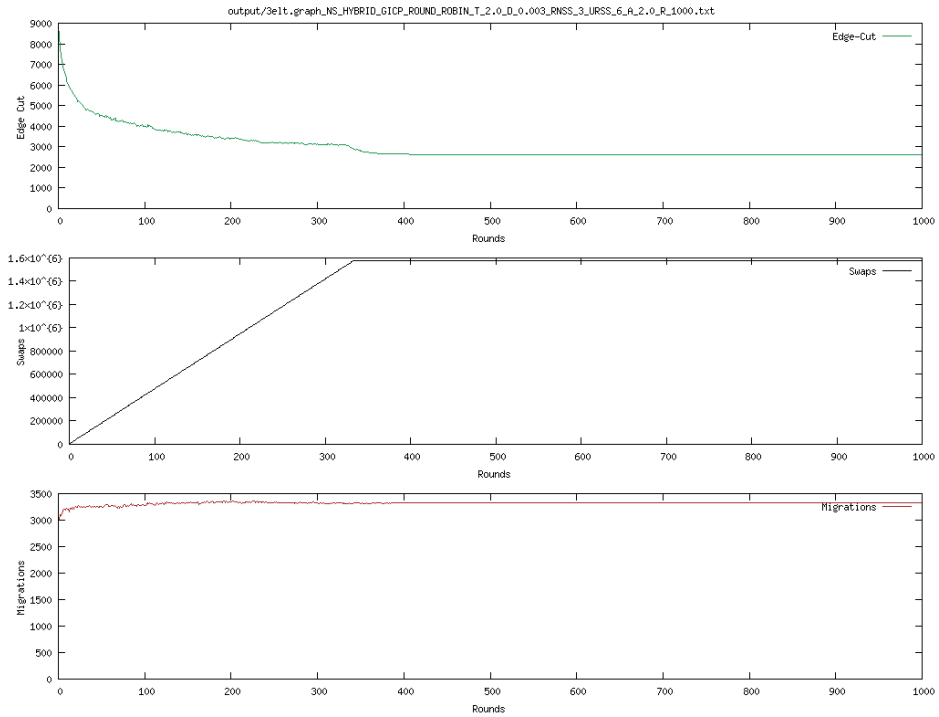

Figure 1: 3elt - Task 1

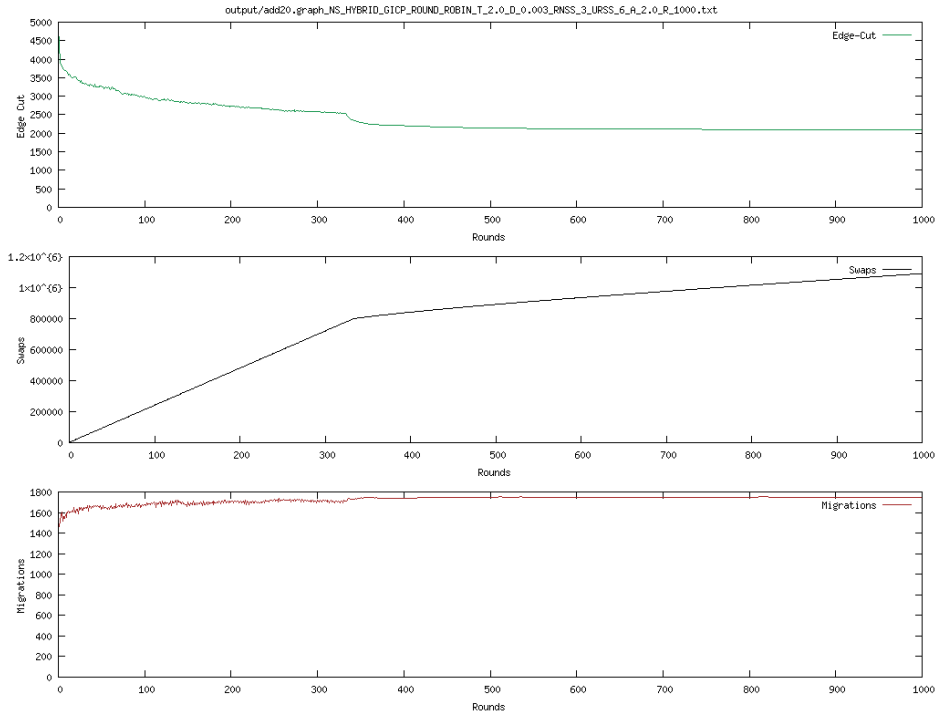

Figure 2: add20 - Task 1

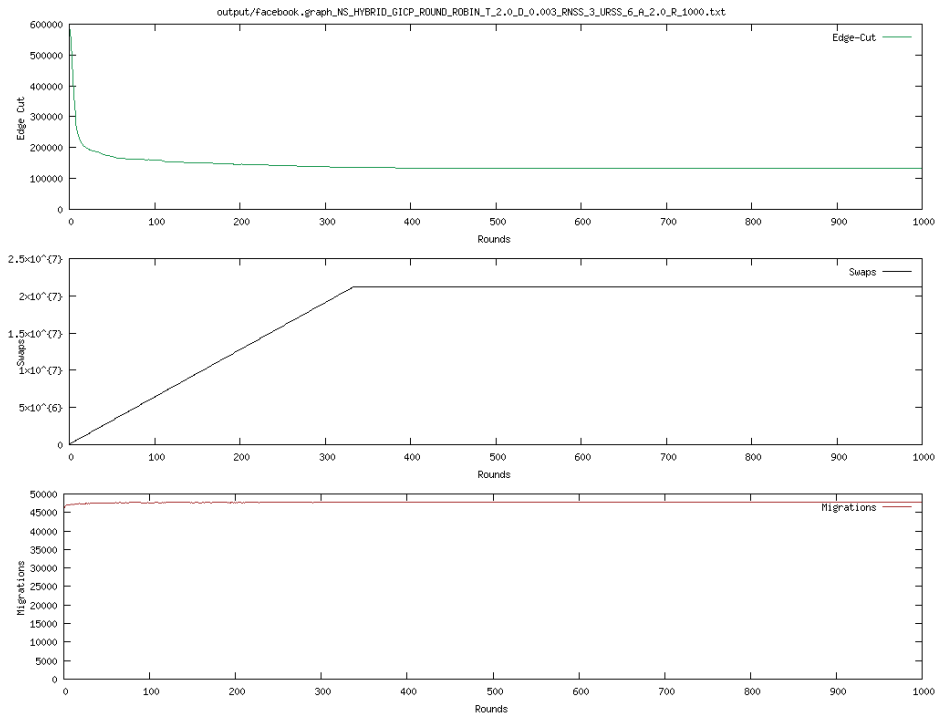

Figure 3: facebook - Task 1

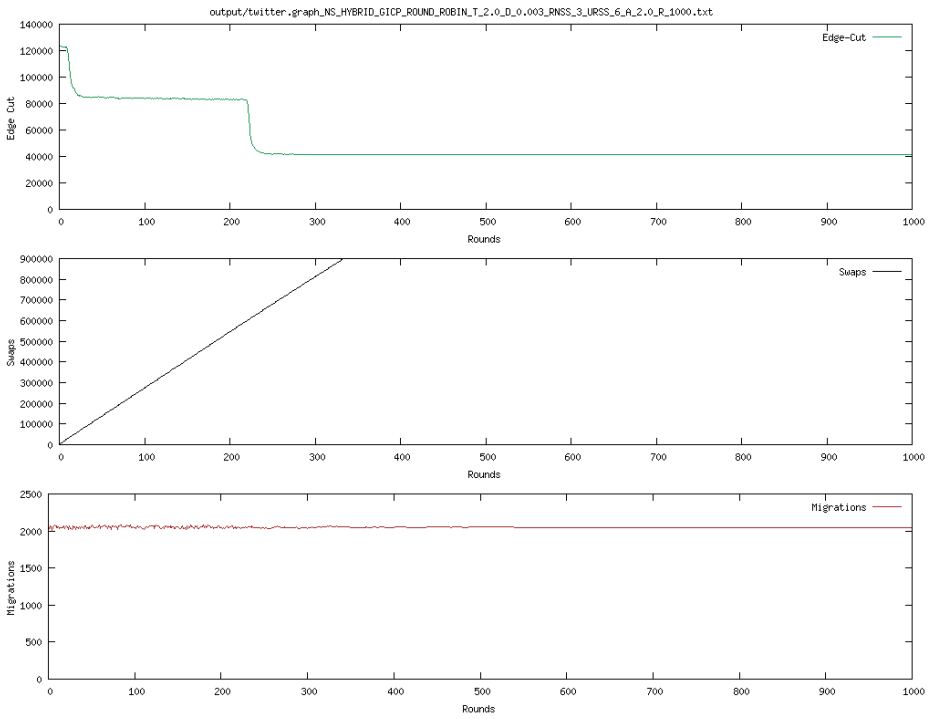

Figure 4: twitter - Task 1

5.2 Task 2

In this first table we have presented the results of the variated simulated annealing version in which at every iteration T changes which is **Task 2 Part 1**.

Dataset	T_{init}	T_{min}	α_{SA}	α	Swaps	Convergence Time	Min Edge-cut
3elt	1	0.001	0.95	2	4714510	NA	2482
add20	1	0.001	0.95	2	2393075	NA	2444
facebook	1	0.001	0.95	2	62690050	NA	148426
twitter	1	0.001	0.95	2	2633315	NA	41685

We can observe from the table above that none of the datasets have achieved convergence with this method of simulated annealing. The figures below presents the change of each metrics over the iterations for the respective datasets.

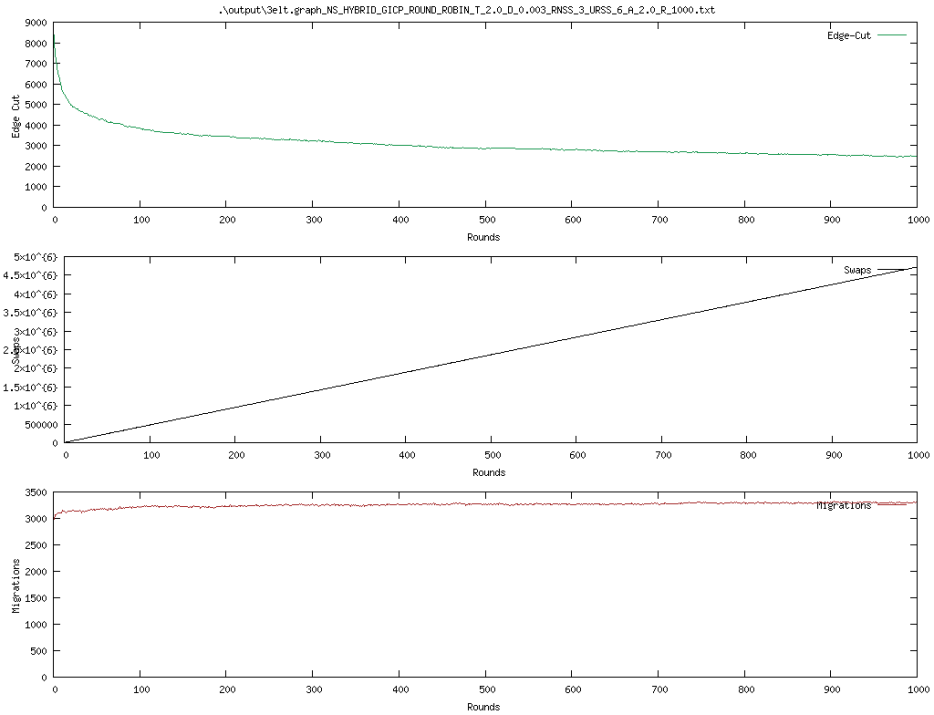

Figure 5: 3elt - Task 2 Part 1

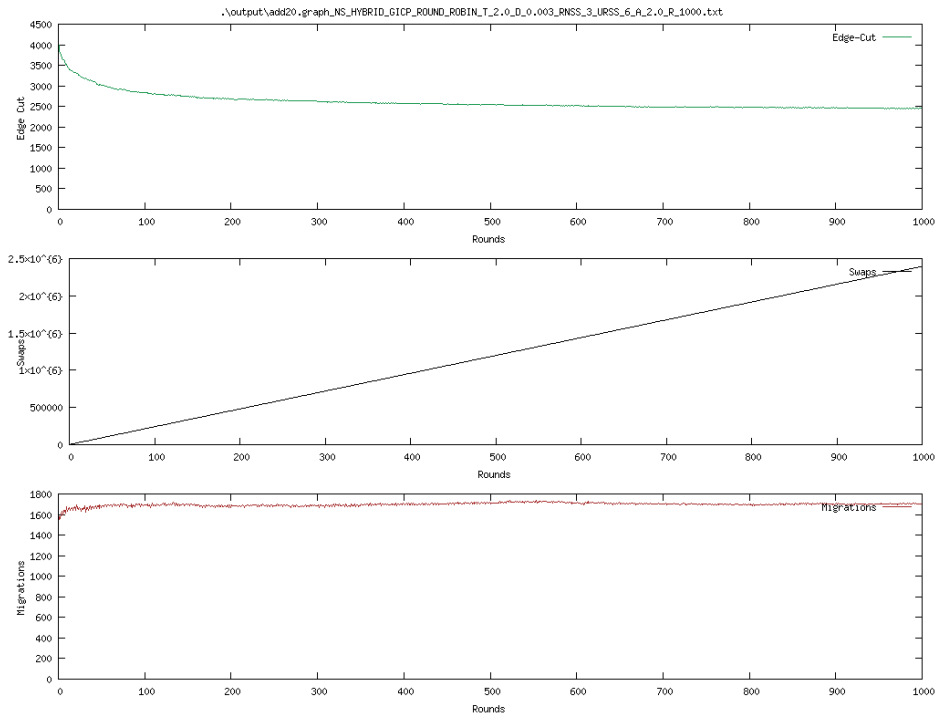

Figure 6: add20 - Task 2 Part 1

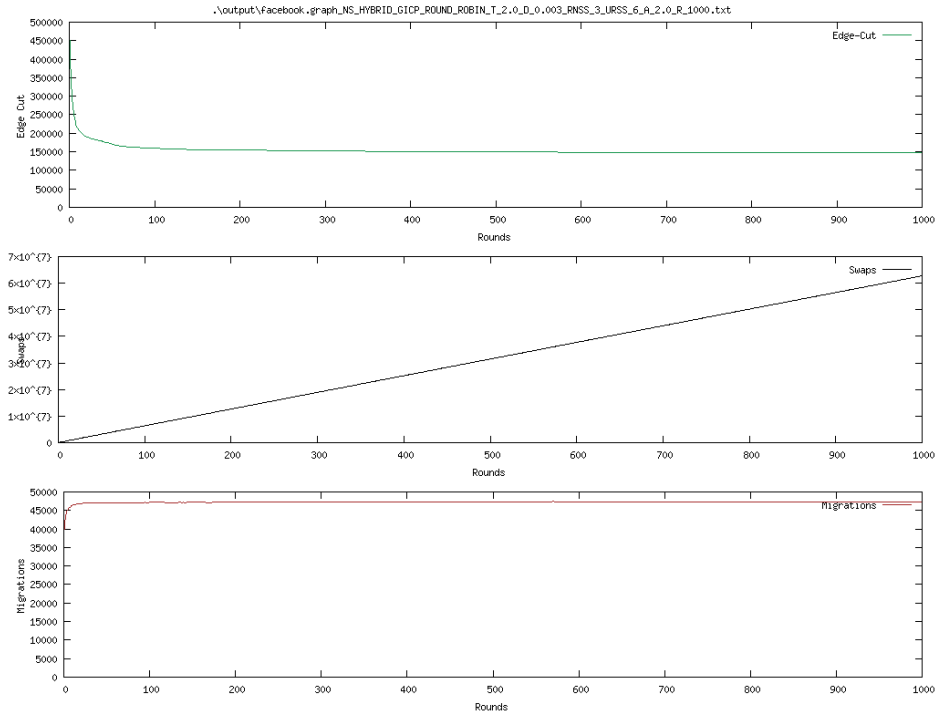

Figure 7: facebook - Task 2 Part 1

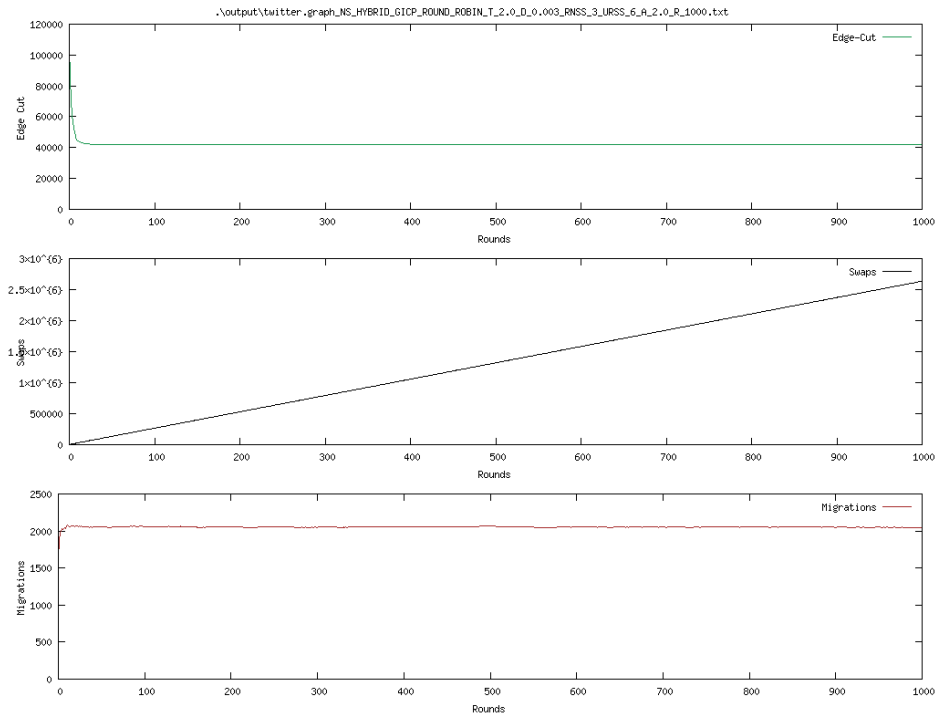

Figure 8: twitter - Task 2 Part 1

The table below presents the results for **Task 2 Part 2** where T is re-initialized once it has reached convergence.

Dataset	T_{min}	δ	α	Interval	Swaps	Convergence Time	Min Edge-cut
3elt	2	0.01	2	200	2402260	~ 100	1948
add20	2	0.01	2	200	1429690	~ 100	1976
facebook	2	0.01	2	200	32163436	~ 100	123472
twitter	2	0.01	2	200	1345756	~ 100	41344

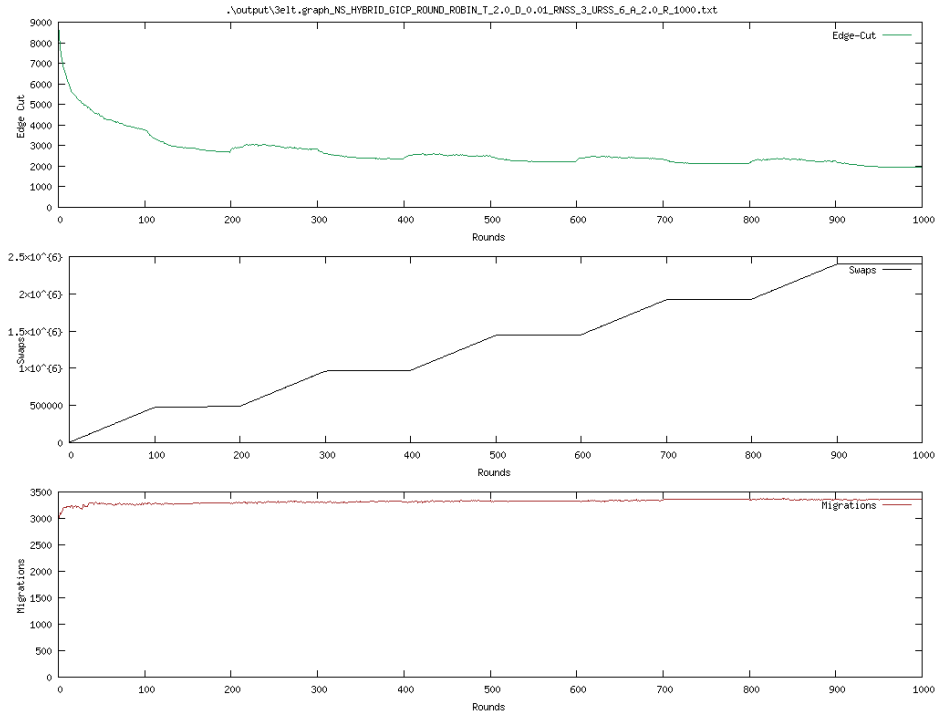

Figure 9: 3elt - Task 2 Part 2

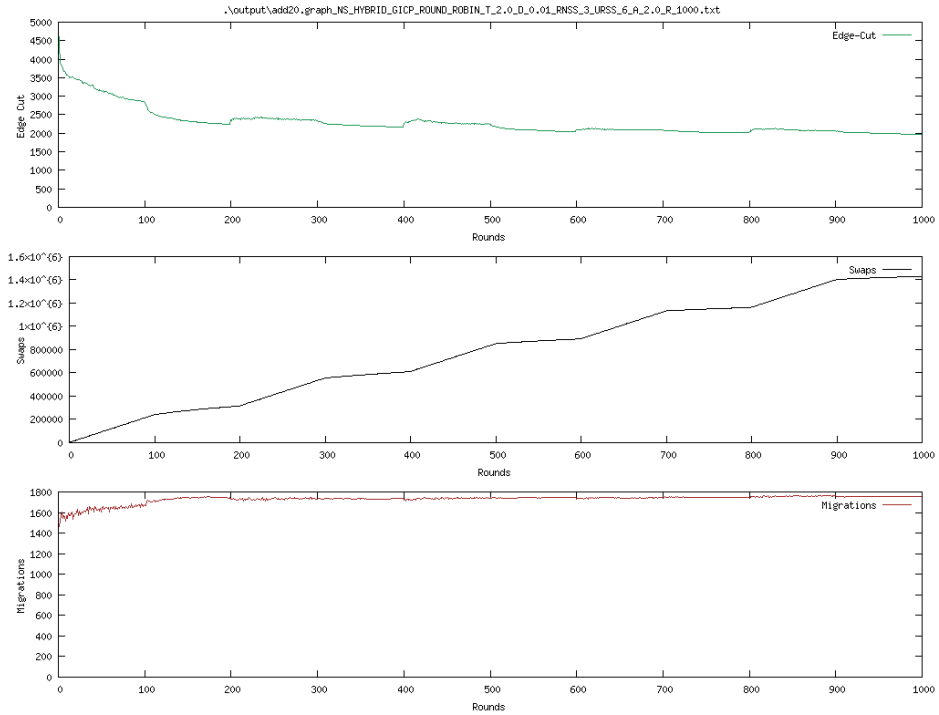

Figure 10: add20 - Task 2 Part 2

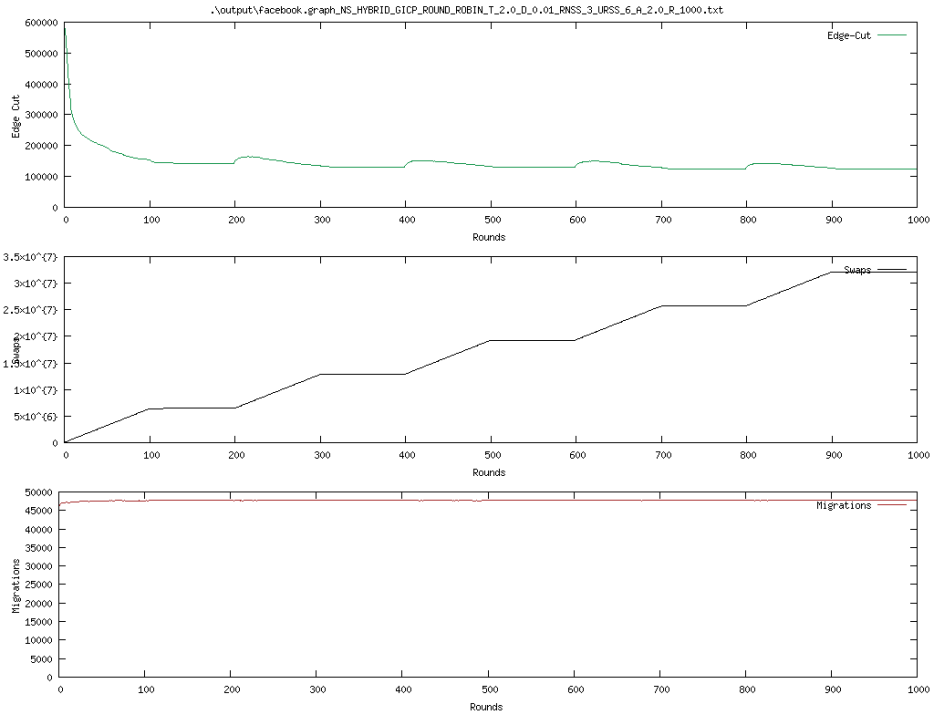

Figure 11: facebook - Task 2 Part 2

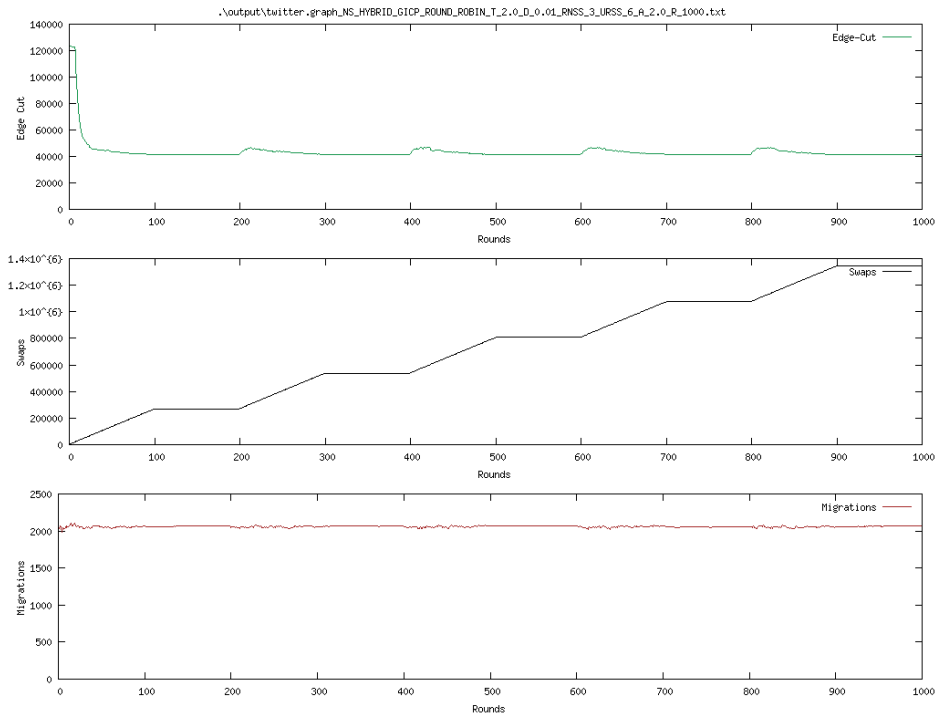

Figure 12: twitter - Task 2 Part 2

5.3 Hyper-parameter Tuning

In order to try to minimize the edge-cut we have considered the graph 3elt and we have performed parameter tuning on it. We have considered the following parameters to tune: δ, α and how often perform the restart operation.

Dataset	T_{min}	δ	α	Interval	Swaps	Convergence Time	Min Edge-cut
3elt	2	0.01	1	300	1909272	~ 100	2622
3elt	2	0.005	2	400	2849811	~ 200	2238
3elt	2	0.01	2	200	2402260	~ 100	1948
3elt	2	0.05	1.5	100	1024257	~ 20	1713
3elt	2	0.05	2.5	100	1021556	~ 20	1583
3elt	2	0.025	2	100	1966606	~ 40	1581
3elt	2	0.05	2	200	524780	~ 20	1517
3elt	2	0.05	2	150	727635	~ 20	1488

6 Optional Task

The last part of this project is to explore ways to improve the Ja-Be-Ja algorithm implemented. Some variations could be to define our own acceptance probability function or the way the algorithm functions. In our case, we performed 2 experiments: 1) change of colors of the node if it improves instead of a mutual swap and 2) change the acceptance probability function to softmax function.

Method 1 resulted in better edge cuts but it generated a highly imbalanced dataset which defeats the goal of Ja-Be-Ja to maintained a balanced K-way partitioning. For datasets that are balanced to begin with i.e. 3elt, it does not result in any improvements at all.

Method 2 uses softmax function keeps the same range between 0 to 1 while performing some normalization for the acceptance probability. This method has resulted in better results as shown in the table below.

Dataset	T_{init}	T_{min}	α_{SA}	α	Swaps	Convergence Time	Min Edge-cut
3elt	1	0.001	0.95	2	3491009	800	1589
add20	1	0.001	0.95	2	1777522	800	2186
facebook	1	0.001	0.95	2	43760192	800	139224
twitter	1	0.001	0.95	2	1774262	800	41793

As we can observe from the table above, the improvements with the change in acceptance probability is not deterministic. This is because it is dependent on type of dataset (the gap on which softmax is computed on). Therefore, we do not observe a single factor of improvement but rather an improvement one way or another on all the datasets such as reduced number of swaps or reduced number of minimum edge cuts.

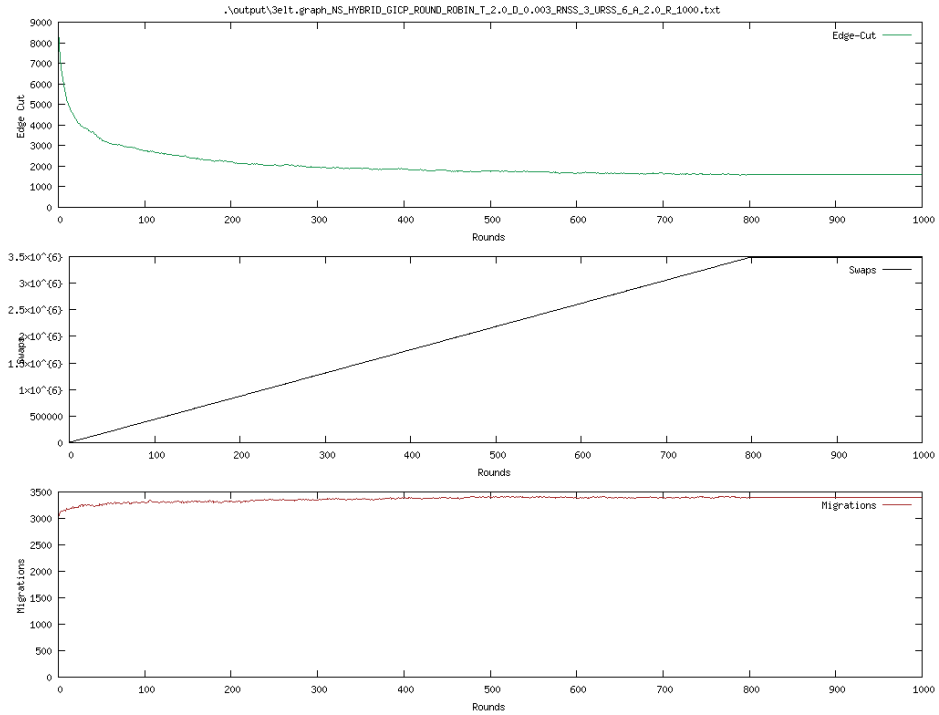

Figure 13: 3elt - Bonus Task

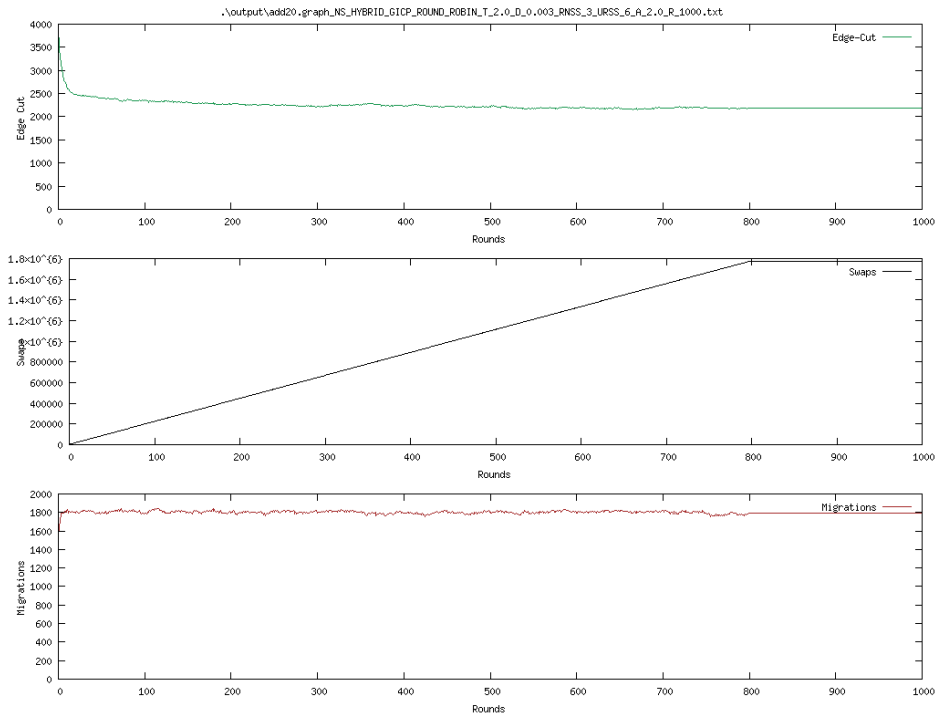

Figure 14: add20 - Bonus Task

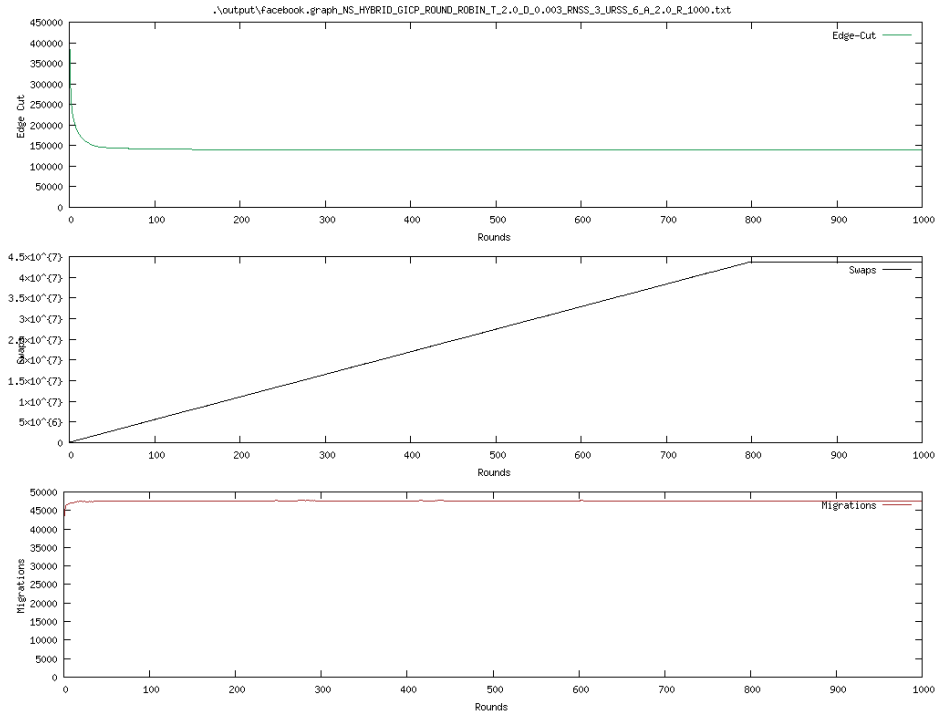

Figure 15: facebook - Bonus Task

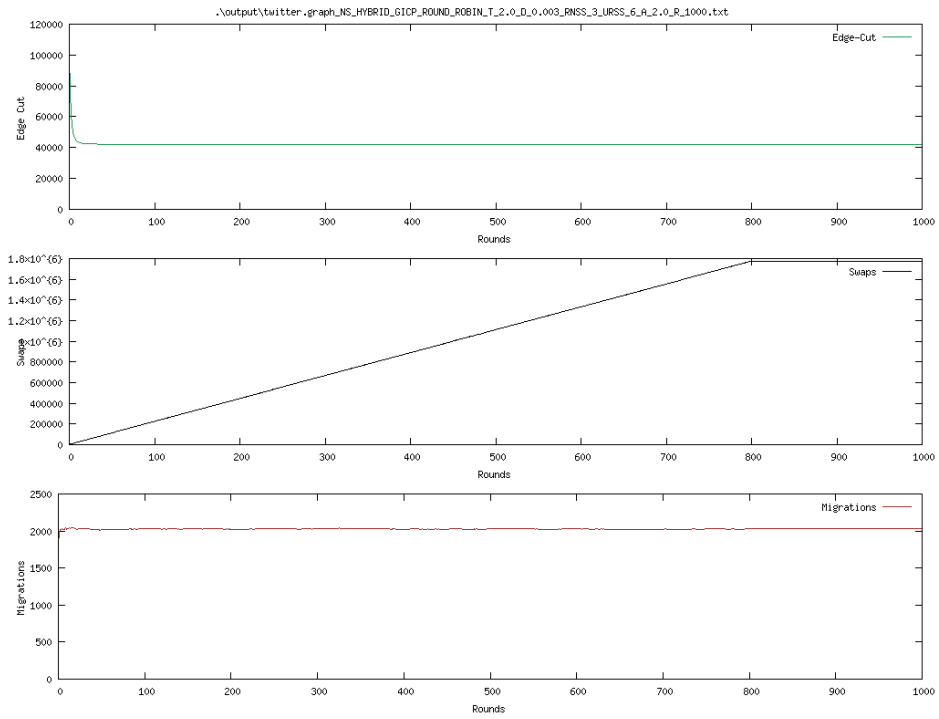

Figure 16: twitter - Bonus Task

7 How to run the code

The implementation of the JaBeJa is written in Java. The following commands need to be run in the command shell in order to obtain the desired input. First, the Java code must be compiled and then it is possible to run it. By adding the graph flag in the run command is possible to specify the desired input graph datasets, e.g. 3elt. The results of the program execution will be saved in the ./output directory as 'filename'.txt
Gnuplot, a portable command-line driven graphing utility, has been used to launch the last command which creates .png image of the results obtained in the previous step. Once Gnuplot is properly installed and added to the path, it is sufficient to launch the last command replacing *'filename'* with the the correct filename and it will save the plots in a file called graph.png.

```
>> ./compile.sh
>> ./run.sh -graph ./graphs/3elt.graph
>> ./plot.sh output/<filename>.txt
```

References

[1] JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning, Fatemeh Rahimian, Amir H. Payberah, Sarunas Girdzijauskas, Mark Jelasity, Seif Haridi, 2013