Report 5: JaBeJa Report

Gard Aasness, Fabian Zeiher, Group 62

December 8, 2022

1 Description

In this assignment the goal was to better understand distributed graph partitioning and get some experience with the distributed gossip based algorithm, JaBeJa. However it should be noted that in our experiments we only used a one-host-one-node setup. The assignment provided us with a skeleton and bunch of methods for us to use in order to implement two important methods in the JaBeBa algorithm: *sampleAndSwap* and *findPartner*. After implementing these functions and analyzing the results, we modified the algorithm to find the smallest edge cuts for a graph. These results was also analyzed.

2 How to run

The code was implemented using Java. It is using Gnuplot to visualize the graphs. The following description is 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

3 Implementation

3.1 Task 1

In task 1 we implemented *findPartner* and *sampleAndSwap*. The paper describing the JaBeBa algorithm provided us with pseudocode of what the

methods would look like. The comand line parameters when running the code are δ (delta), α (alpha) and T (temperature). In the default version of the algorithm, the temperature is decreased by a constant (delta), making the temperature decrease in a linear pattern, until it reaches a minimum value, which is 1. Results can be seen in figure 1.

3.2 Task 2

In the second task we experimented with the results while modifying the algorithm.

3.2.1 2.1

A problem when dealing with large number of nodes, it can become an optimization problem for the algorithm. To deal with this, we implemented a simulated annealing algorithm. This is done in order to avoid being stuck in a local maxima early on in the process, by inserting just enough randomness into the algorithm. In order to decide whether to swap nodes or not, we implemented the acceptance probability function described in this paper[1]. The function was given as follows: $a = e^{\frac{cNew - cOld}{T}}$ where c is a cost function. Because instead of a cost function we have more of a "benefit"-function producing integer values, directly applying the acceptance probability function from the article would result in a wrong distribution of the acceptance probability. We therefore changed the formula slightly to:

$$a = e^{\frac{\frac{1}{bOld} - \frac{1}{bNew}}{T}}$$

where b is the benefit function. This forumla results in the correct distribution of the acceptance probability between 0.0 and 1.0. Results can be seen in figure 4.

3.2.2 2.2

For the simulated annealing algorithm implemented in the previous chapter, when the temperature reaches its minimum value, the edge cut converges as there will be no more bad swaps. To experiment further, we tried restarting the simulated annealing algorithm by resetting temperature back to its initial value when hitting the final value, see figure 3. To analyze which parameter combinations gave the lowest edge cut we employed a bash script to test different combinations. We changed the initial temperature as well as delta (the number of temperature decrease in each round). Our best result can be seen in figure 2.

[1] `http://katrinaeg.com/simulated-annealing.html`

3.3 Task 2 (Optional)

In our improvement of the JaBeJa algorithm we combined the updated simulated annealing algorithm with restarts. The annealing allows the algorithm to avoid getting stuck in a local minima. Restarting the algorithm after converging supports this behaviour further by introducing more randomness. One problem with restarting is that the algorithm might never converge to a stable result. We combat this by continually lowering the intensity of the restarts. Therefore after a certain amount of time the algorithm will converge to a final solution. This combination between the updated simulated annealing and restarts has given us the lowest amount of edge cuts out of all the experiments we have done, with a total amount of under 800, see figure 5.

4 Plots

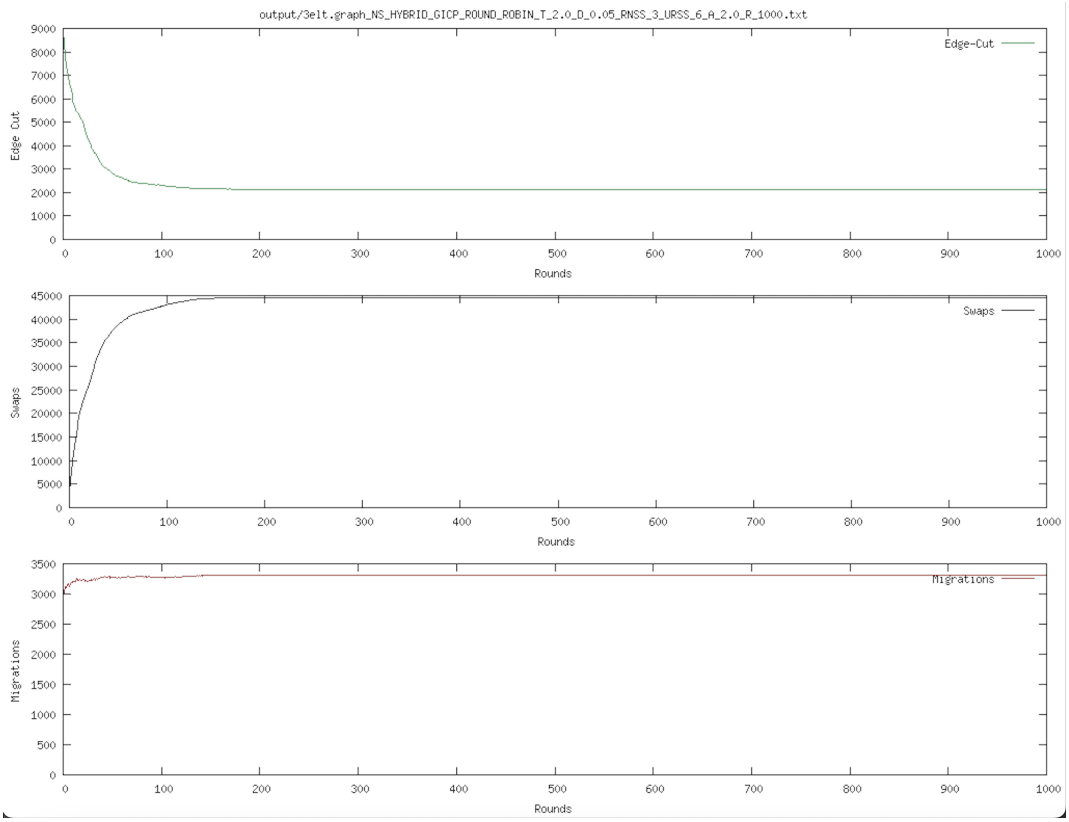

Figure 1: 3elt.graph with linear simulated annealing and no restarts, where $\delta = 0{,}05$, T = 2.

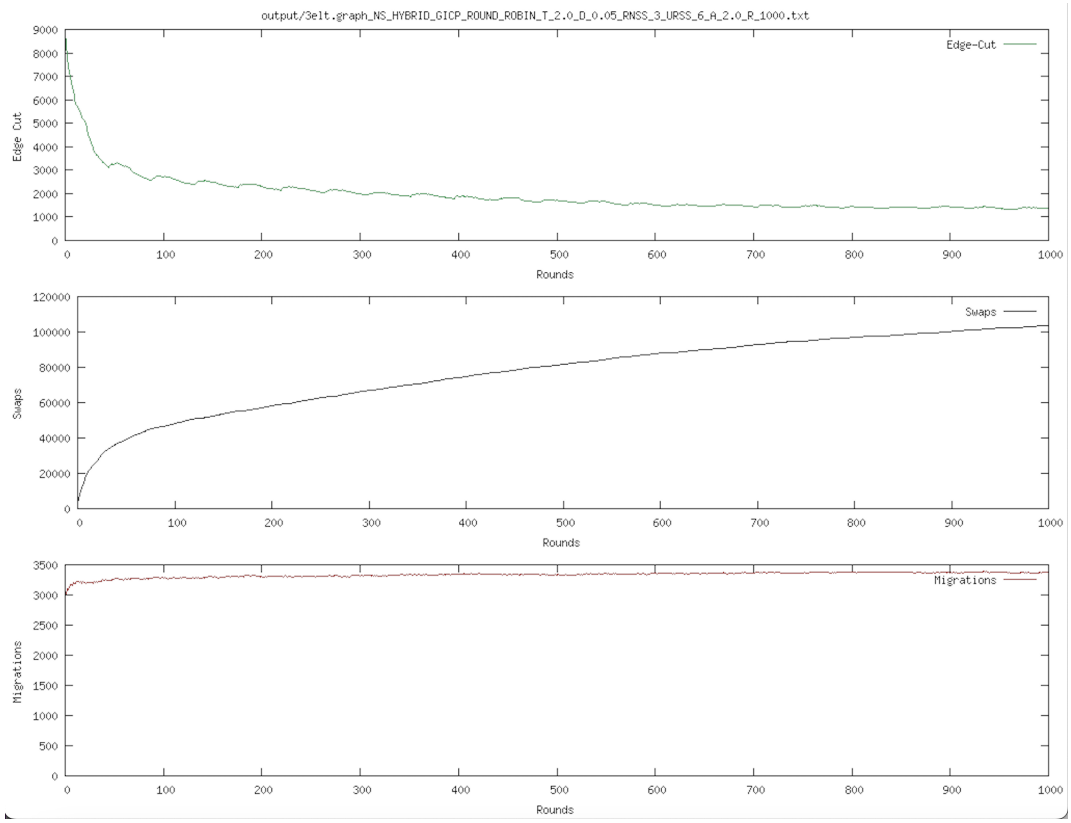

Figure 2: 3elt.graph with linear simulated annealing and restarts, where δ = 0,05, T = 2.

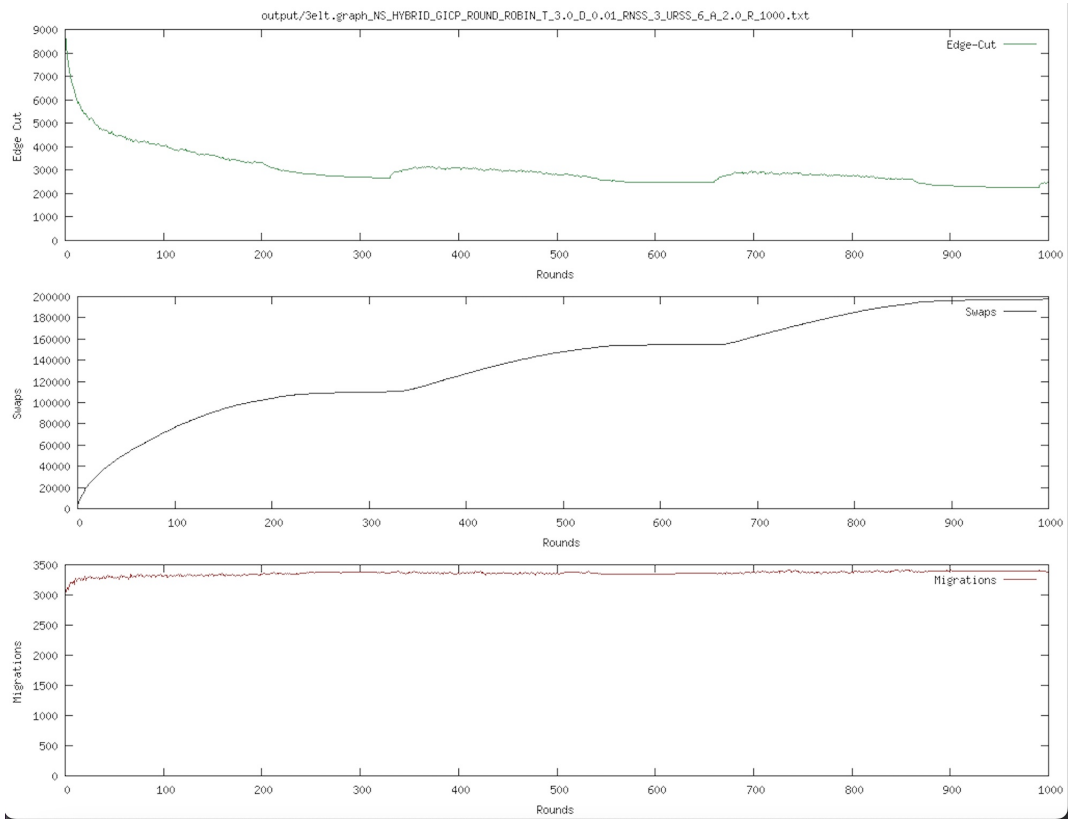

Figure 3: 3elt.graph with linear simulated annealing and restarts, where δ = 0,01, T = 3.

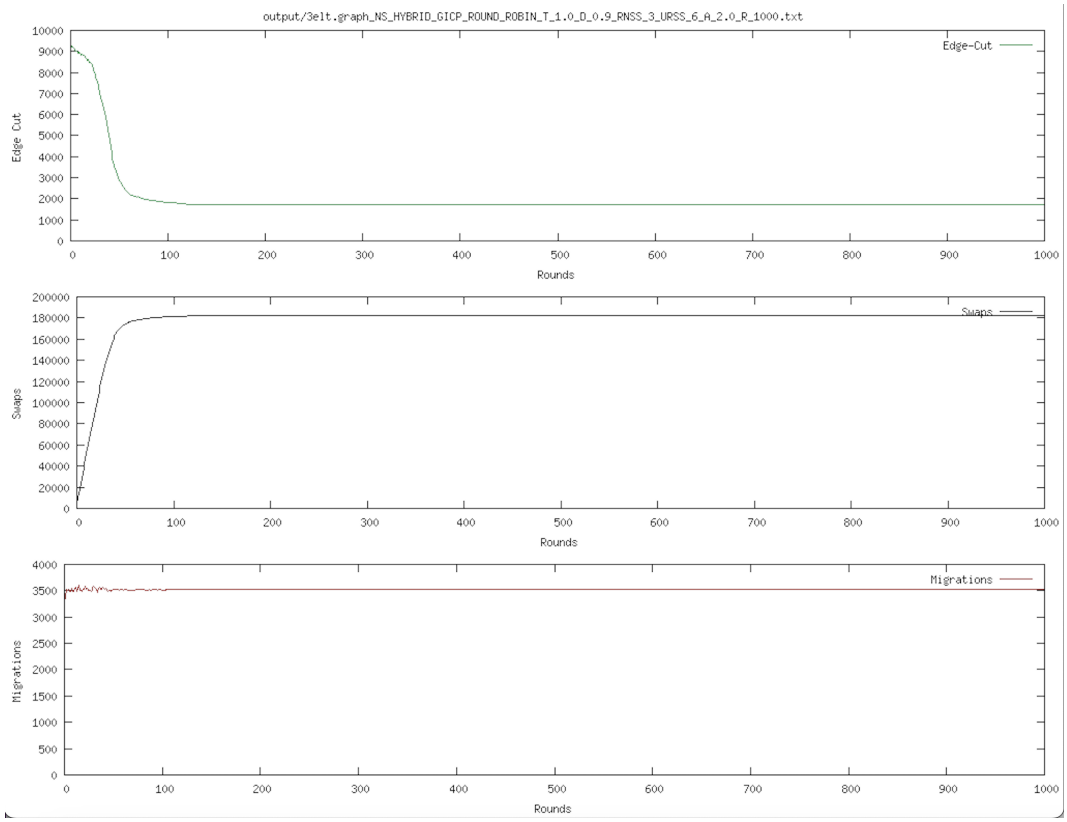

Figure 4: 3elt.graph with exponential simulated annealing and no restarts, where $\delta = 0{,}9$, T = 1.

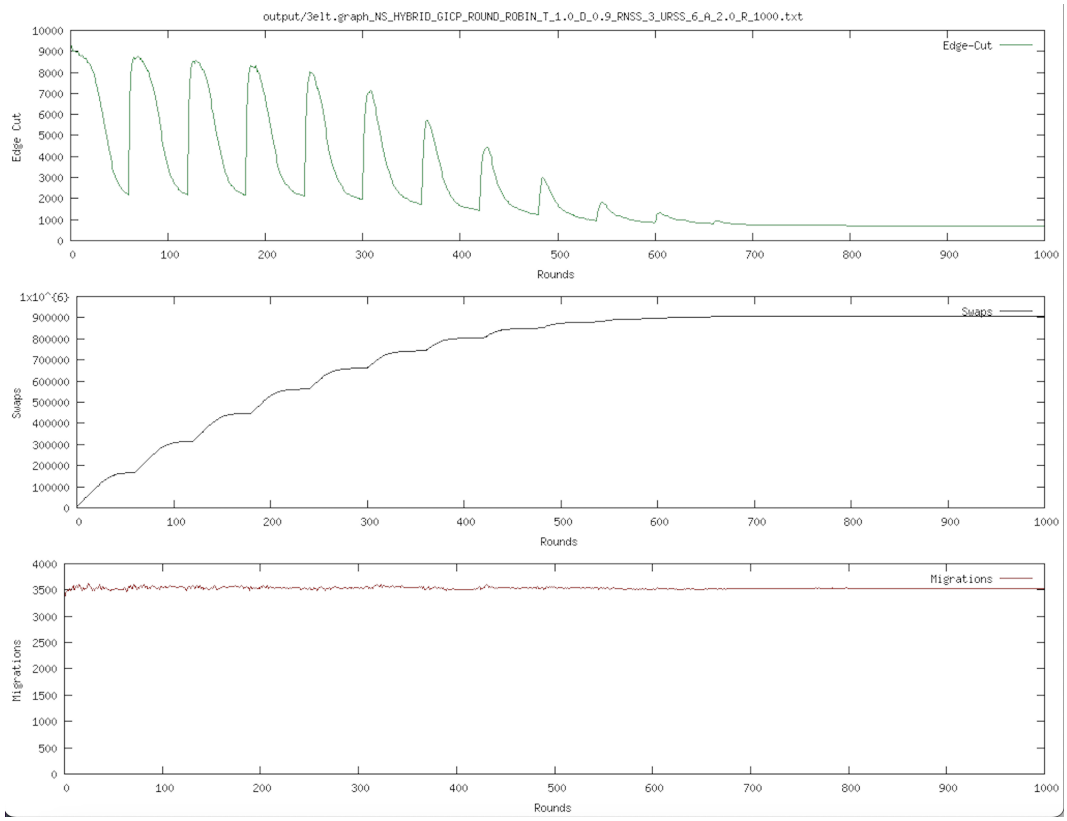

Figure 5: 3elt.graph with exponential simulated annealing and restarts with a lower temperature than the previous temperature value for each restart, where $\delta = 0{,}9$, T = 1.