Homework 5 - Ja-Be-Ja Graph Partioning

Perttu Jääskeläinen Group 11 Gabriele Morello
perttuj@kth.se morello@kth.se

December 12, 2022

Contents

1 Introduction

In this homework, we implemented an algorithm for creating a balanced graph partitioning in a distributed fashion, intended to support very large graphs that may not fit in the memory of a single machine. To do this, we implemented the Ja-Be-Ja algorithm [1], which uses simulated annealing and local processing in nodes to converge into a partitioned graph.

The implementation was done in Java, extending the implementation provided for us [2].

2 Implementation

We separated the implementations in different files in order for them to be more easily distinguishable. The first task is implemented in **JabejaTask1.java**, while the second one, *you guessed it*, is implemented in **JabejaTask2.java**.

To tweak the results for both implementations, we used various parameters to obtain different graphs. For example, for **Task2**, we used **beta = 0.9** and **beta = 0.8** to see how the convergence of the implementation was impacted (see Results in chapter 4 for details). For **Task1**, we tweaked the simulated annealing restart ratio - i.e. when the simulated annealing would "start over" after it reached it's lowest value and Ja-Be-Ja converged. For this, we used

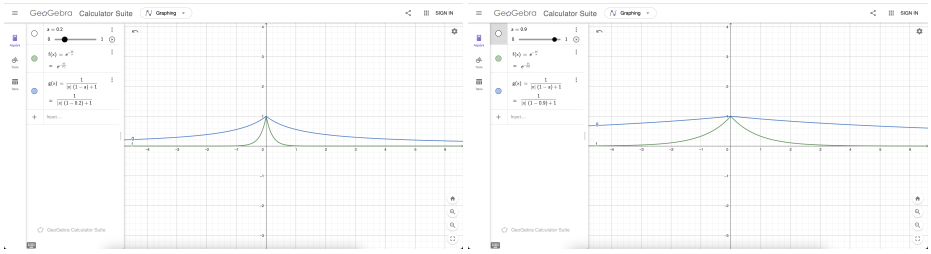

Figure 1: *The distribution of our custom probability function (blue) values compared to the one provided for us (green) - on the left, we have the scale for lower values of T, and on the right for the higher values. Note that we never reach the negative values, since we take the absolute value of X (which is the difference in degrees when swapping vs. not swapping colors).*

ratios of **100** and **200** rounds after the lowest temperature value was reached (since Ja-Be-Ja converges shortly afterwards). We did the similar changes for **Task2**, but instead of setting the temperature to some high value, we resetted it back to 1 (100% probability to make a bad swap).

To achieve the extra task (seen in function **calculateCustomAcceptanceProbability** in **JabejaTask2.java**) we decided to create our own custom acceptance probability function. We noticed that the provided distribution operates on negative values because we use it when the old value is higher than the new one, we also saw how it starts at on for a difference of zero and it decreases when the difference is higher. We came up with a rational function to differentiate from the original one that was using an exponential, we moved it in (0,1) by adding one to the denominator. We observed how the temperature acted on the function and we noticed that a higher temperature created a lower probability, to reproduce this we multiplied the difference by the probabilistic opposite of the temperature and in the end, we got the following formula:

$$\frac{1}{(old - new)(1 - T) + 1}$$

3 Paralellization of the Algorithm

Since the algorithm is intended to be suitable for large-scale distributed scenarios, we argue that the implementation could be greatly improved by introducing multiple threads (or parallel machines) into the process - rather than iterating through all the nodes in the graph(s) sequentially. This way, we could process most of the swaps in the local environments of all nodes and only communicate with external nodes using the protocol described in the paper when necessary. We decided to implement the customized acceptance probability instead, but chose to include this chapter here since we had a discussion about it.

4 Results

For comparison between tasks, we modified the simulated annealing to re-start at certain points when processing the graph data. We experimented a bit, and found that we got the best results when we set the reset round to be the point when T reached it's lowest value, incremented by 100. For example, for the second task where we started with a T value of 1 with a minimum limit of 0.001 - when we did reach the limit (around round 220), we would re-start the annealing (i.e. reset T back to 1) at round 320. These can be seen as the small bumps in our results graphs, such as in figures 2 and 4.

There didn't seem to be any noticeable difference when re-starting the annealing process more than once or twice - so the resulting graphs presented in this report use the round when the annealing reaches is lowest value multiplied by 2 (for task 2 graphs) and the round incremented by 100 for task 1 graphs.

5 Discussion

The initial implementation was trivial, since it required implementing very little and it was easy to follow along the paper in how it should be done. However, we did have some struggles implementing the second task - for some reason, our edge cuts weren't converging, even though the T value was constantly going downward.

We realized it was because of our cost function being implemented incorrectly, as well as us implementing the acceptance probability in the wrong way - using just T, rather than the formula provided for us. Once we discovered this through some debugging, it was smooth sailing from there on to compare the remainder of the graphs.

References

[1] S. Girdzijauskas M. Jelasity S. Haridi F. Rahimian, A. H. Payberah. *JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning.* IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems, 2013.

[2] Salman Niazi. https://github.com/smkniazi/id2222, 2016.

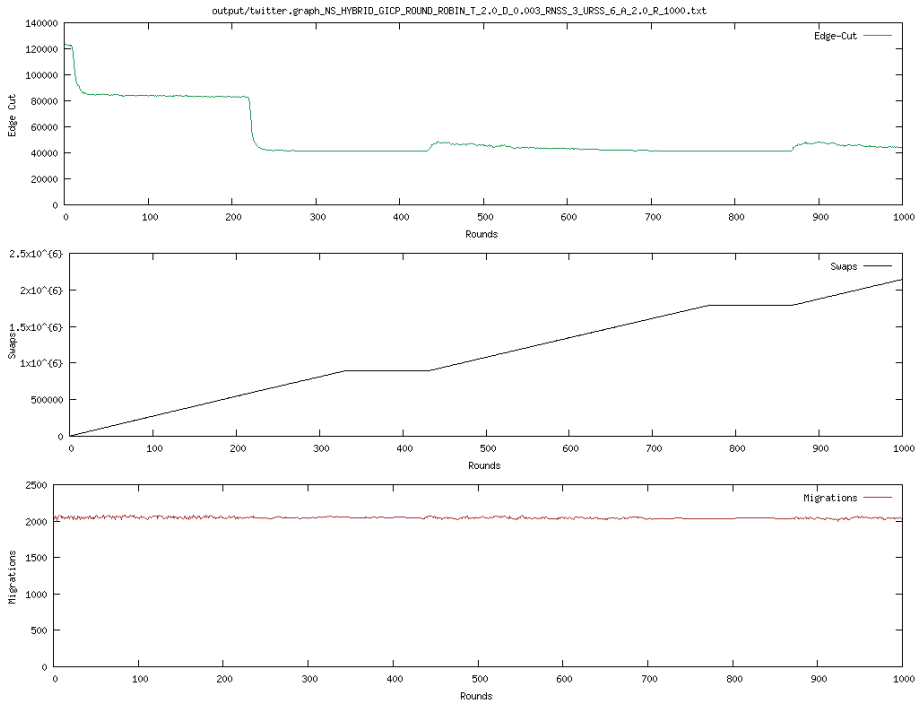

Figure 2: *The graphs as results from running the task 1 implementation of the twitter graph.*

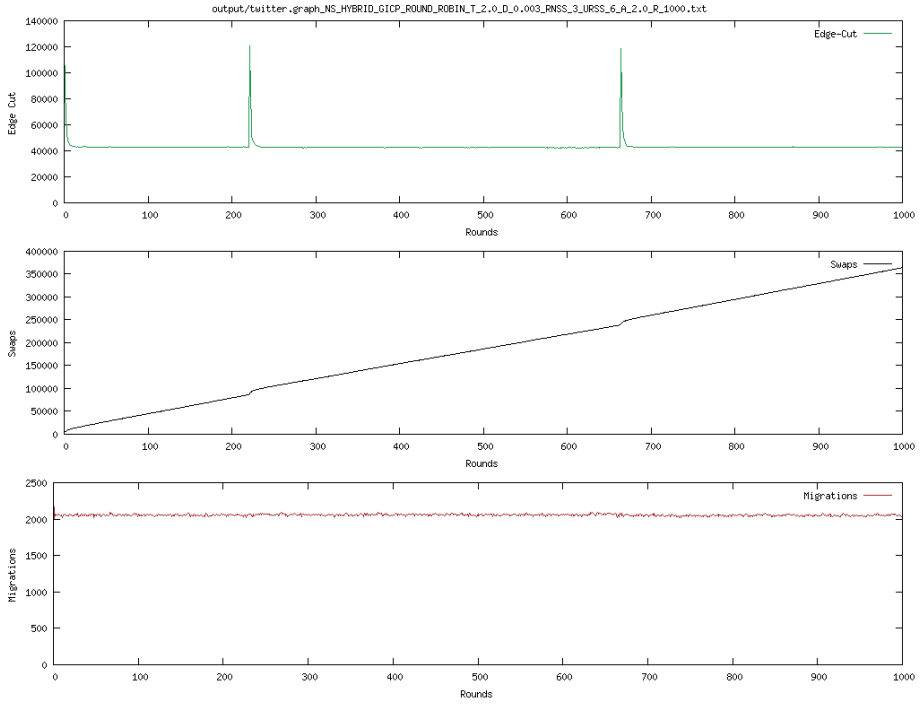

Figure 3: *The figures as results from running the task 2 implementation using our custom acceptance probability function for the twitter graph. Here we can see that the convergence is quite early, even though we simulate the annealing at around rounds 250 and 680.*

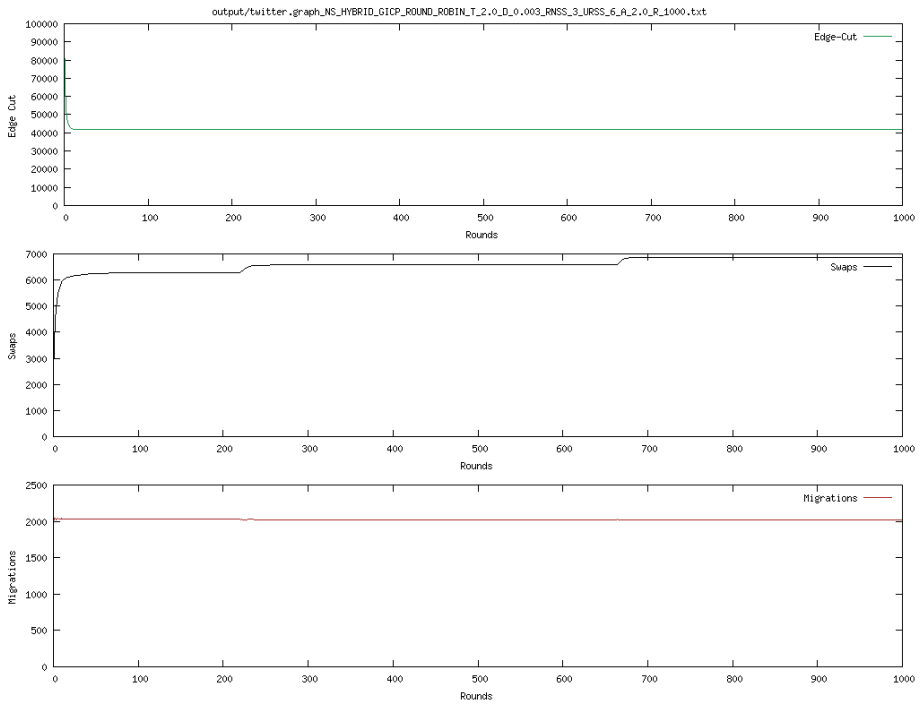

Figure 4: *The figures as results from running the task 2 implementation using the probability function from the provided article for the twitter graph. Compared to our custom probability function, we can see that there are barely any new swaps after the initial convergence in a very early round - even though we reset the annealing process.*