Report 5: Homework Report Template

Georg Zsolnai, Oscar Reina

December 6, 2024

1 Introduction

The main goal of this assignment was to study and implement JA-BE-JA as discussed in the paper titled: "JA-BE-JA : A Distributed Algorithm for Balanced Graph Partitioning". This paper outlines a way to achieve graph partitioning using

For doing this, our assignment divided into two main challenges:

1. Task 1. Implement the Ja-Be-Ja fully into the provided shell code
2. Task 2. Implement the new annealing function from the blog post and test the effects it has. Ensure a fair comparison between improvement and base Ja-Be-Ja.
3. BONUS: Improve the implementation and report findings on different parameters

2 Dataset

The following data set has been used to test the implementation of the assignment, which is found in the current Github Repository: `https://github.com/smkniazi/id2222`. After cloning this repository you can move the /graph folder into the right place. These graphs represent different representation of directed graphs with different properties and different amount of nodes to solve for.

3 Implementation

3.1 Task 1

For the resolution of this task, we implemented and completed the *sampleAndSwap* and *findPartner* methods both marked with TODO tags. For this task, the Ja-Be-Ja algorithm employs a linear annealing method based on

a temperature variable. During execution, this method decreases the temperature linearly in each round. However, this linear trend could lead to premature convergence and a block in a local optima. The results of this initial Ja-Be-Ja implementation can be seen for graphs: **3elt**, **add20** and **twitter** in Figures 1, 3 and 5, respectively.

Time taken for each graph:

- 3elt: start 15:52:11 end 15:53:15 : 1min 4sec
- add20: start 15:53:29 end 15:53:54 : 25sec
- twitter: start 15:54:38 end 15:57:14 : 2min 36min

3.2 Task 2

Based on the linear implementation of the annealing method and the possibility of the results to get stuck in a local optima, we have implemented an annealing simulated mechanism described by [**?**] in order to change the rate of convergence. This is done by providing an exponential cooling of the temperature for every round. Furthermore, in order to avoid getting stuck in a local optima and avoid cooling down "too fast" we can reset the temperature back to the maximum after a specific amount of rounds. Once these amount of rounds have passed we assume that the function has converged and we can reset it. In order to compare the results with the previous section, we have executed this updated algorithm against the same graphs. These can be seen for **3elt add20** and **twitter** in Figures 2, 4 and 6, respectively.

Time with Annealing:

- 3elt: start 16:01:03 end 16:02:06 : 1min 3sec
- add20: start 16:02:22 end 16:02:46 : 24sec
- twitter: start 16:03:06 end 16:05:40 : 2min 34sec

We see slight improvements in the time but nothing to really consider great.

3.2.1 3elt graph

Comparing the results for the execution of tasks without and with annealing, as shown in Figures 1 and 2, it can be seen that the implementation of annealing gets a lower overall number of edge cuts. This shows the improved optimization, as the algorithm lowers the partitioning cost. Furthermore, it can also be seen that the amount of swaps has raised much quicker during the annealing process. This could mean a more aggressive exploration during the early rounds, where the system allows more exploration to avoid local minima. This exploration then plateaus around a consistent value. Finally, regarding migrations, there is not that much difference between the two

executions, since both converge to similar values and stabilize after a similar number of rounds. This could hint us that the annealing implementation has small impact on node migration but achieves much better in overall edge cuts.

3.2.2 20add graph

Comparing the results for the execution of tasks without and with annealing, as shown in Figures 3 and 4, it can be seen that there are significant improvements when using annealing. For the edge cuts, the graph with annealing has a much sharper initial decrease, and stabilizes at a much lower edge cuts value than the one without the annealing implementation. This shows us that the new annealing implementation lowers the partitioning cost by exploring more early in the process. Furthermore, regarding the swaps, the execution without annealing has a smoother trend and bigger amount of swaps, which shows that it makes smaller and more gradual changes during its optimization process showing a less structured exploration and frequent adjustments around local optima. On the other hand, surprisingly, the implementation with annealing results a lower number of swaps early on, which could be due to a more systematic way to explore and stabilise. For migrations, the non-annealing implementation can be seen to have a faster initial increase followed by a smoother stabilization, whereas during the annealing implementation we have a more bumpy trend due to the resetting of the temperature. This demonstrates that annealing helps to reduce the edge cuts and control the color changes also.

3.2.3 Twitter graph

Comparing the results for the execution of tasks without and with annealing, as shown in Figures 5 and 6, it can be seen that there are similar trends as those in the other graphs. With annealing, the number of edge cuts decreases much faster and earlier than without, getting very similar results in fewer iterations. Furthermore, according to the swaps, the annealing process shows a faster increase and does not plateauing as early as the implementation without annealing, though with a significant decrease in swaps. This happens because of the periodic resetting of the temperature, which makes further exploration and prevents premature convergence. Finally, migrations show a similar trend overall, reflecting the behavior seen in previous graphs.

3.2.4 Conclusions

The table 1 below shows the conclusions we made from Task 1 and Task 2, outlining the central changes from the initial annealing function and the improved version.

Aspect	Annealing (annealing = true)	No Annealing (annealing = false)
Swap Acceptance	Is based on *probabilistic acceptance* using T and acceptance_probability. Early swaps are frequent; later swaps are rare.	Swaps occur whenever the new value improves the solution; no probabilistic control.
Exploration vs. Stability	Balances *exploration* (early) and *stability* (late) using the cooling schedule.	Focuses entirely on *improvements*, leading to aggressive swapping.
Total Number of Swaps	Fewer swaps overall, as the algorithm stabilizes in later rounds.	More swaps overall, as marginal improvements are always accepted.
Behavior Over Rounds	Swapping slows down as T approaches MIN_T.	Consistent swapping behavior across all rounds.

Table 1: Comparison of behavior with and without annealing in the algorithm.

3.3 Bonus - Improvement

To improve the algorithm marginally we re-thought of way in which we can improve the proposed acceptance probability to perhaps cool down more aggressively as time progresses. Hence we took the approach of modifying the equation to look as follows:

$$e^{\frac{(c_{new}-c_{old})}{T^{annealing\_rounds}}}$$

The only change was to use exponentiation in the denominator of the function and put the annealing_round in that place. This increases the denominator exponentially, drastically cooling down the temperature quicker than before. The annealing_rounds were increased every time we complete a round and call the saCoolDown() function and is reset similarly when we reach the MAX_ROUNDS value.

Time with Bonus:

- 3elt: start 16:06:33 end 16:07:38 : 1min 4sec
- add20: start 16:08:19 end 16:08:44 : 25sec
- twitter: start 16:09:07 end 16:11:37 : 2min 30sec

We see the greatest change here and notice that we are a bit quicker overall.

The following were print statements for the OLD annealing from Task 2 and the improved version implemented for the bonus titled NEW.

- OLD: Jabeja: 290 - round: 999, edge cut:1784, swaps: 38574, migrations: 3328
- NEW: Jabeja: 290 - round: 999, edge cut:1636, swaps: 40013, migrations: 3333

4 Setup and execution

To build and run this project, please follow the following steps:

1. Ensure that you have Java and Maven installed.
2. Download the data set provided in Section 2 and place it within the */graphs* directory that you have to create in the root directory of the repository.
3. Make sure to create a folder titled "/output" in the same root folder as where the */graphs* folder is as this is where the images will be created
4. Once the dataset is downloaded and dependencies are installed, simply execute the following command:

```
./compile.sh
./run -graph ./graphs/*your chosen graph*
./plot .sh output/result
```

5 Comparison of Graphs with and without Annealing

Figure 1: 3elt graph without annealing

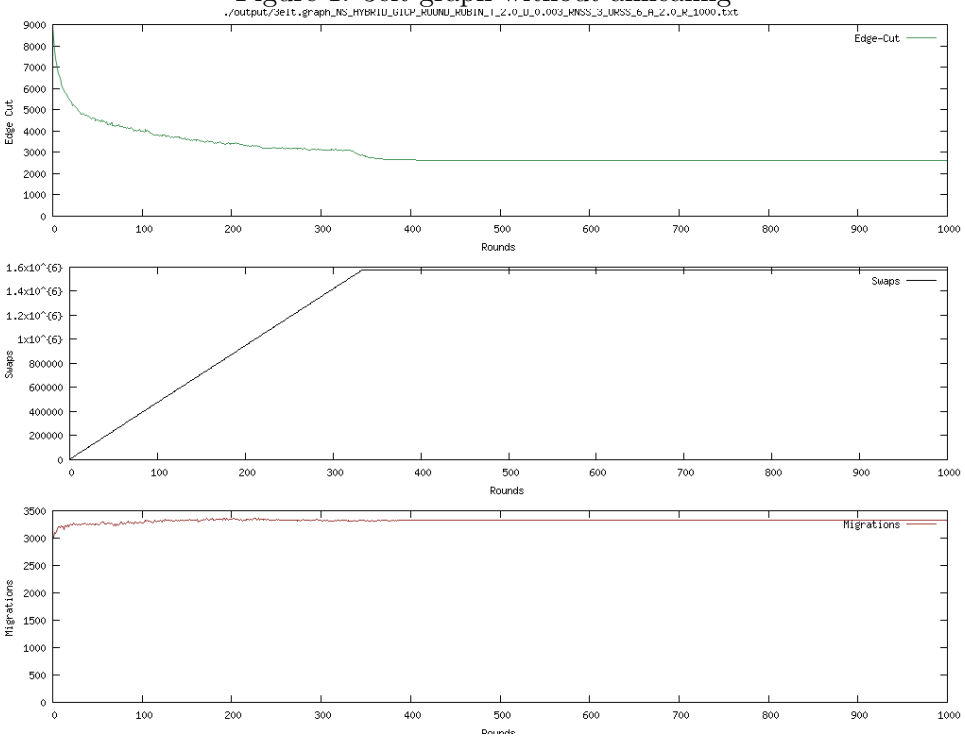

Figure 2: 3elt graph with annealing

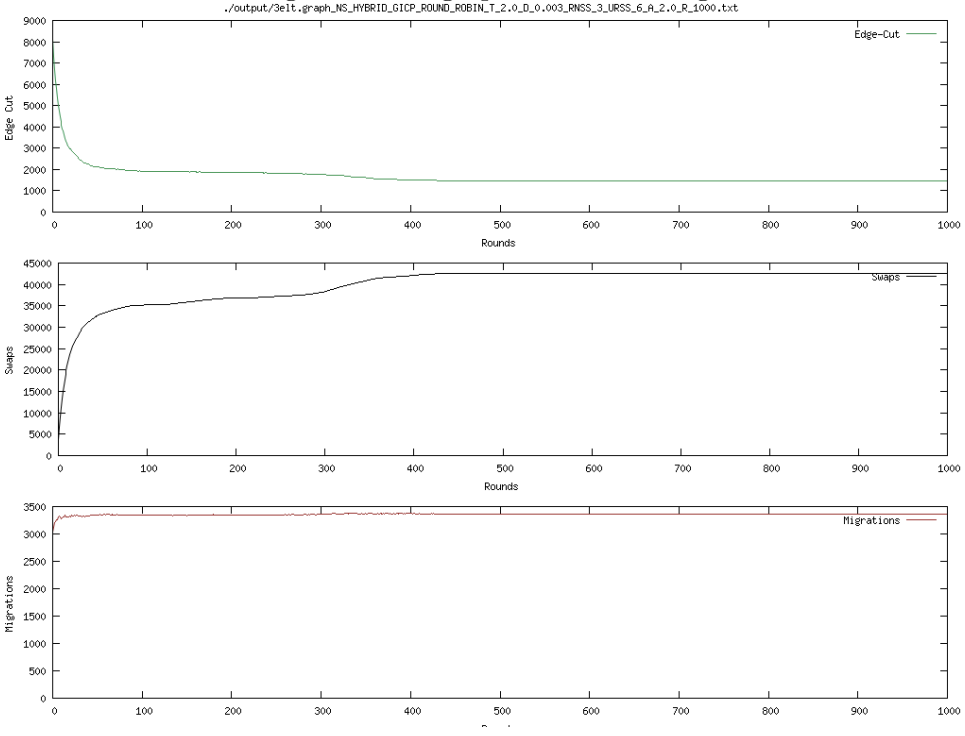

Figure 3: add20 graph without annealing

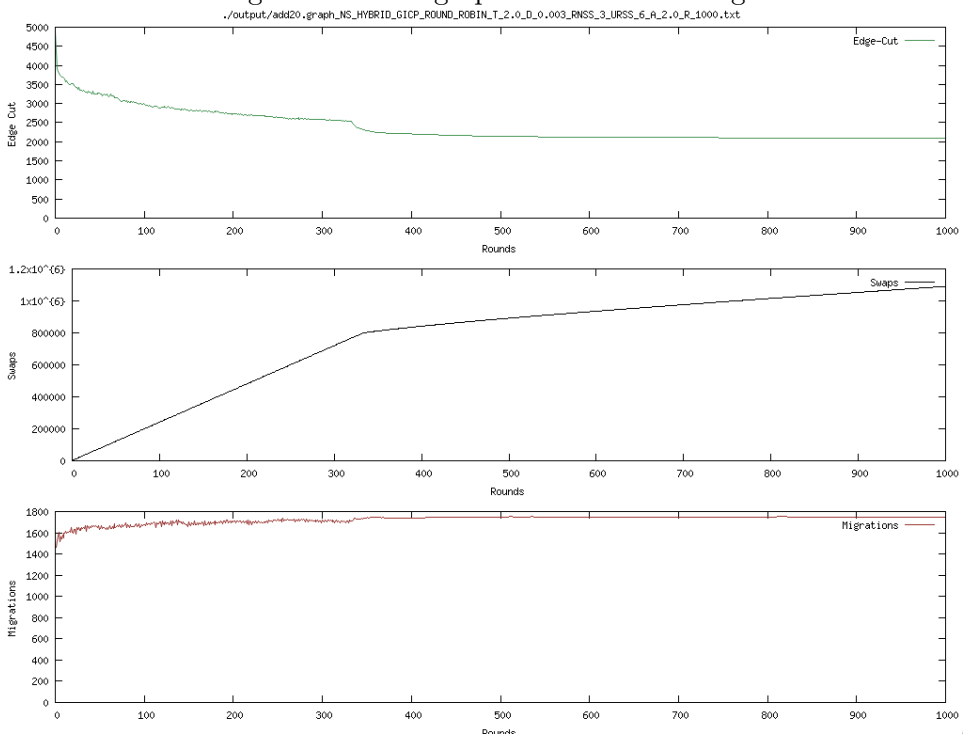

Figure 4: add20 graph with annealing

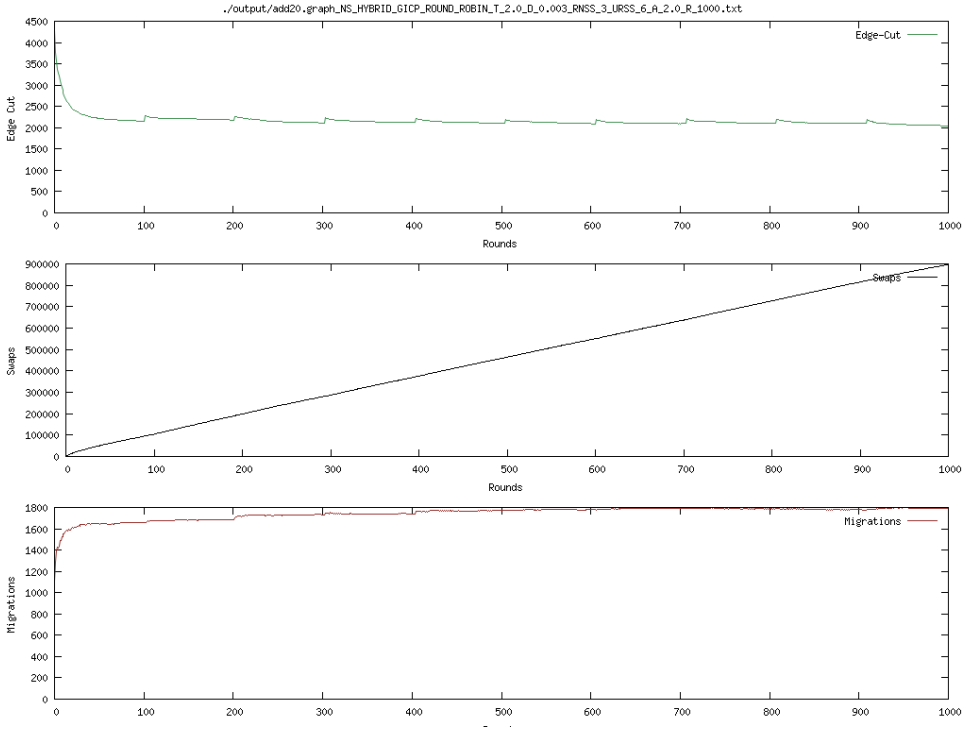

Figure 5: Twitter graph without annealing

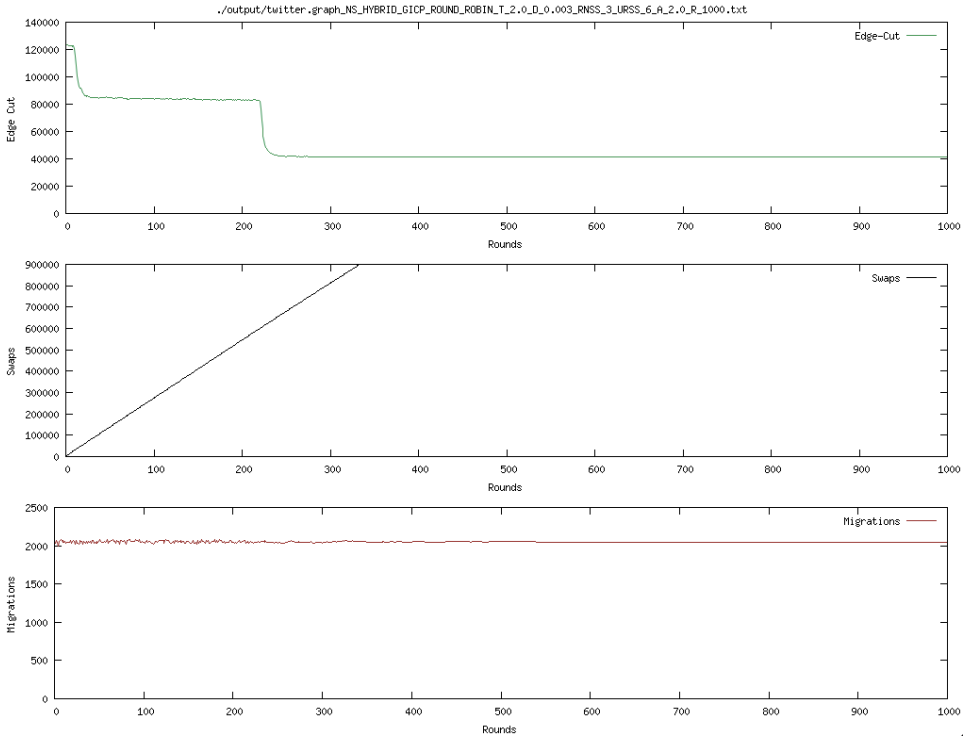

Figure 6: Twitter graph with annealing