Homework 5

K-way Graph Partitioning Using JaBeJa

Badai Kesuma Hamid Dimyati

December 2020

1. Code commentary

This assignment consists of two tasks. In the first task, we are to implement the Ja-Be-Ja algorithm [1]. We need to complete the `Jabeja.java` program by filling the `sampleAndSwap(int)` method and the `findPartner(int, Integer[])` method as described in the paper. In the second task you will tweak different Ja-Be-Ja configurations in order to find the smallest edge cuts for the given graphs.

2. How to run

a. Clone https://github.com/hamiddimyati/id2222-data-mining-advanced.git

b. Move to assignment-5 folder:

```
cd id222-data-mining-advanced/assignment-5
```

c. Make sure maven and gnuplot are already installed. Open terminal and run these commands for 3 graph files: 3elt, add20, and twitter graphs

```
./compile.sh
./run -graph.sh ./graphs/graph_file.graph
./plot.sh output/result_file.txt
```

d. Modify all the parameters through the CLI.java and Jabeja.java for these configurations

   i. Configuration 1 - linear function to decrease the temperature without reset
      Keep the Temperature = 2 and Rounds = 1000
      Try Delta = {0.01, 0.001, 0.003}

   ii. Configuration 2 - linear function to decrease the temperature with reset
      Keep the Temperature = 2 and Rounds = 1000
      Try Delta = {0.001, 0.003}

   iii. Configuration 3 - exponential function to decrease the temperature
      Keep the Temperature = 1 and Rounds = 1000
      Try Delta = {0.50, 0.99}

3. Results

In this section, we discuss the Jabeja's performance with different parameters, acceptance probability functions, and simulated annealing algorithms. For the linear simulated annealing algorithm, we use linear acceptance function:

$$(new * T > old) \ \&\ (new > highest)$$

For the exponential simulated annealing algorithms, the temperature decreases exponentially by $T_{k+1} = T_k * \square$ with acceptance probability function

$$e^{\frac{Enew - Eold}{T}}$$

a. 3elt graph

In this graph, we first try the given parameter from the github codes, linear simulated annealing with Delta = 0.003, T = 2, Rounds = 1000. In the following figure we can see the result

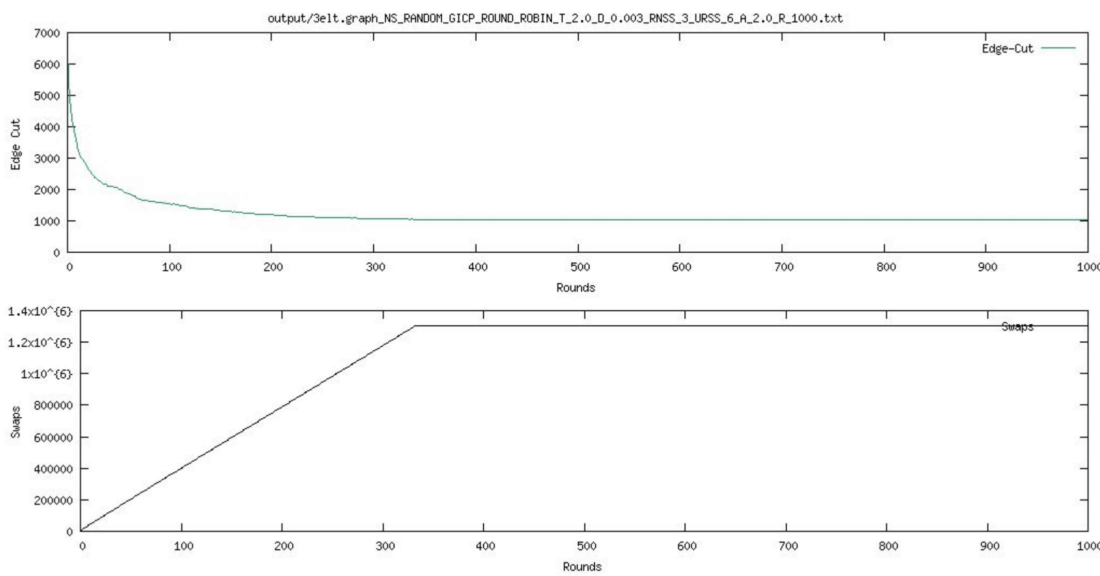

Figure 1. Linear simulated annealing (T = 2, D = 0.003, R = 1000)

We can see the algorithm converges at round 330 and the edge-cut descents steadily. But to really measure how good it's performance, we will compare against other parameters with the bigger Delta = 0.01 (which converges at round 100) and smaller Delta = 0.001 (which converges at round 1000).

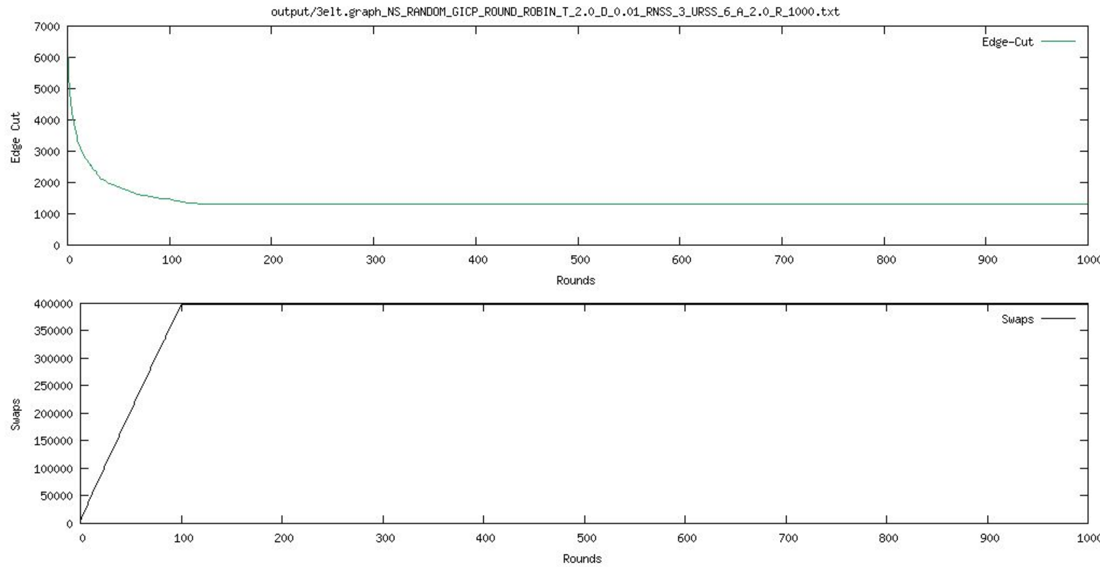

Figure 2. Linear simulated annealing (T = 2, D = 0.01, R = 1000)

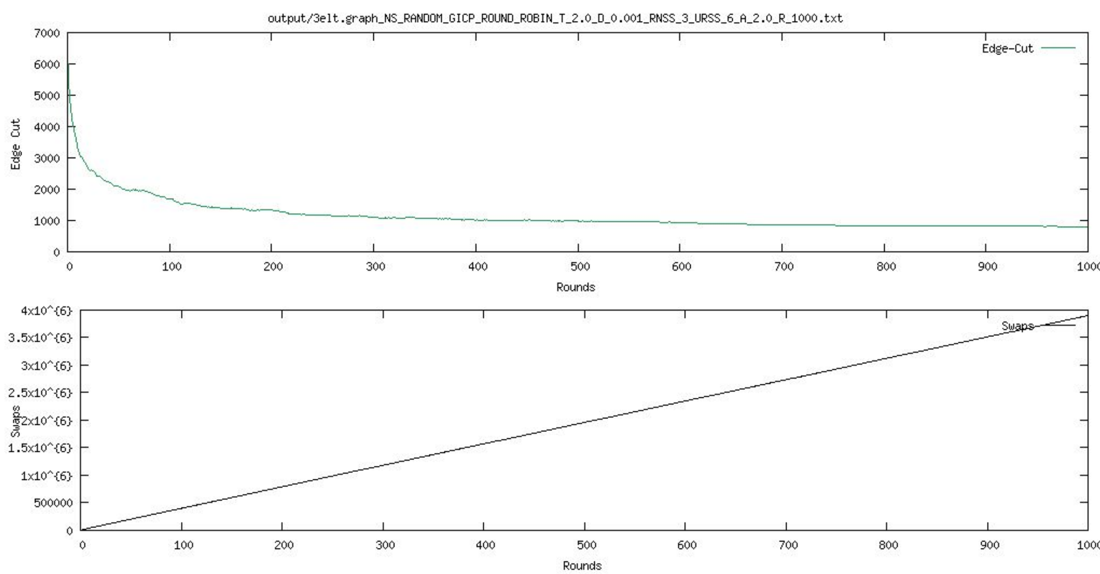

Figure 3. Linear simulated annealing (T = 2, D = 0.001, R = 1000)

In this configuration, we can clearly see that the slower the temperature decreases will have better performance with lower edge-cut. For this reason, the other graphs will use Delta = 0.001 for linear simulated annealing without reset.

With smaller T, the edge-cut will descend more steadily. Thus, we try to restart the temperature when it reaches T = 1. In this configuration, Delta = 0.003 will give 2 resets at round 330 and 660 while Delta = 0.01 will give 9 resets at round 100, 200, .. , 900. We can inspect the performance in the following figures.

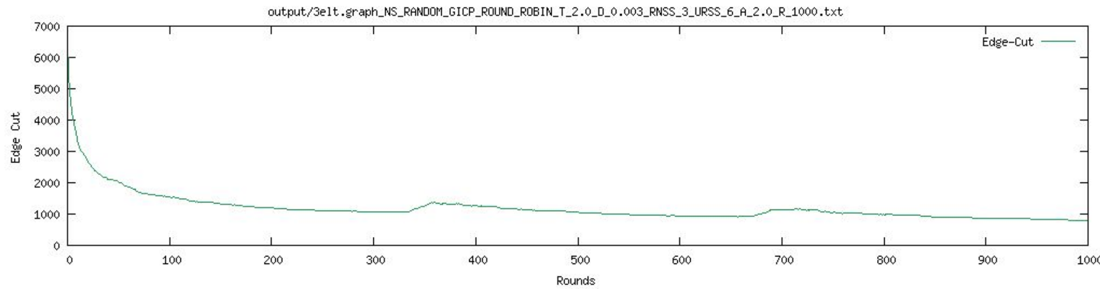

Figure 4. Linear simulated annealing (T = 2, D = 0.003, R = 1000) with reset

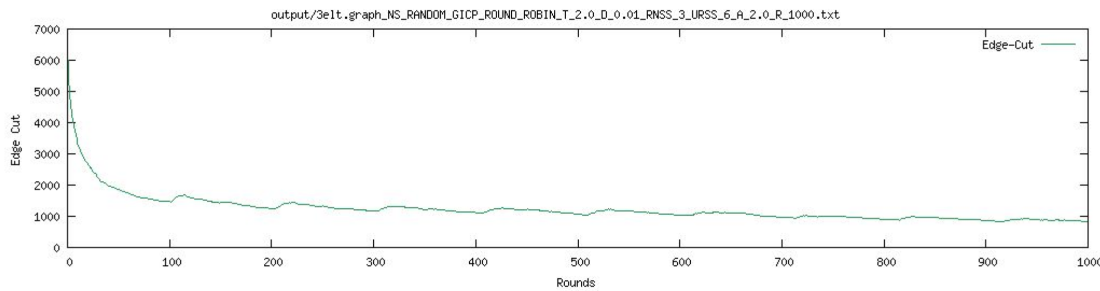

Figure 5. Linear simulated annealing (T = 2, D = 0.01, R = 1000) with reset

We can see that in each reset, the edge-cut will increase a little bit and then decrease again. It was meant so that the algorithm does not get stuck in the local minima. At round 1000, we can see that Delta = 0.003 performed slightly better than Delta = 0.01.

Lastly, we had to change the simulated annealing algorithm where T decreases exponentially, using Temperature = 1 as the maximum allowed, Rounds = 1000, and the smaller Delta = 0.5 that has shorter convergence time and the bigger Delta = 0.99 that has enough flexibility to search the solutions space.

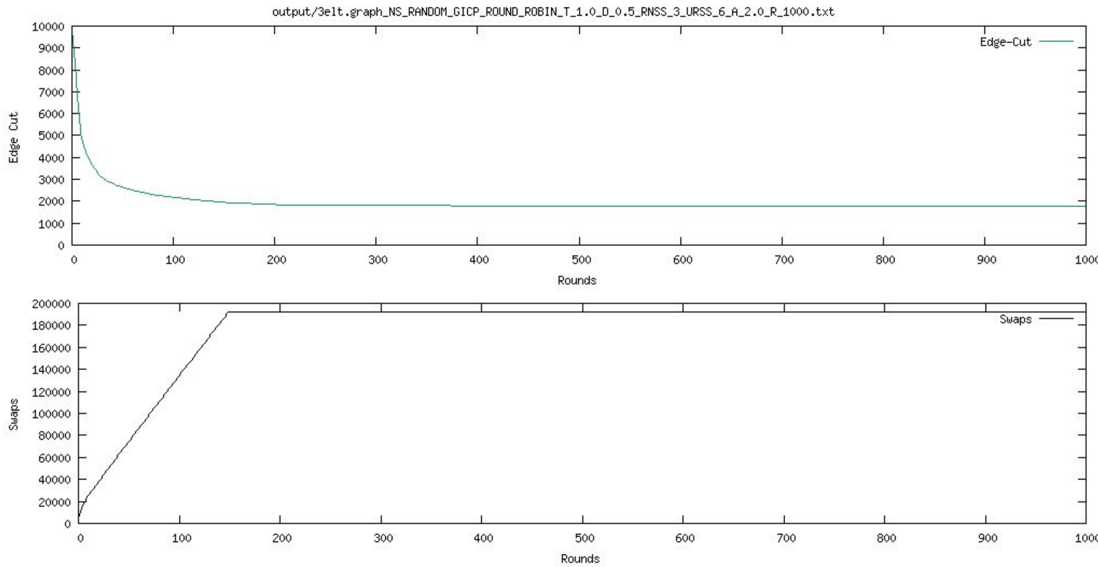

Figure 6. Exponential simulated annealing (T = 1, D = 0.5, R = 1000)

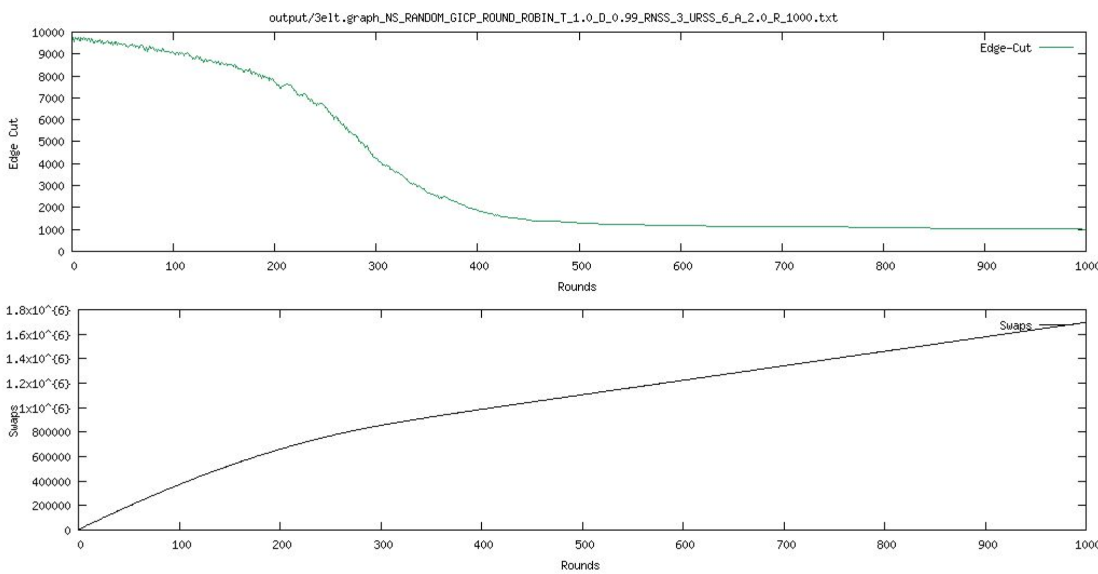

Figure 7. Exponential simulated annealing (T = 1, D = 0.99, R = 1000)

Bigger Delta gives better performance to the edge-cut. But it is still worse compared to linear simulated annealing (T = 2, D = 0.001, R = 1000) without reset and linear simulated annealing (T = 2, D = 0.003, R = 1000) with reset that has similar performance. But, we can get better performance with reset if we have more rounds, since the performance without reset will only descend slightly with T = 1.

b. add20 graph

Using the same configurations with the previous graph, we can see the result of each configuration and each parameter.

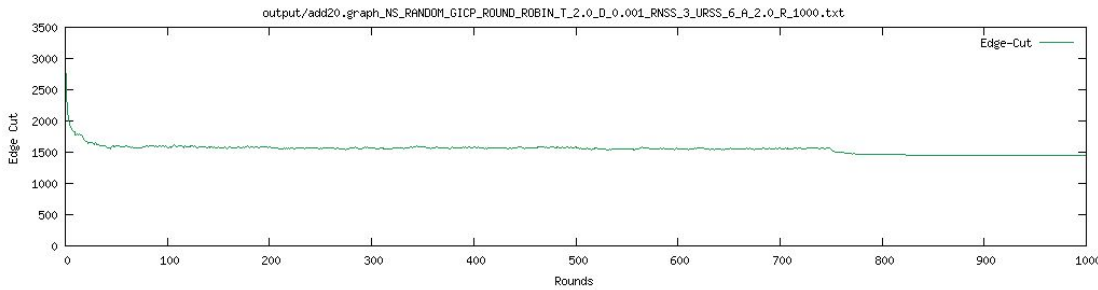

Figure 8. Linear simulated annealing (T = 2, D = 0.001, R = 1000) without reset

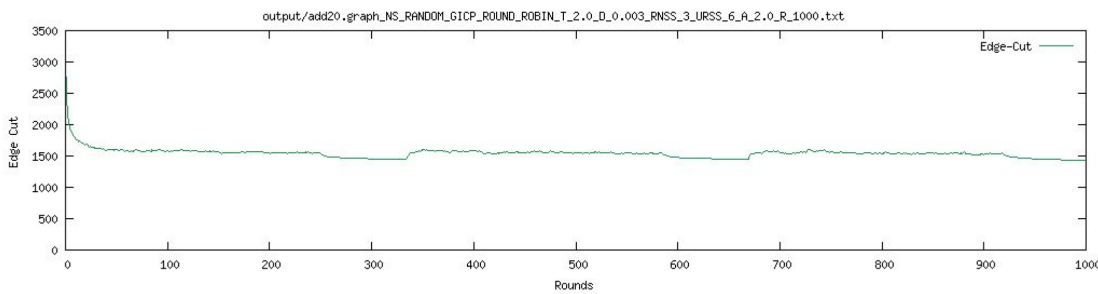

Figure 9. Linear simulated annealing (T = 2, D = 0.003, R = 1000) with reset

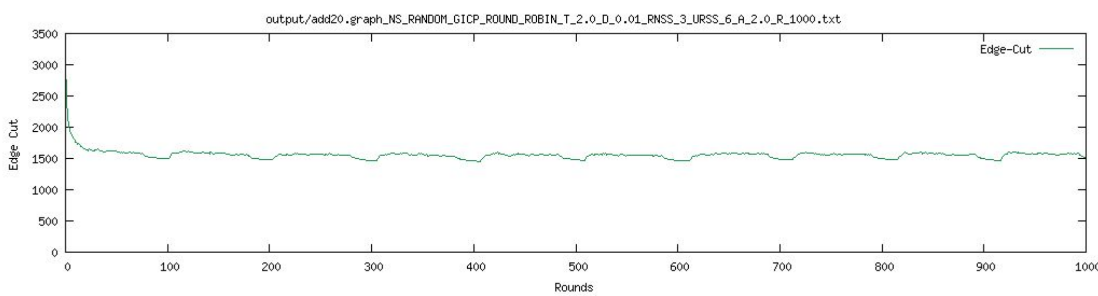

Figure 10. Linear simulated annealing (T = 2, D = 0.01, R = 1000) with reset

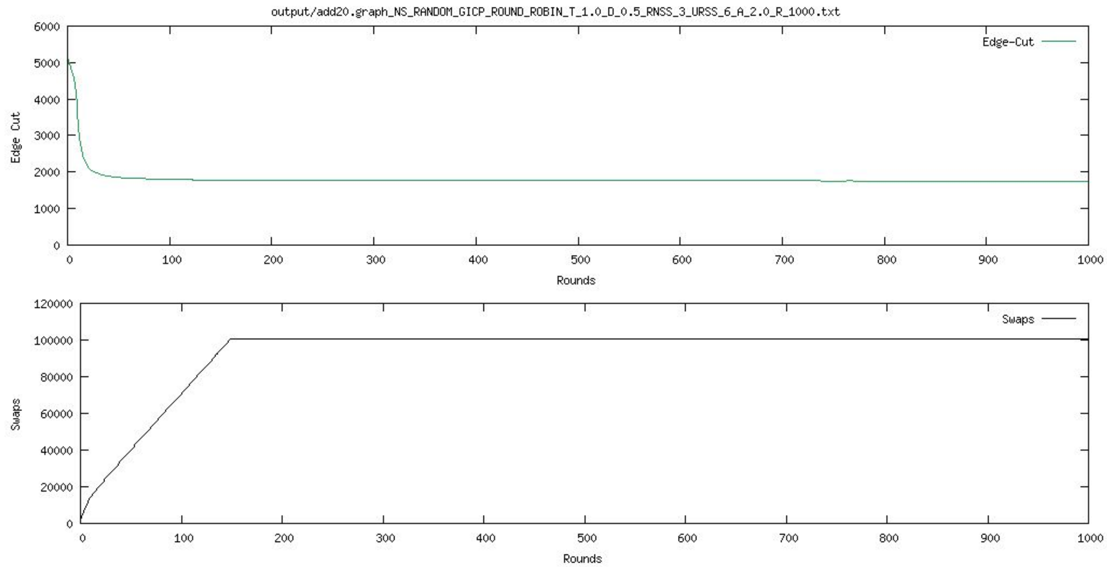

Figure 11. Exponential simulated annealing (T = 1, D = 0.5, R = 1000)

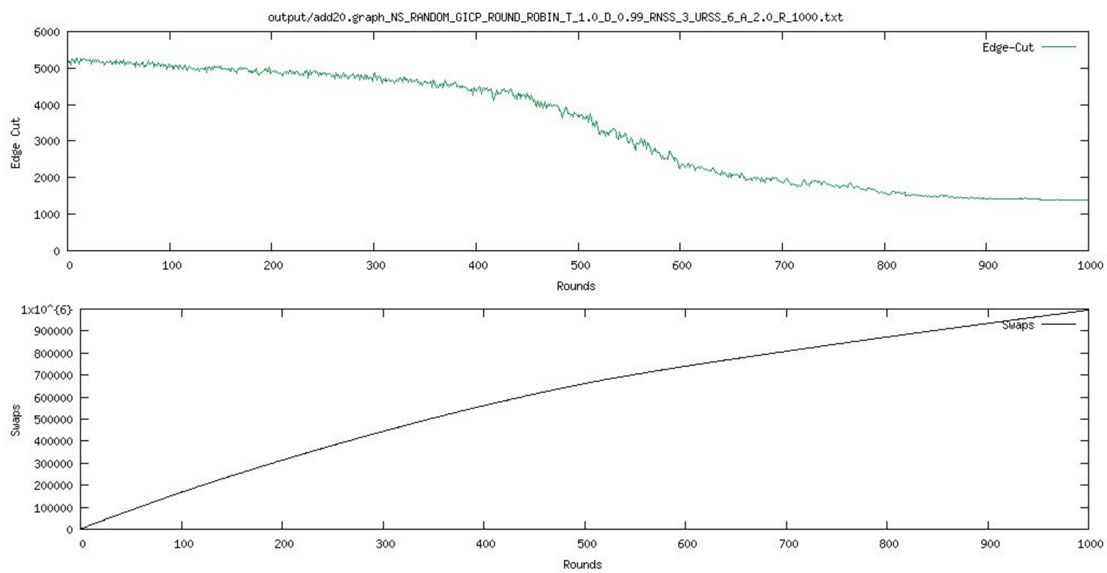

Figure 12. Exponential simulated annealing (T = 1, D = 0.99, R = 1000)

Different from the previous graph, we can see in the add20 graph that exponential simulated annealing algorithm gives better performance compared to the linear simulated annealing, with the best performance achieved by Delta = 0.99 with edge-cut around 1300 in rounds 1000. While the linear algorithm only reached edge-cut around 2000.

c. twitter graph

Using the same configurations with the previous graph, we can see the result of each configuration and each parameter.

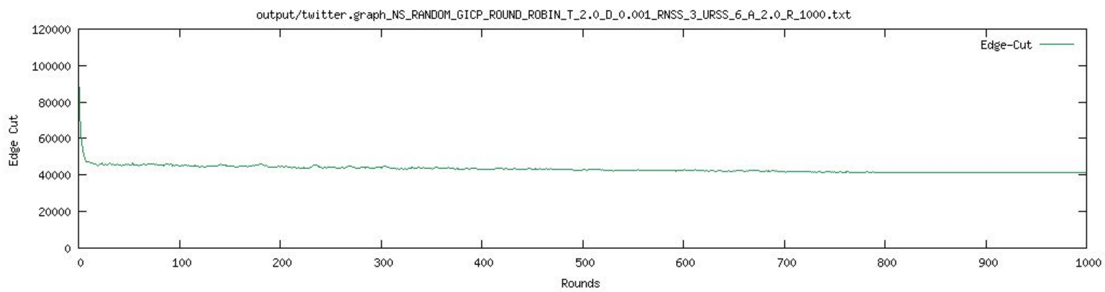

Figure 13. Linear simulated annealing (T = 2, D = 0.001, R = 1000) without reset

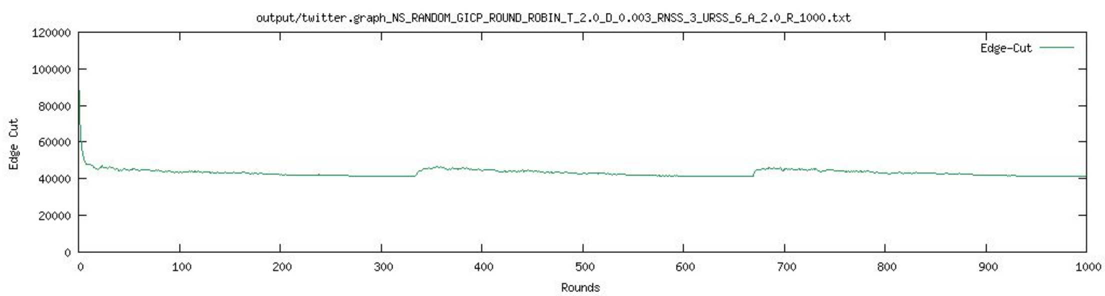

Figure 14. Linear simulated annealing (T = 2, D = 0.003, R = 1000) with reset

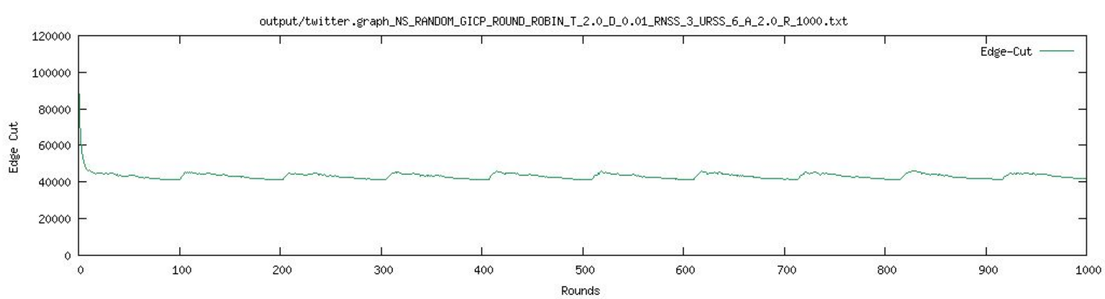

Figure 15. Linear simulated annealing (T = 2, D = 0.01, R = 1000) with reset

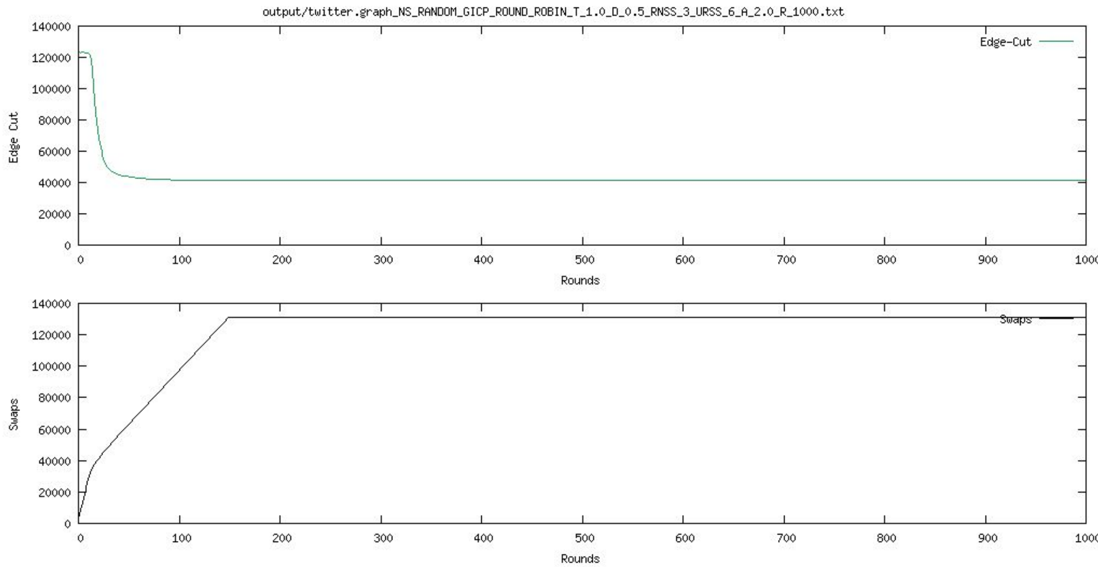

Figure 16. Exponential simulated annealing (T = 1, D = 0.5, R = 1000)

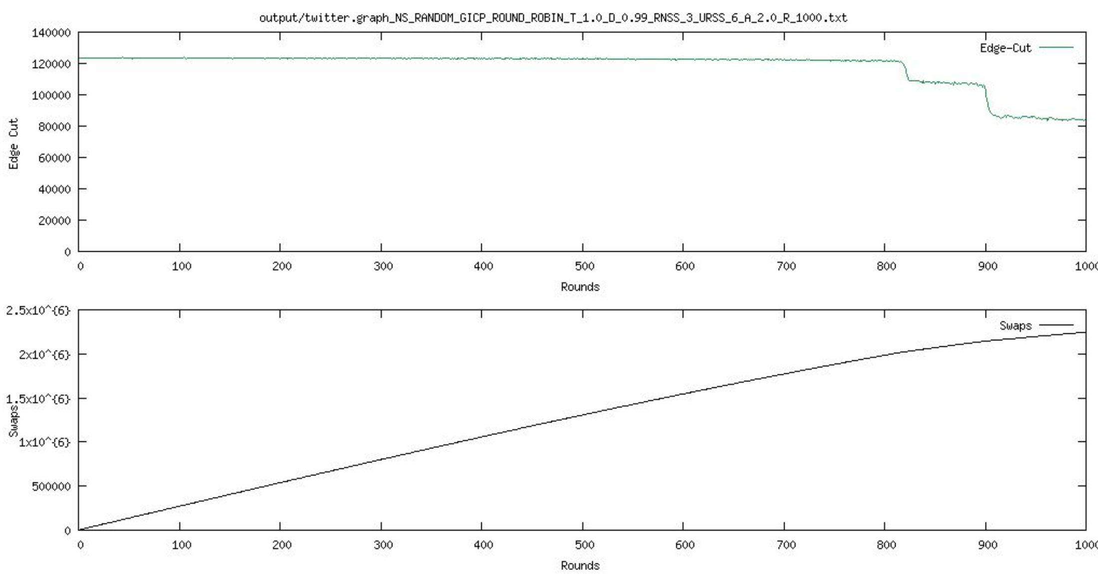

Figure 17. Exponential simulated annealing (T = 1, D = 0.99, R = 1000)

In the twitter graph, all the linear simulated annealing algorithms perform better compared to the exponential simulated annealing algorithm, with slight differences between each linear simulated annealing parameter.

References

[1] F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity and S. Haridi, "JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning," 2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, Philadelphia, PA, 2013, pp. 51-60, doi: 10.1109/SASO.2013.13.