ID2222 Data Mining

Homework 5: K-way Graph Partitioning Using JaBeJa

Noel Karlsson Johansson
noelkj@kth.se

Alexander Olsson
alexao@kth.se

December 10, 2023

1 Introduction

The JaBeJa algorithm is a distributed, self-stabilizing algorithm designed for balanced graph partitioning, crucial in parallel computing and data mining. It addresses the challenge of dividing a graph into subgraphs while minimizing the number of edges between them, known as the edge-cut. This is essential for efficient parallel processing and data management. The algorithm employs a local search strategy where nodes are swapped to reduce the edge-cut iteratively. Nodes assess their local neighborhood and swap positions with other nodes to improve the overall partitioning. This process continues until a satisfactory balance and minimal edge-cut are achieved. By optimizing the distribution of nodes across partitions, JaBeJa enhances the performance of parallel computations and facilitates effective data analysis.

Explanation of relevant terms

- **Graph**: A structure representing relationships between objects, where nodes represent entities like social media users, and edges represent relationships or interactions between them.
- **Edge-cut**: The number of edges that cross the boundaries of partitions in a graph. Minimizing the edge-cut is a key objective in graph partitioning, as it represents the inter-partition communication cost. We aim to minimize the number of edge-cuts to improve efficiency.
- **Swaps**: Refers to the number of times nodes exchange their partitions (colors) in an attempt to reduce the edge-cut. The JaBeJa algorithm uses local search and swaps between nodes to optimize the partitioning. Minimizing the number of swaps leads to a more efficient algorithm, indicating that the algorithm can find a near-optimal partitioning with fewer iterations, which saves computational resources.
- **Migrations**: The number of nodes that need to be moved from their initial partition to their final partition after the algorithm has converged. Minimizing migrations means that less data needs to be moved across partitions or physical machines after the algorithm has converged, reducing the costs associated with data transfer.
- **Delta**: The cooling schedule parameter in the simulated annealing process used by the algorithm. It determines the speed at which the temperature is reduced during the optimization process. A higher delta value leads to a faster cooling schedule, affecting the number of swaps and the quality of the partitioning. The goal is to adjust delta to effectively balance the algorithm's exploration and exploitation phases.

2 Task 1

2.1 Implemented JaBeJa Algorithm

To implement the JaBeJa algorithm, modifications were made to the JaBeJa.java class, particularly sampleAndSwap(...) and findPartner(...). Link to assignment source code: GitHub.

Graph	Delta	Edge-cut	Swaps	Migrations
add20	0.003	2 095	1 090 263	1 751
3elt	0.003	2 604	1 580 209	3 328
Twitter	0.003	41 156	899 515	2 049

Table 1: Results with basic JaBeJa implementation

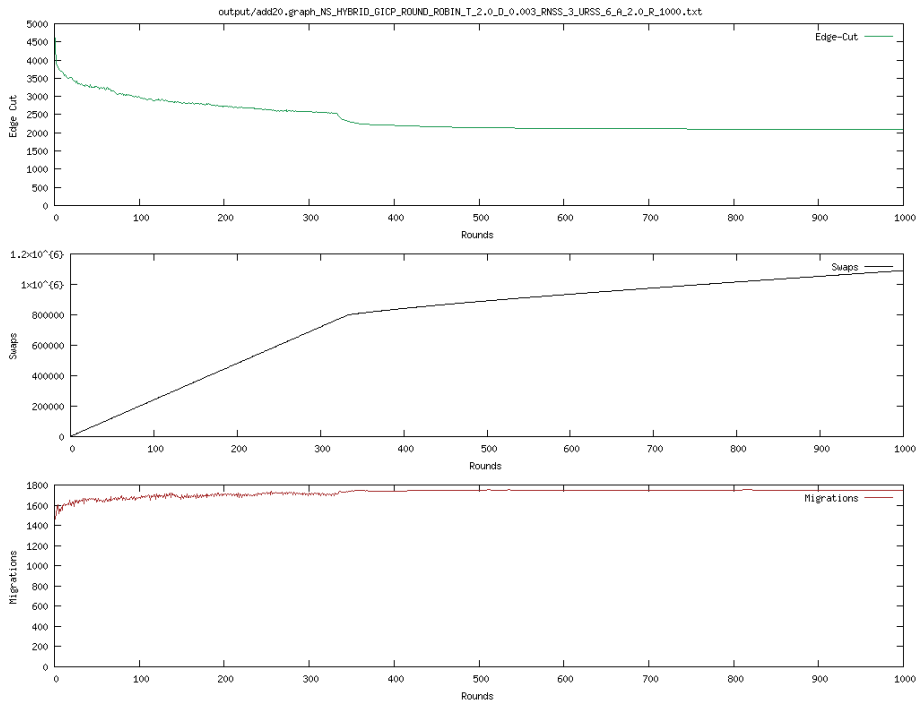

Figure 1: add20

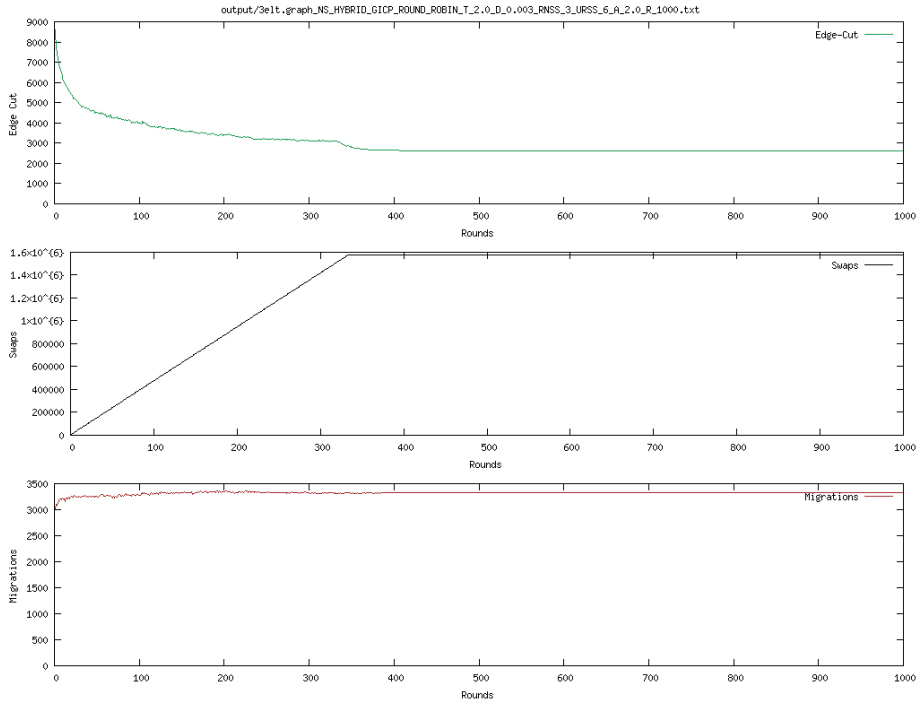

Figure 2: 3elt

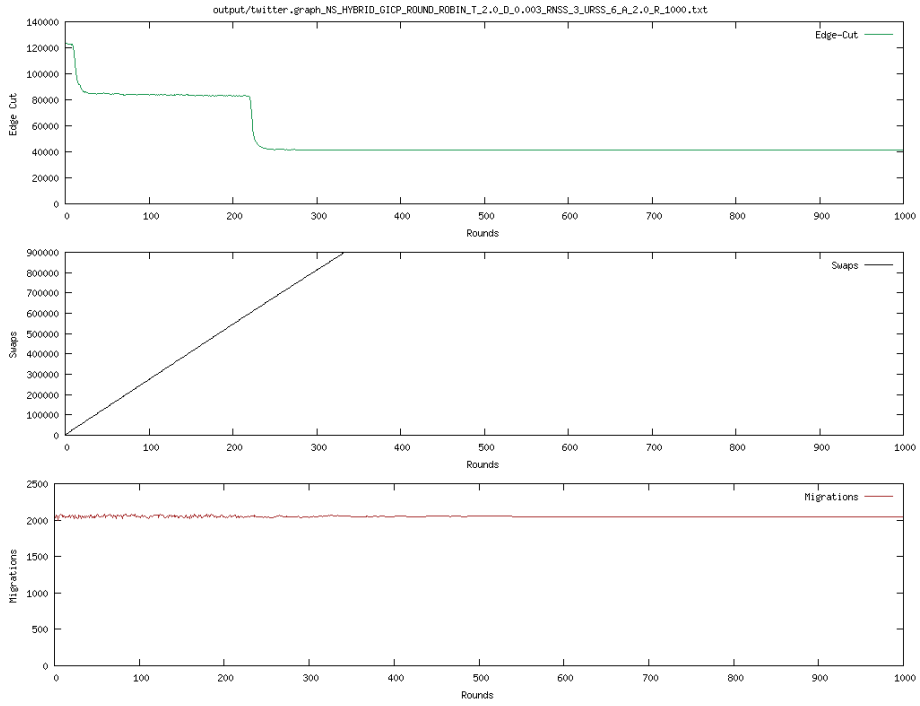

Figure 3: Twitter

3 Task 2

3.1 Adjusting JaBeJa Configurations

Task 2 delved into refining JaBeJa configurations to find minimal edge cuts in given graphs. The focus was on understanding how changes in simulated annealing algorithm affected performance. A different annealing mechanism was implemented, and parameters were tweaked to observe convergence rates.

The comparison between Table 2 and Table 1 highlights improved statistics across nearly every metric with the adoption of the new annealing mechanism.

Graph	Delta	Edge-cut	Swaps	Migrations
add20	0.90	1 933	582 758	1 710
3elt	0.90	1 824	31 114	3 346
Twitter	0.90	41 250	6 036	2 017

Table 2: Results with simulated annealing

Particularly noteworthy is the substantial reduction in the number of swaps for all graphs. As a result, switching from a linear to an exponential reduction in temperature boosted both efficiency and performance.

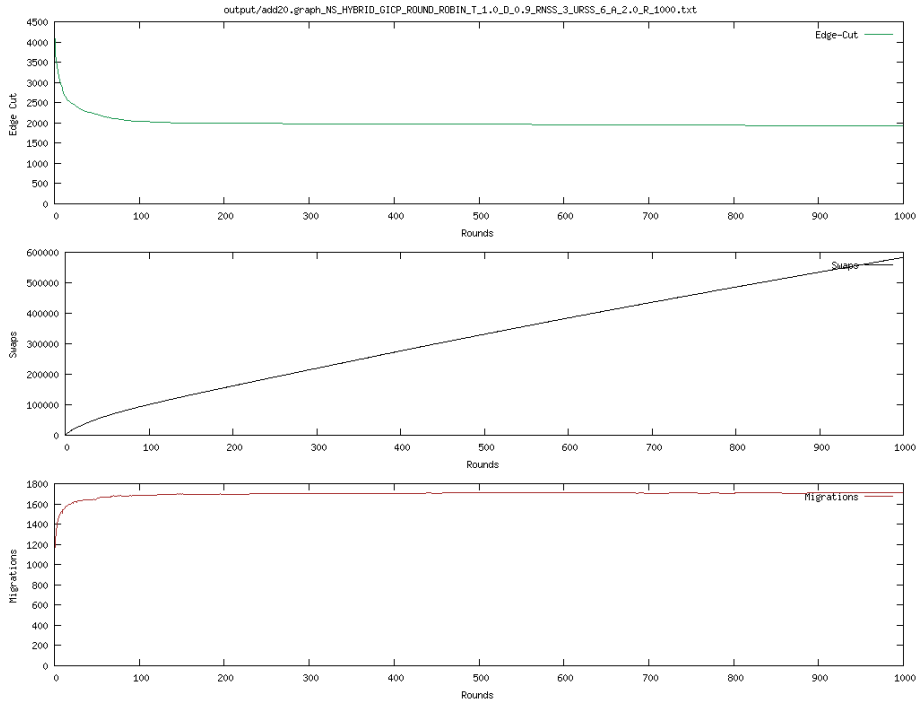

Figure 4: add20

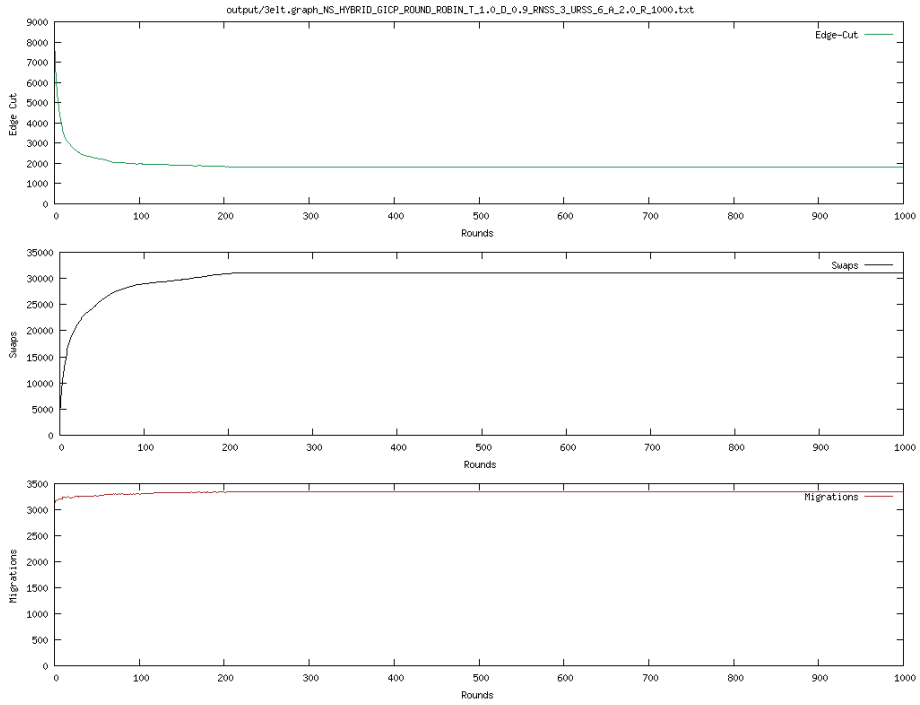

Figure 5: 3elt

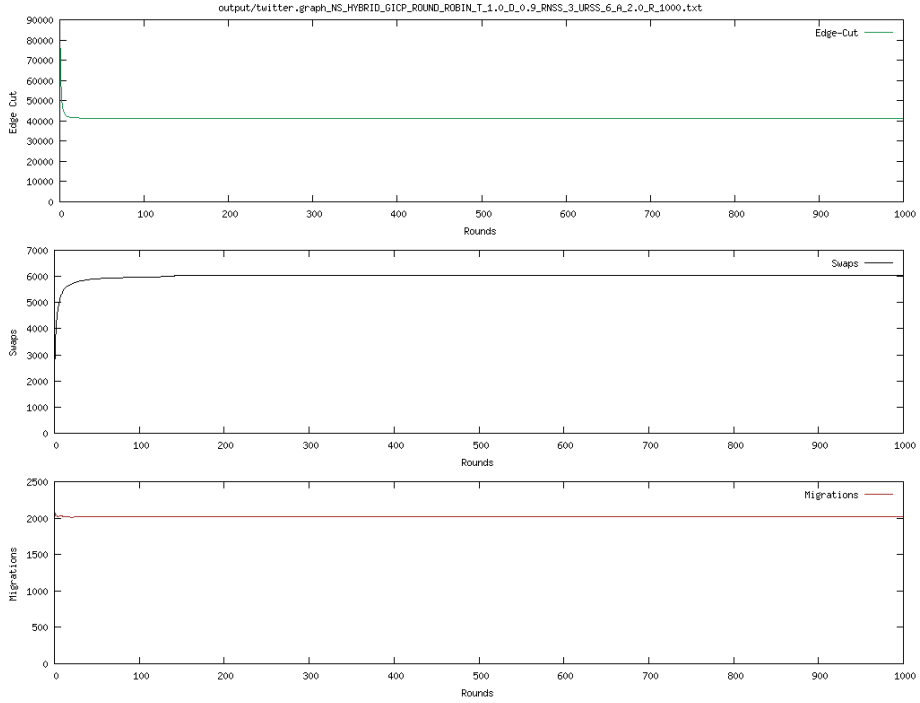

Figure 6: Twitter

3.2 Simulated Annealing Analysis

In this phase, we explore restarting the simulated annealing in hopes for even lower edge-cuts.

Graph	Delta	Edge-cut	Swaps	Migrations
add20	0.90	2 163	837 611	1 746
3elt	0.90	1 562	33 925	3 337
Twitter	0.90	41 294	6 504	2 024

Table 3: Results with simulated annealing, restarted at every 400 rounds

Examining Table 3, it becomes evident that restarting the temperature every 400 rounds yields positive effects for the 3elt dataset, negative effects for add20, and neutral outcomes for the Twitter dataset. This observation implies that, for the add20 dataset, the temperature reset might not be sufficient to escape a local minimum or suggests that we are potentially already situated at a global minimum.

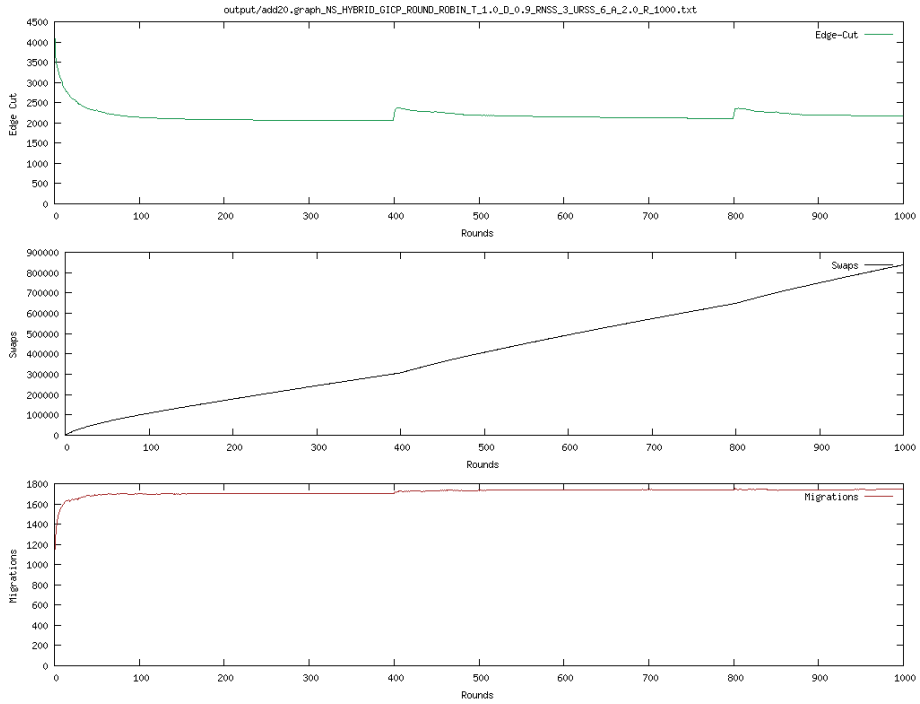

Figure 7: add20

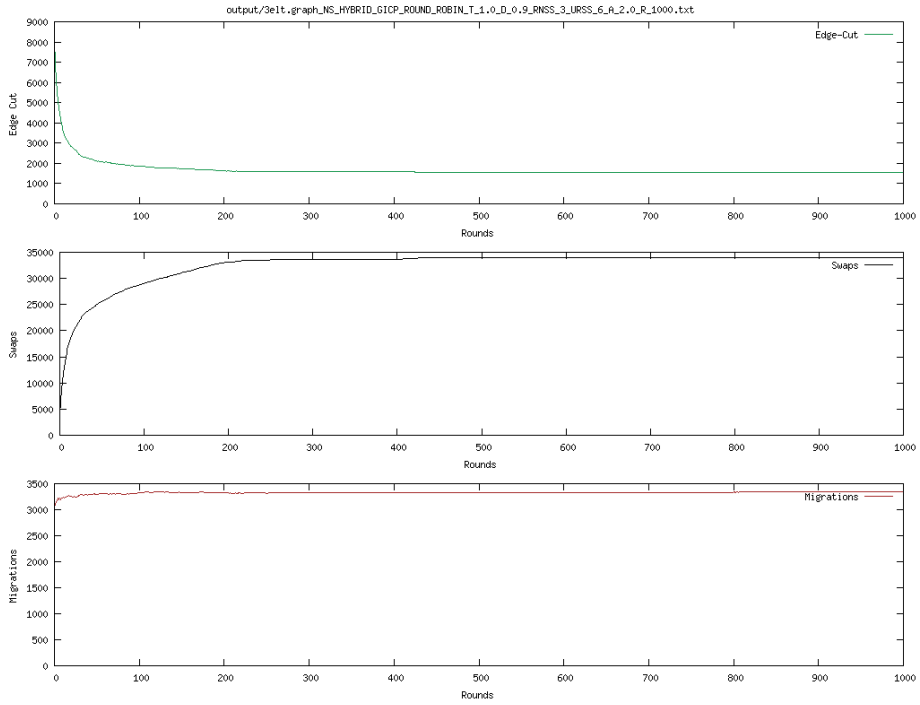

Figure 8: 3elt

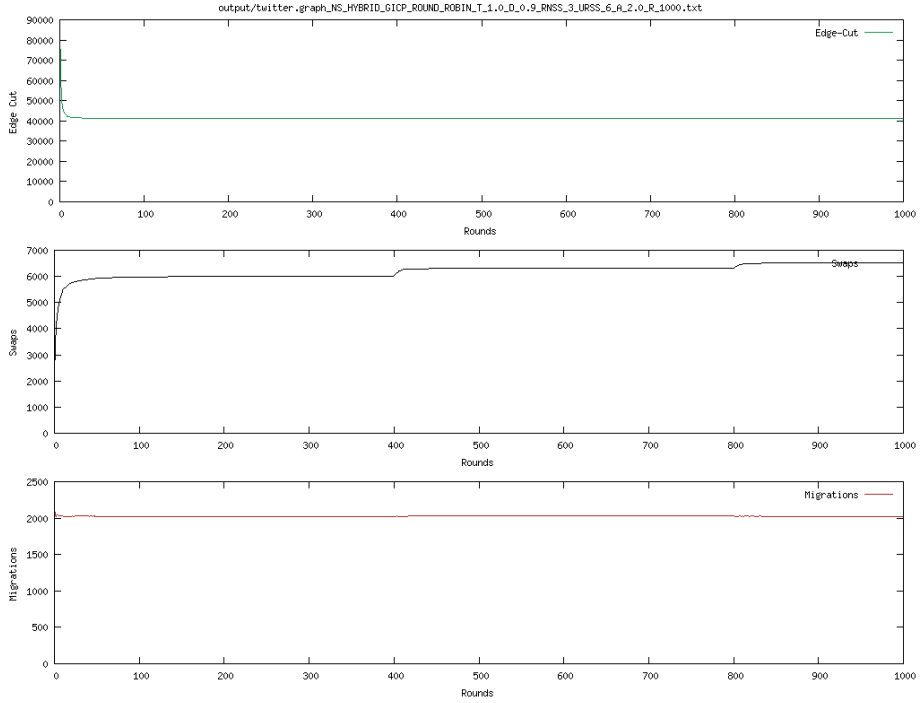

Figure 9: Twitter

4 Bonus

This bonus implements two changes, the first is a varying temperature reduction function

```
private float customTemperatureReduction(int round) {
    // For bonus task
    if (bonusTaskVer1) {
      return T *= 1000 / (1000 + round/3);
    }
  }
```

The other is a new acceptance probability

```
private double getAcceptanceProbability(double oldDegree, double
    ↪ newDegree) {
    // The acceptance probability function receives the previous
    ↪ cost, the new cost,
    // and the present temperature, then outputs a value ranging
    ↪ from 0 to 1.
    // This value serves as an advisory on the advisability of
    ↪ transitioning to the new solution.

    if (bonusTaskVer2) {
      return Math.exp((1 / oldDegree - 1 / newDegree) / T);
    }
  }
```

4.1 Custom Temperature Reduction

Graph	Delta	Edge-cut	Swaps	Migrations
add20	0.90	2 138	737 161	1 734
3elt	0.90	1 574	37 653	3 318
Twitter	0.90	41 261	5 950	2 033

Table 4: Results with only custom temperature reduction function

In the comparison of results between the Bonus task (Table 4) and Task 2.2 (Table 3), a notable trend emerges in the Swaps metric, reflecting the number of bad swaps made during the simulated annealing process. For the add20 and twitter datasets, we observe a slight reduction in the number of Swaps in the Bonus task, where the custom temperature reduction function is employed. Specifically, in the add20 dataset, the Swaps decrease from 837 611 to 737 161, and in the twitter dataset, from 6 504 to 5 950. This reduction signifies improved efficiency in reaching optimized solutions. However, the 3elt dataset appears to have a higher number of Swaps in the bonus task, suggesting that the custom temperature reduction function may have a more pronounced impact on certain datasets, while others, like 3elt, remain relatively unaffected.

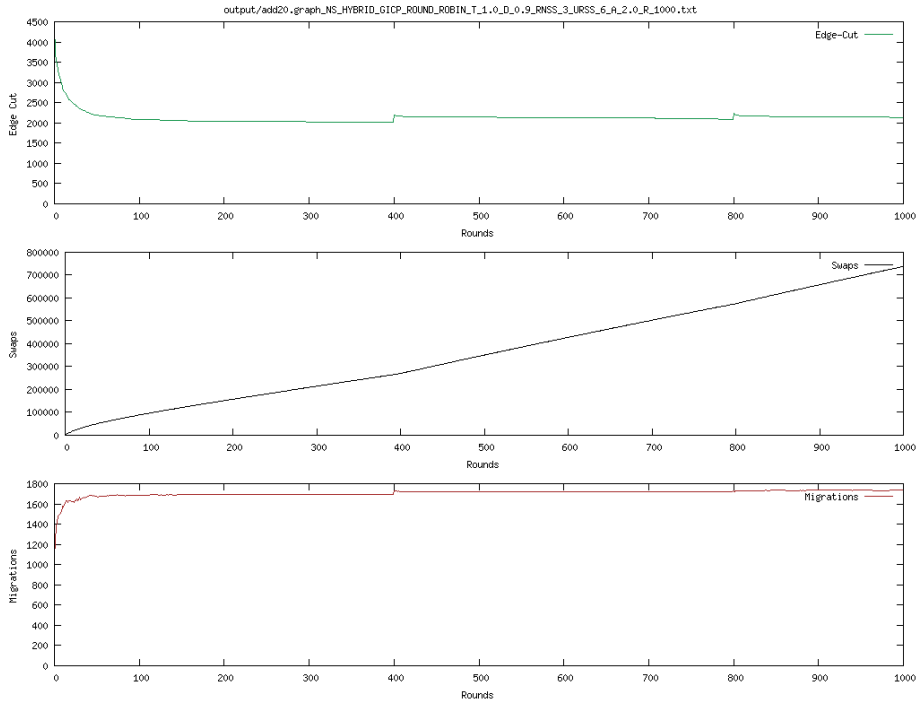

Figure 10: add20

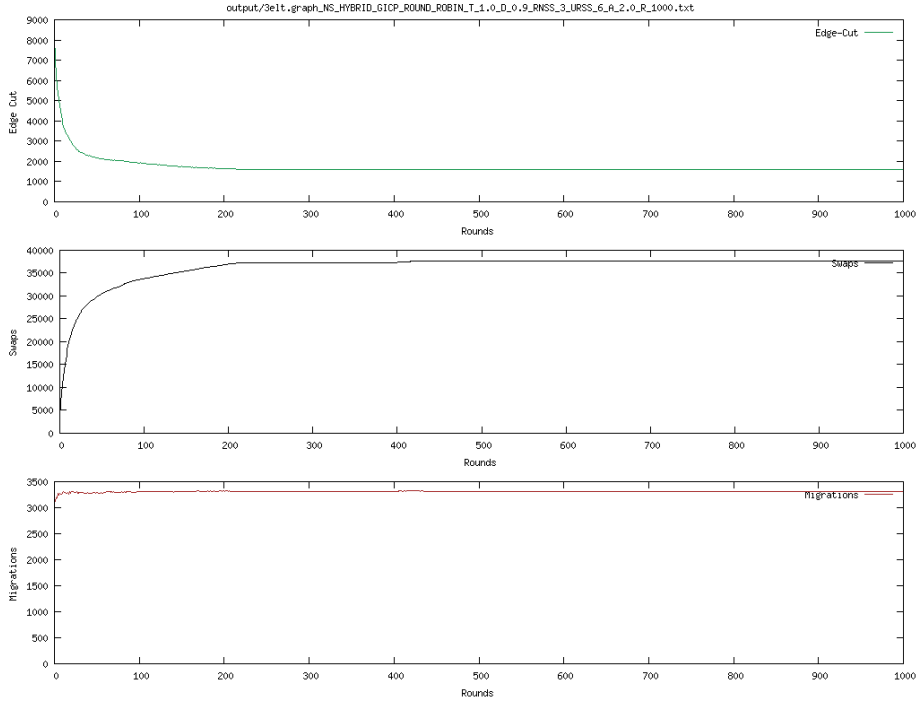

Figure 11: 3elt

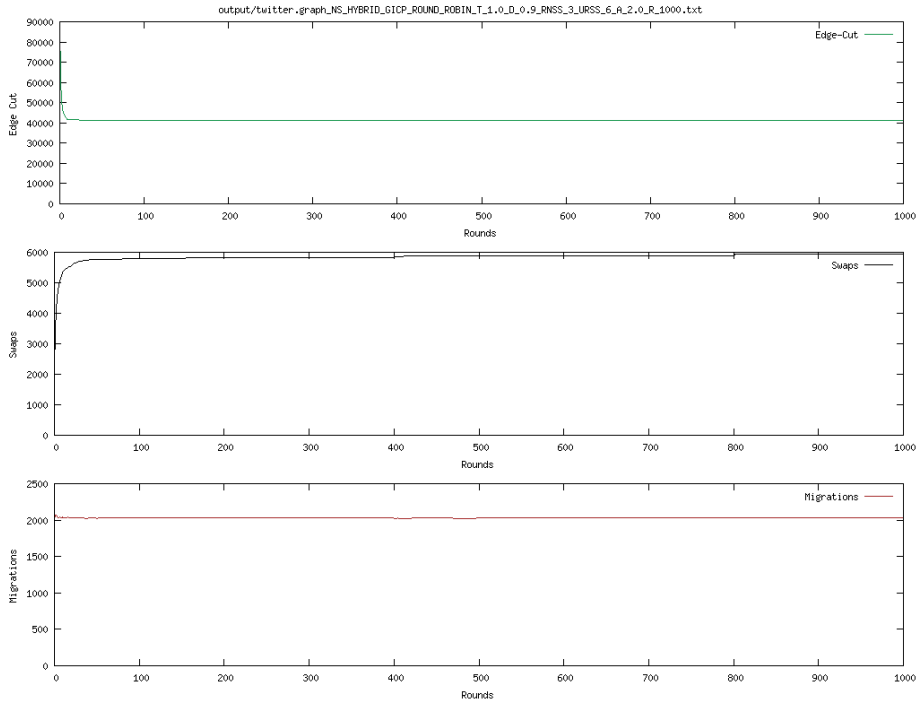

Figure 12: Twitter

4.2 Custom Acceptance Probability

Graph	Delta	Edge-cut	Swaps	Migrations
add20	0.90	2 272	758 875	1 832
3elt	0.90	1 188	112 682	3 517
Twitter	0.90	48 542	317 674	2 054

Table 5: Results with custom temperature reduction function and custom acceptance probability function

Table 5 presents the outcomes of the JaBeJa algorithm when employing both a custom temperature reduction function and a custom acceptance probability function. Notably, for the 3elt graph, the implementation of these custom functions resulted in a significant reduction in edge-cuts, dropping to 1 188 from 1 562, as observed in Table 3. At the same time, the improved performance involves a higher computation cost, which can be seen in the larger number swaps compared to the results in Table 3.

For the add20 and Twitter graphs, while there are notable changes in edge-cuts, swaps, and migrations, the impact is less pronounced compared to the 3elt graph. This shows that the degree of improvement varies based on the inherent structure and properties of the graphs being analyzed.

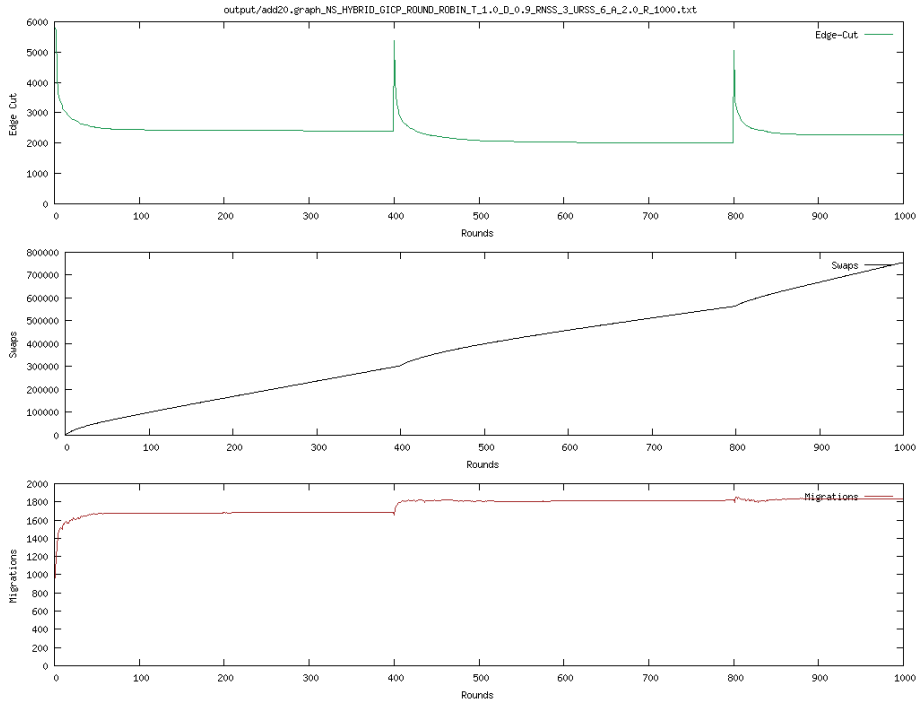

Figure 13: add20

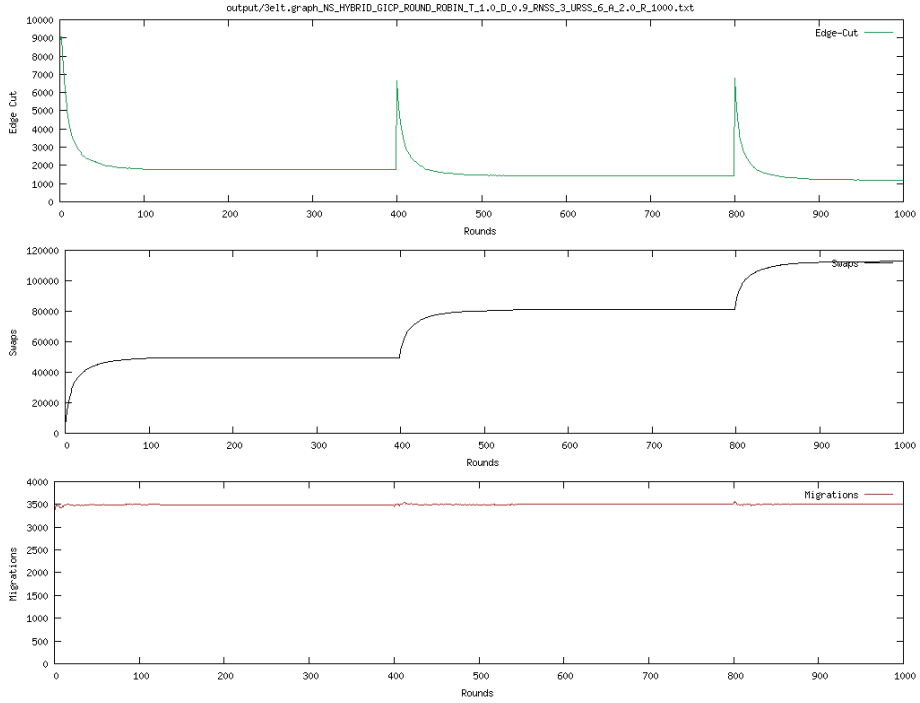

Figure 14: 3elt

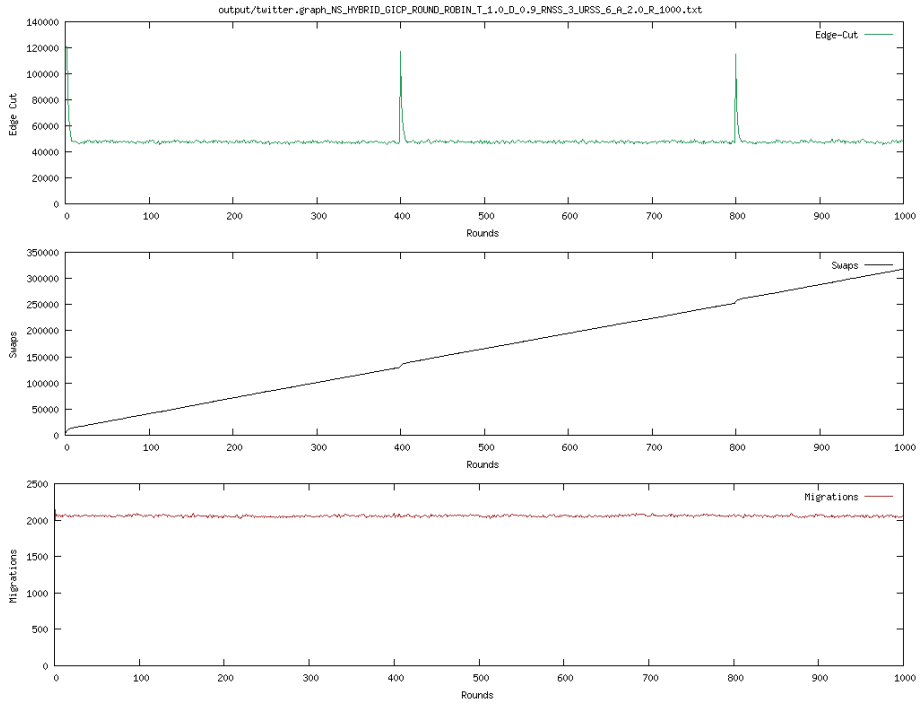

Figure 15: Twitter

5 How to run code

To run the Java code on a Windows machine, begin with installing the following packages:

- Maven
- gnuplot
- Git Bash

After installing the packages, extract the folder from the handed-in zip-file. Then, open Git Bash and do the following:

1. Choose directory `cd <folder-path>`
2. Execute `./compile.sh`
3. Execute `./run.sh -graph ./graphs/<graph-name>.graph`
4. Execute `./plot.sh output/<result-name>.txt`
5. See text file with results in the output folder, and saved graph image in the root folder.

To run the code with different set-ups, enable or disable functionalities at row 26–29 in the `Jabeja.java` file.

6 Literature

Readings F. Rahimian, A. H. Payberah, S. Girdzijauskas, M. Jelasity, and S. Haridi, *JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning*, SASO2013, pp. 51-60.

F Rahimian, AH Payberah, S Girdzijauskas, M Jelasity, S Haridi, *A distributed algorithm for large-scale graph partitioning*, ACM Transactions on Autonomous and Adaptive Systems (TAAS) 10 (2), 12.

K. E. Grolinger, February 2014, *The Simulated Annealing Algorithm* , accessed on December 10, 2023.