K-way Graph Partitioning Using Ja-Be-Ja

Eva Engel
Tori Leatherman

December 12th 2022

1 Introduction

As detailed in the assignment, we have implemented the Ja-Be-Ja algorithm from the paper by Rahiminian, et al. The assignment consists of two tasks which we will describe in more detail below, both containing variations of the Ja-Be-Ja algorithm as a method of peer-to-peer graph partitioning. For the assignment, we will analyze 3 provided datasets: 3elt, add20, and facebook.

2 Ja-Be-Ja

Algorithm 1 Edge/Vertex Cut

```
procedure cut(graph, policy)
    // policy identifies the partitioning algorithm, i.e., edge-cut or vertex-cut;
    bestPartner ← getBestPartner(self.getNeighbours(), policy, T_r);
    if (bestPartner = null) then
        bestPartner ← getBestPartner(graph.getRandomVertices(), policy, T_r);
    if (bestPartner ≠ null) then
        if (policy = EdgeCut) then
            swapVertexColor(self, bestPartner);
        else
            swapEdgeColor(self, bestPartner);
    T_r ← T_r − δ;
    if (T_r < 1) then
        T_r ← 1;
```

Figure 1: Ja-Be-Ja algorithm

Figure 1 presents the core of Ja-Be-Ja algorithm that runs periodically by all vertices of a graph. As it shows, they use the hybrid heuristic for vertex selection, which first tries the local policy, and if it fails it follows the random policy.

3 Task 1

Using the provided framework of the Ja-Be-Ja algorithm, we were tasked with modifying the following three methods contained in the Ja-Be-Ja class: *saCoolDown()*, *sampleAndSwap()*, and *findPartner()*. Referencing the psuedo-code contained in the original paper we were able to implement the Ja-Be-Ja's algorithm using the Hybrid model. This refers to the way the partners of a node are chosen. In the Hybrid case, the algorithm first selects local neighbors of the node. If this attempt fails, the algorithm will select a random sample of nodes for partners.

For Task 1, we keep the default values for the temperature T and alpha a parameters. We implement the standard annealing procedure contained in the document, such that there is a linear acceptance probability function. After doing so we get the following results for the three graphs:

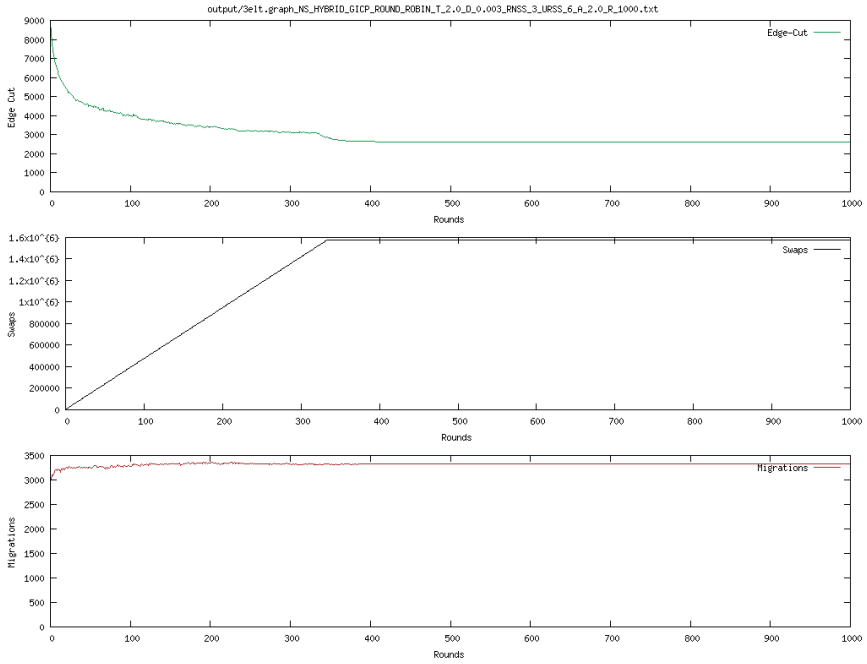

Figure 2: Task 1 – 3elt graph with linear annealer

The corresponding values for the number of swaps and the minimum edge cut observed can be found in the below table:

Graph	Minimum Edge Cut	#Swaps
3elt	2604	1580209
add20	2095	1090263
facebook	134246	21200364

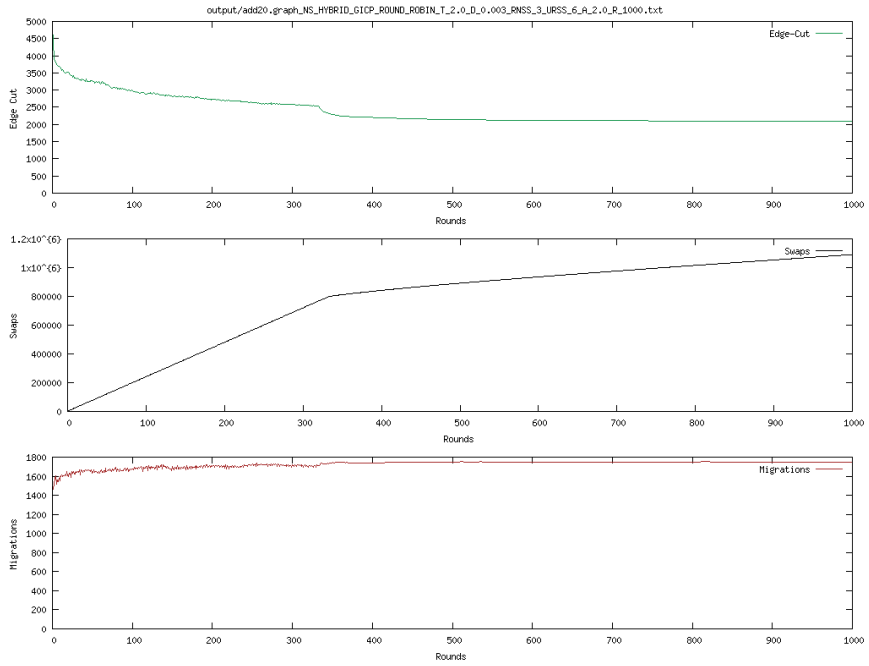

Figure 3: Task 1 – add20 graph with linear annealer

4 Task 2

For the second task, we implement an exponential annealer that obeys the following logic. The temperature T is now updated each iteration by multiplying by some δ, which leads to exponential decay until it reaches a minimum. With each iteration the Ja-Be-Ja algorithm functions as follows:

- If the new solution has lower cost then the old solution, the new solution will be accepted.
- If the new solution is not lower than the old solution, we calculate the acceptance probability and accept the new solution if this probability is larger than some random number. The acceptance probability is calculated as follows:

$$p_{acc} = e^{\frac{c_{new} - cold}{T}} \tag{1}$$

  where T is the updated temperature.

With this exponential annealing, we get the following figures for each of the 3 graphs:
The following table summarizes the results for each graph after round 999:

Graph	Minimum Edge Cut	#Swaps
3elt	10361	4719786
add20	5860	2394923
facebook	616154	63728451

As you can see from the above figures, we do not reach convergence with this configuration. As mentioned in the article, the acceptance probability measures whether or not the algorithm should jump to a new solution or not, generally the closer to 1.0, the more likely the new solution is better. Thus if the random number generated is below 0.5, there is a large chance that the algorithm will accept worse solutions and will not converge.

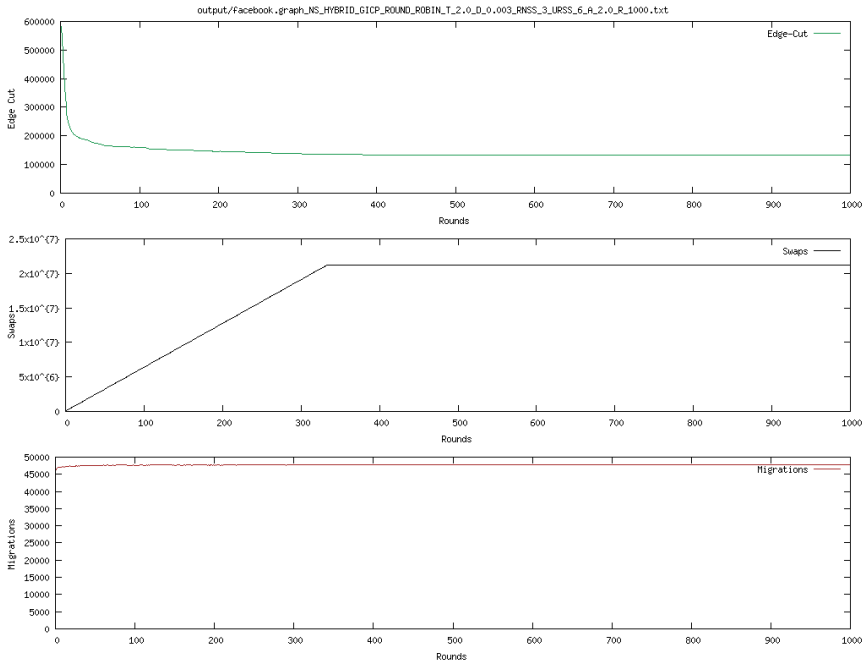

Figure 4: Task 1– facebook graph with linear annealer

Additionally, for the second task we add restarts. Thus after 400 rounds, we reset the the temperature T to the initial value. This allows the algorithm to find the global minima for the edge cut.

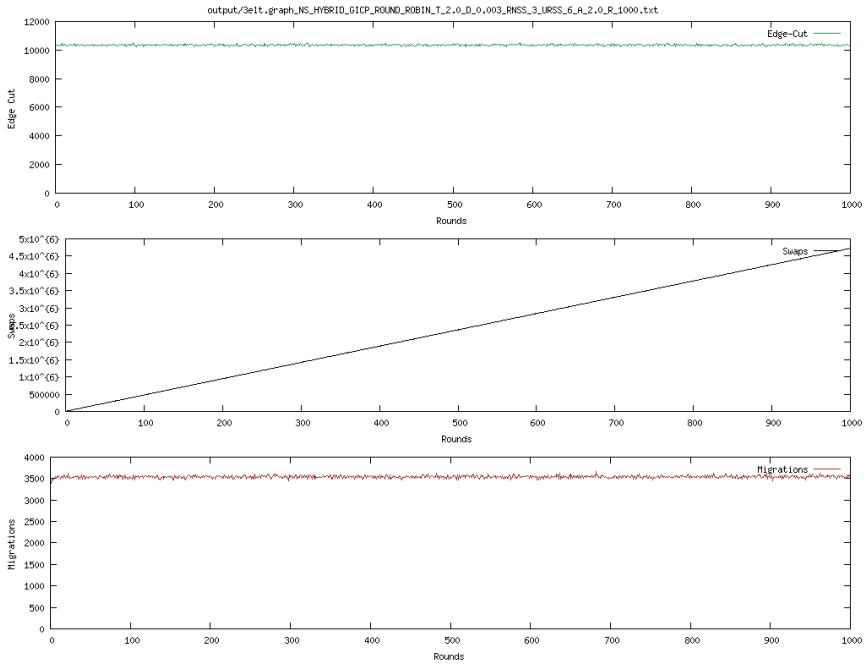

Figure 5: Task 2 – 3elt graph with exponential annealer

5 Bonus Task

In order to find an even better configuration, we impose another constraint to the exponential annealing algorithm. We now require that the acceptance probability is greater than 0.5. As stated earlier, the closer the acceptance probability is to 1, the better the new value. Thus imposing this constraint, gave us the fastest convergence of all the methods. The results from these methods can be seen in the below figures and table.

Graph	Minimum Edge Cut	#Swaps
3elt	1668	49447
add20	2311	1299450
facebook	137137	5725646

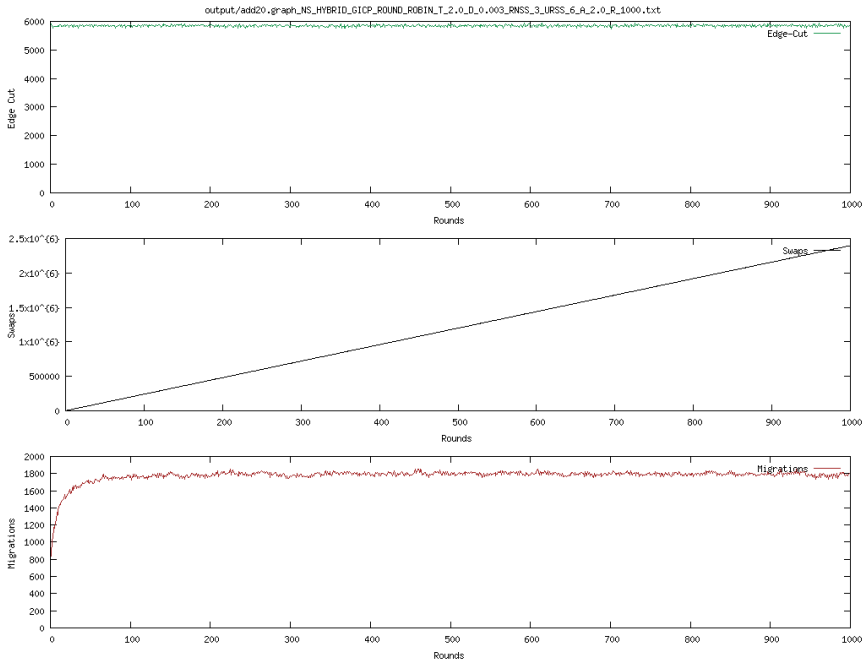

Figure 6: Task 2 – add20 graph with exponential annealer

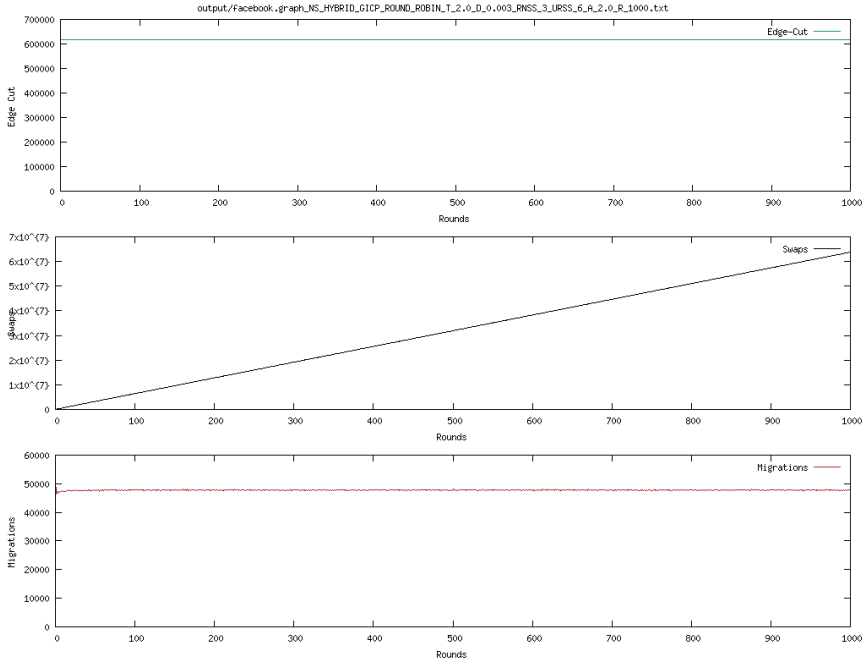

Figure 7: Task 2 – facebook graph with exponential annealer

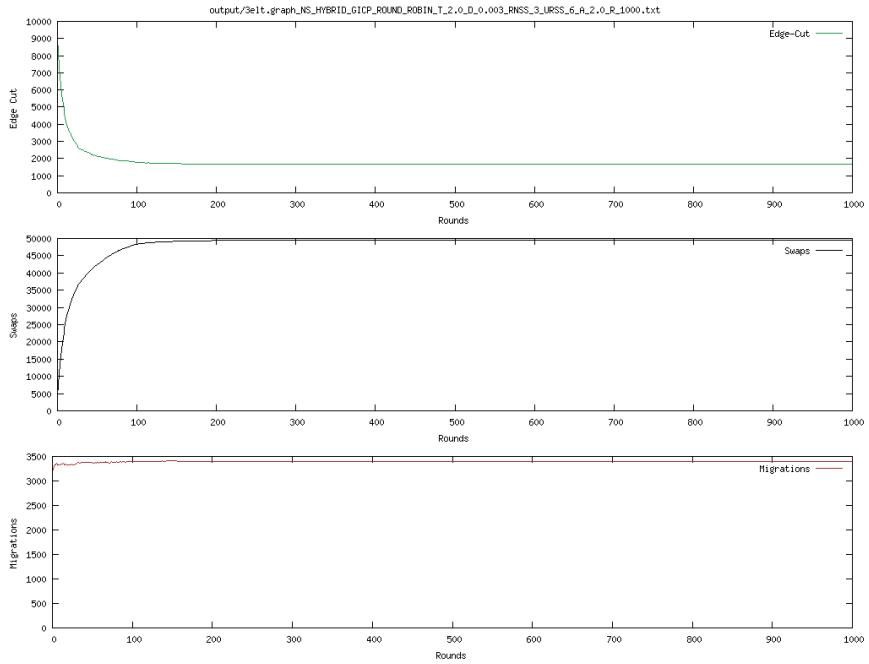

Figure 8: Bonus task – 3elt graph with exponential annealer

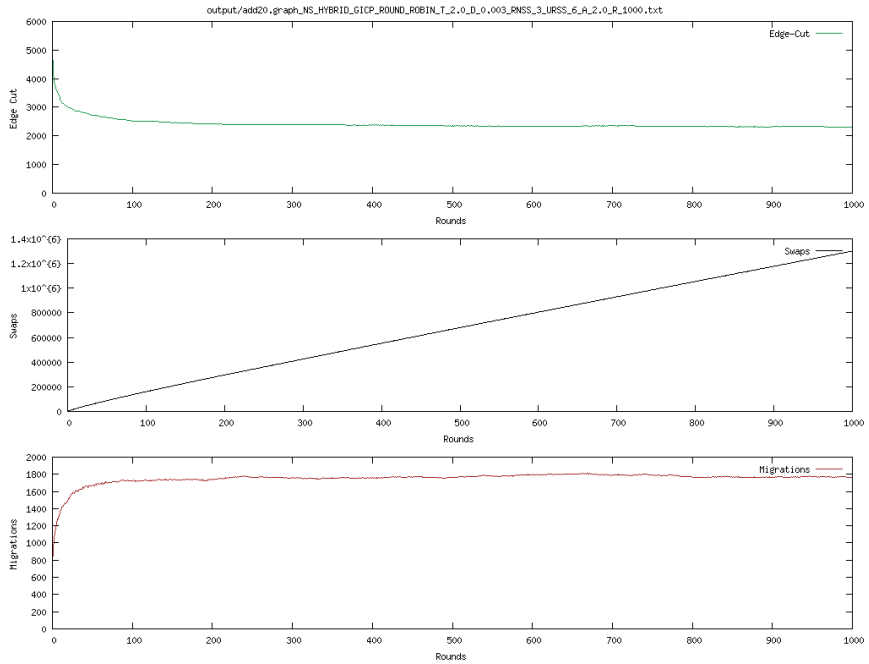

Figure 9: Bonus task – add20 graph with exponential annealer

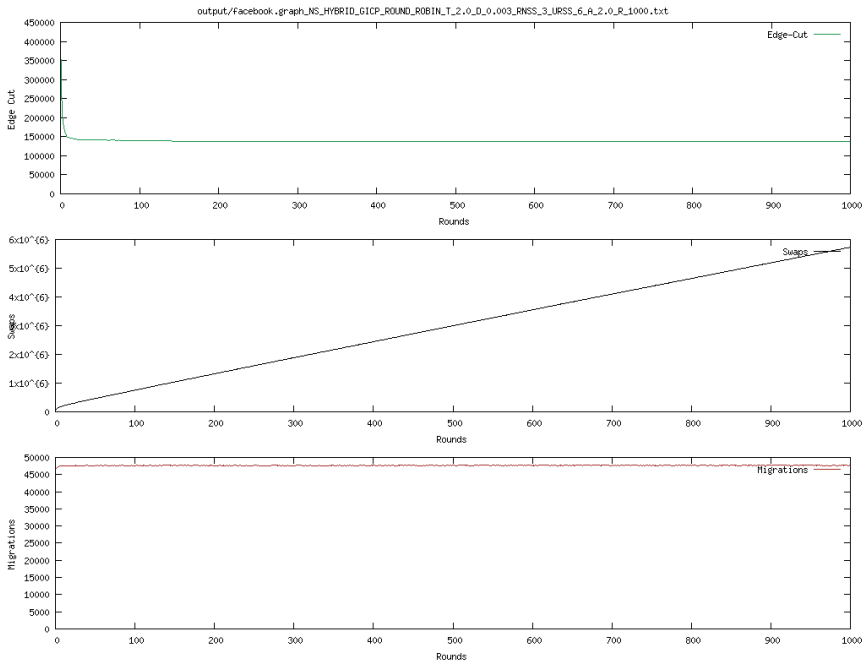

Figure 10: Bonus task – facebook graph with exponential annealer