Homework 5: K-way Graph Partitioning Using JaBeJa

By Group45

1 Introduction

The graph partitioning problem often called the min-cut problem, is about splitting a graph into different parts while trying to have as few edges between them as possible. There's another challenge called balanced or uniform graph partitioning, where it's important to have an equal number of nodes in each part. In big graph processing systems, distributed graph partitioning is super important. The main aim is to divide a graph into smaller pieces that can be handled more efficiently.

The paper we're looking at deals with handling big graphs, like those showing connections on social media with billions of users and billions of links. This is a tough challenge because of the massive size of these graphs.

This assignment is about understanding distributed graph partitioning using gossip-based peer-to-peer techniques following the given paper. We're paying special attention to JaBeJa, especially a technique in the paper called JA-BE-JA. It's a way of using gossip among peers to split a graph into parts, and we want to understand how it works.

2 Overview of the JA-BE-JA Algorithm

JA-BE-JA combines a gossip-based communication strategy with a randomized initialization, local search based on color swaps, and simulated annealing for iterative refinement. The ultimate goal is to achieve a k-way graph partitioning with an emphasis on optimizing both edge and node cuts in a distributed and decentralized manner.

- **Gossip-Based Approach:** JA-BE-JA adopts a gossip-based approach for distributed graph partitioning. In gossip-based algorithms, nodes in a network exchange information iteratively and in a decentralized manner.
- **Randomized Initialization:** The algorithm employs a randomized initialization strategy. Colors are assigned to nodes uniformly at random at the beginning of the algorithm.
- **Local Search Operator:** Utilizes a local search operator based on color swaps between neighboring nodes. Nodes attempt to change their color to the most dominant color among their neighbors. Color swaps are used to achieve the desired partitioning.

- **Simulated Annealing:** Incorporates simulated annealing to guide the search and prevent the algorithm from getting stuck in local optima. Simulated annealing involves gradually decreasing the probability of accepting worse solutions during the iterative process.
- **Optimization Goals:** Aim to iteratively find k-way graph partitioning. Focuses on optimizing both edge and node cuts, indicating a joint optimization criterion.

3 Tasks implementations

3.1 Task 1: sampling policy and swapping technique

In the first task, we implemented the Ja-Be-Ja algorithm by modifying the provided scaffolding source code for the Ja-Be-Ja simulation. Specifically, we needed to modify the JaBeJa.java class and implement all methods that are marked with TODO tags based on the description given in the paper:

- sampleAndSwap(...) method
- and the findPartner(...) method.

These two methods contains key components for the JA-BE-JA algorithm operation namely; the sampling policy and the partner selection technique. These components play key roles in determining how nodes interact and exchange information during the distributed graph partitioning process. Let's delve into the details:

I. **Sampling Policy:** The sampleAndSwap(...) method is responsible for defining how a node selects a set of candidate nodes for color swapping. Three possible sampling policies are considered:
   a. Local (L) Policy: Each node considers its directly connected nodes (neighbors) as candidates for color exchange. Nodes attempt to swap colors with their immediate neighbors.
   b. b. Random (R) Policy: Each node selects a uniform random sample of nodes in the graph. Various techniques exist for taking a uniform sample of the graph at a low computational cost.
   c. Hybrid (H) Policy: Immediate neighbor nodes are initially selected (local policy). If the local selection fails to improve the pair-wise utility, the node is given another chance for improvement. The node selects a node from its random sample (random policy) to attempt further improvement.

II. **Partner Selection**: The findPartner(...) method is responsible for defining how a node selects a swap partner, i.e., the node with which it exchanges colors. To decide if two nodes should swap colors, a function to measure the pair-wise

utility of a color exchange is used. This function evaluates the benefit or cost of swapping colors between two nodes. Task 2: Study the effect of simulated annealing

3.2 Task 2: Study the effect of simulated annealing

The task is to tweak different JaBeJa configurations to find the smallest edge cuts for the given graphs. Especially will study the effect of simulated annealing. Modify the saCoolDown() method.

- **Task 2.1**: The original saCoolDown() method in the Ja-Be-Ja implementation utilizes a linear function to reduce temperature, with the temperature being scaled by the cost function. To introduce a different cooling function for simulated annealing, we have opted for the widely used exponential decay function. We modify ***saCoolDown*** (..) method to integrate exponential decay.
- **Task 2.2**: An option to restart T to initial value is added to avoid being stuck at a local minimum. We observe that after 200 to 400 rounds of the simulation, the minCut value gets steady. Though we optionally restart the temperature T after convergence which is observed after around 400 rounds.

3.3 Task 3: Study the effect of simulated annealing

The acceptance probability (P) is a value between 0 and 1, representing the likelihood of accepting the new solution. The algorithm explores the solution space by iteratively making random moves and accepting or rejecting those moves based on the acceptance probability function.

Back to the implementation, the acceptance probability (P) is implemented in the method findPartner() using an old value olds and a new value news as follows:

P = e^((new_value – old_value) / T)

3.4 Results discussion

When we ***run.sh -help***, we can see many options (some are given, and others are added by us) these simulation parameters will be used to run different scenarios:

- -graphs: List of graph names.
- node_policies: List of node selection policies.
- -alpha N : Alpah parameter (default: 2.0)
- -delta N : Simulated annealing delta. (default 0.003)
- -optionAnnealing VAL : Annealing option, linear or exp (default: linear)
- -roundtoRestart N: choose to set T to the intaial value after n
- -optionAccepyanceProb VAL: option, linearAccepatnce or expAccepatnce

3.5 Task1 Results

For Task 1, we run the simulation using run_Task1.sh script in which the default settings are used along with considering two different node policies (LOCAL and HYBRID) and

linear annealing (delta = 0.003). The annealing is linear decreasing over time as described in the paper. We notice that the hybrid node selection policy resulted in a minimum edge cut compared to the local node selection policy.

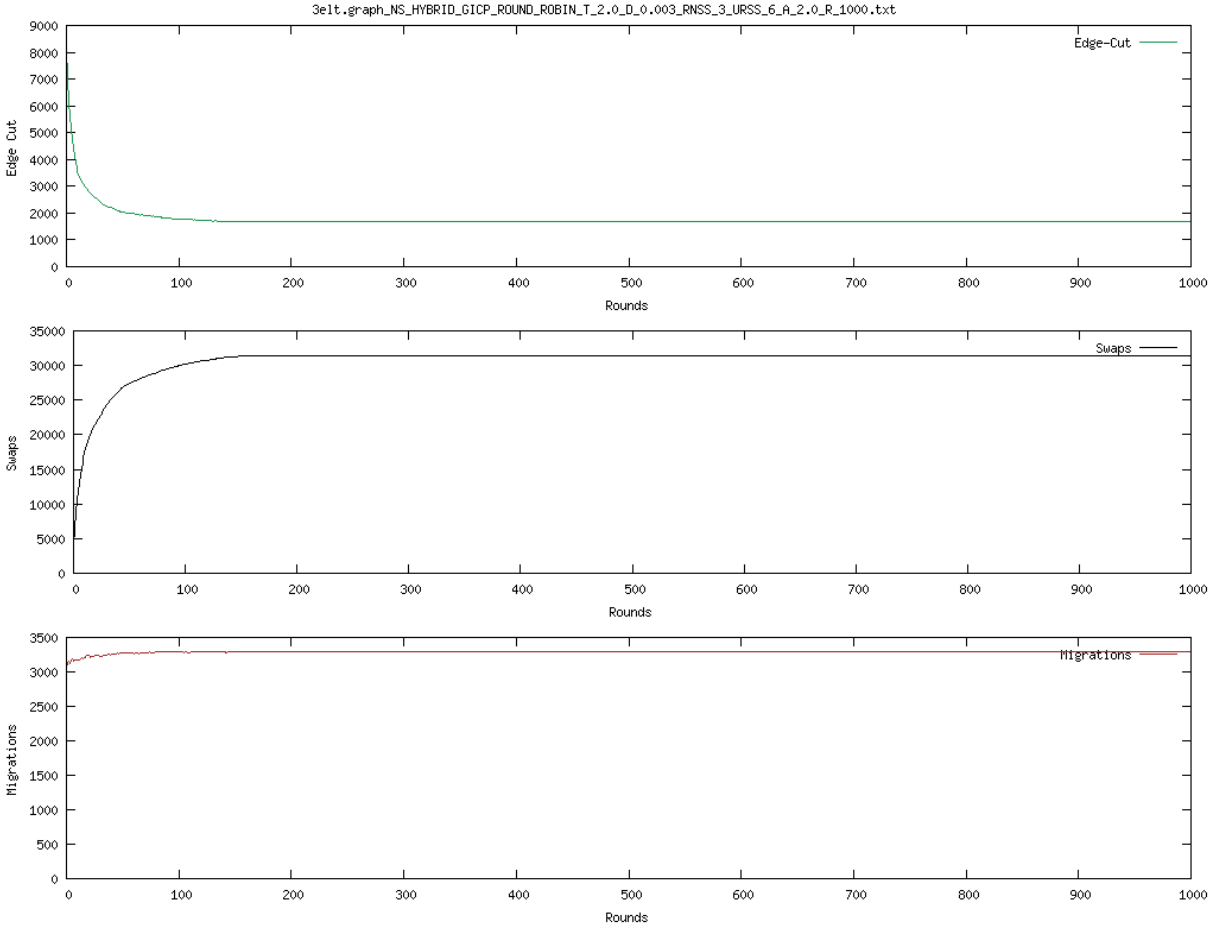

Figure 1: 3elt graph with linear annealing, hybrid node selection policy

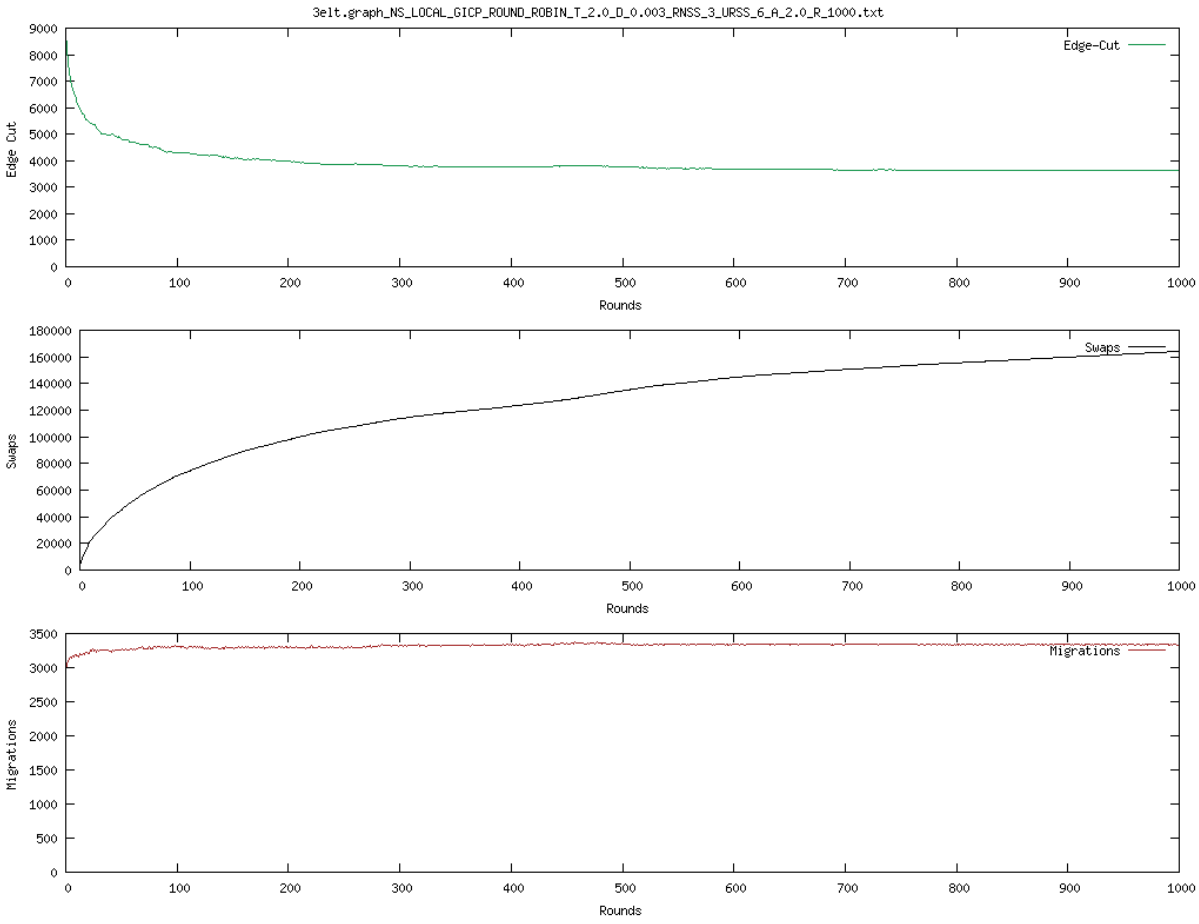

Figure 2: 3elt graph with linear annealing, local node selection policy

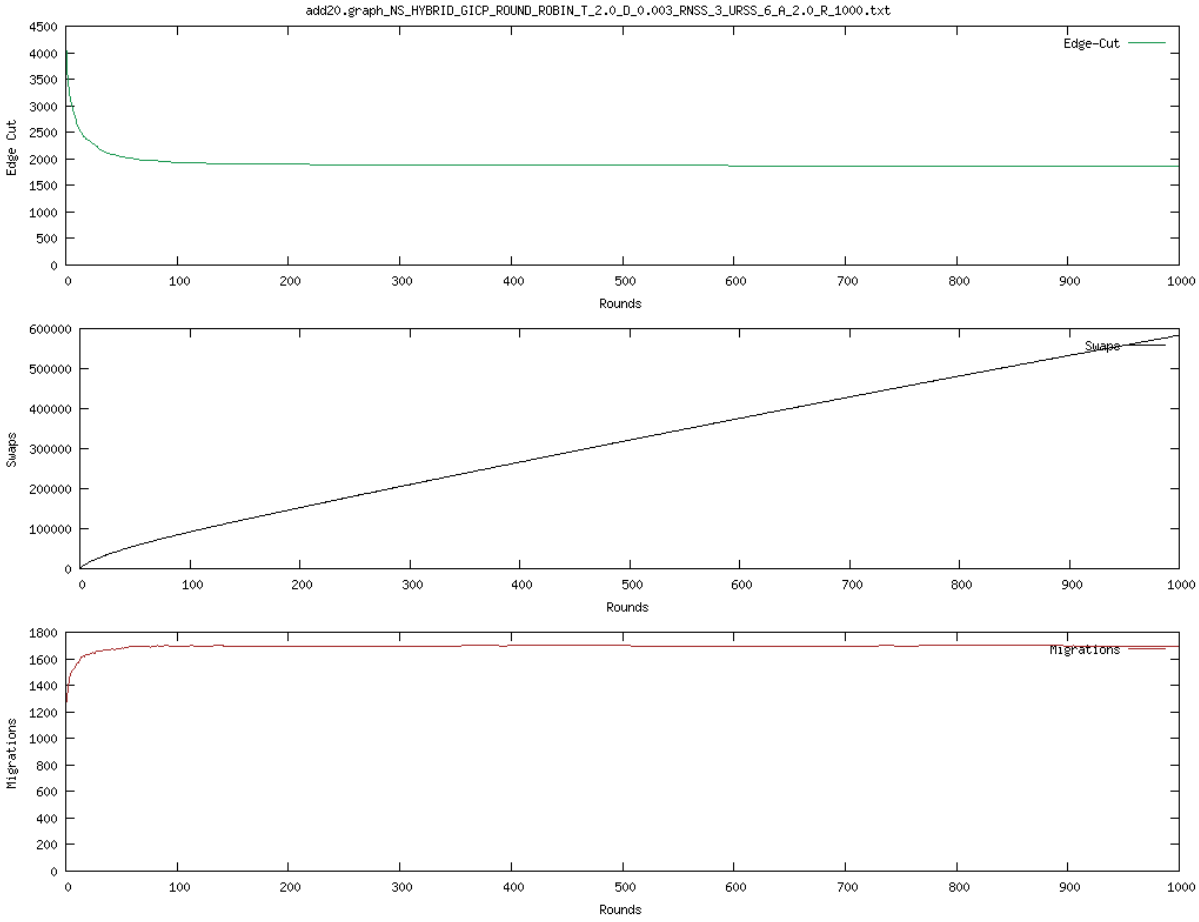

Figure 3: **add20 graph with linear annealing, hybrid node selection policy**

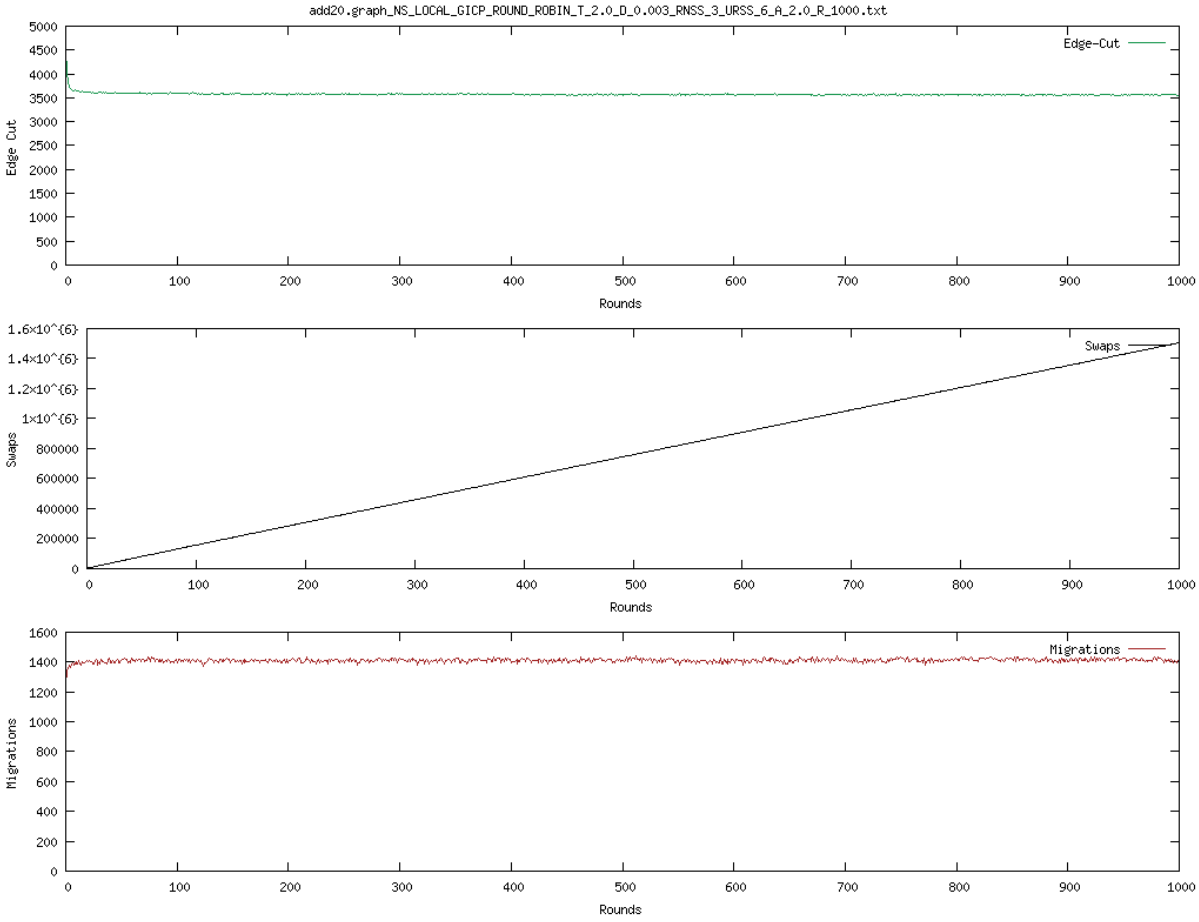

Figure 4: add20 graph with linear annealing, local node selection policy

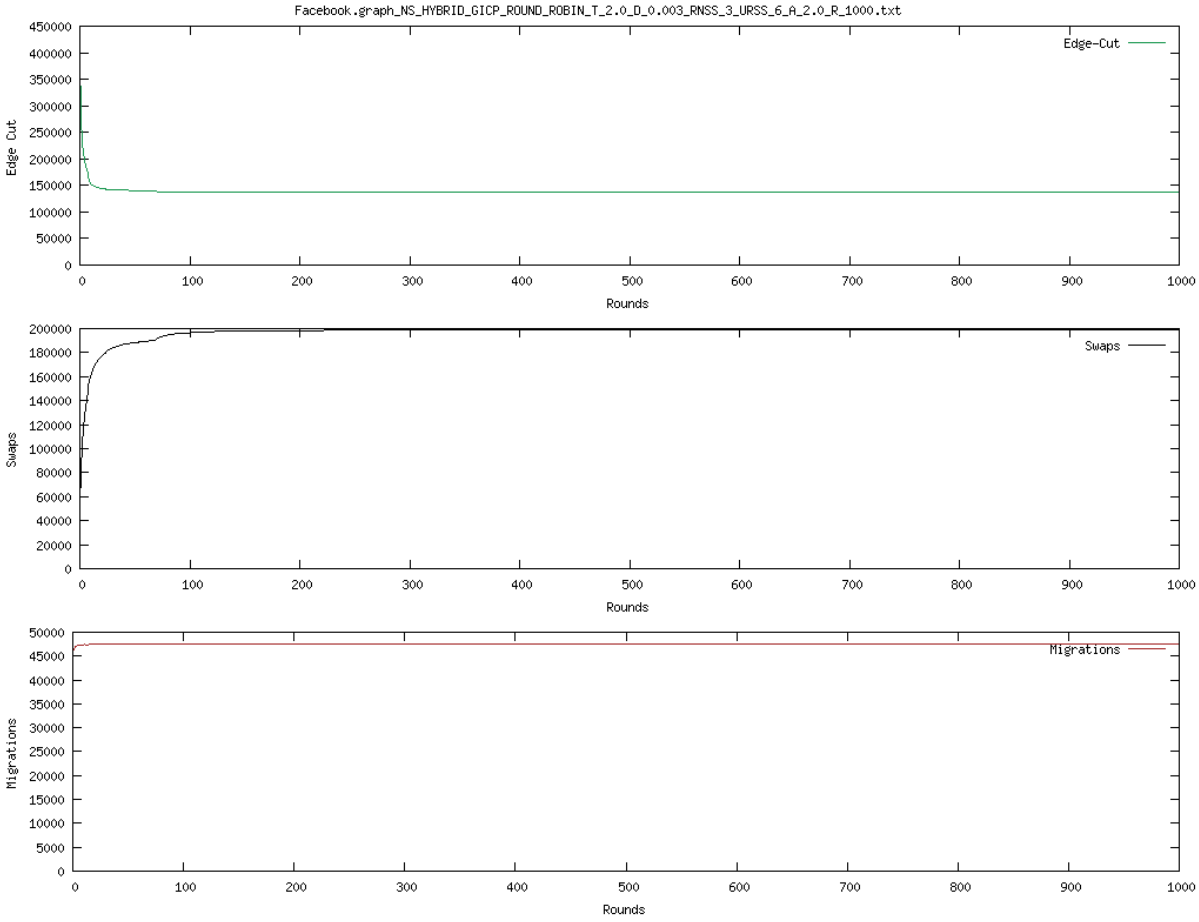

Figure 5 Facebook.graph_NS_HYBRID_

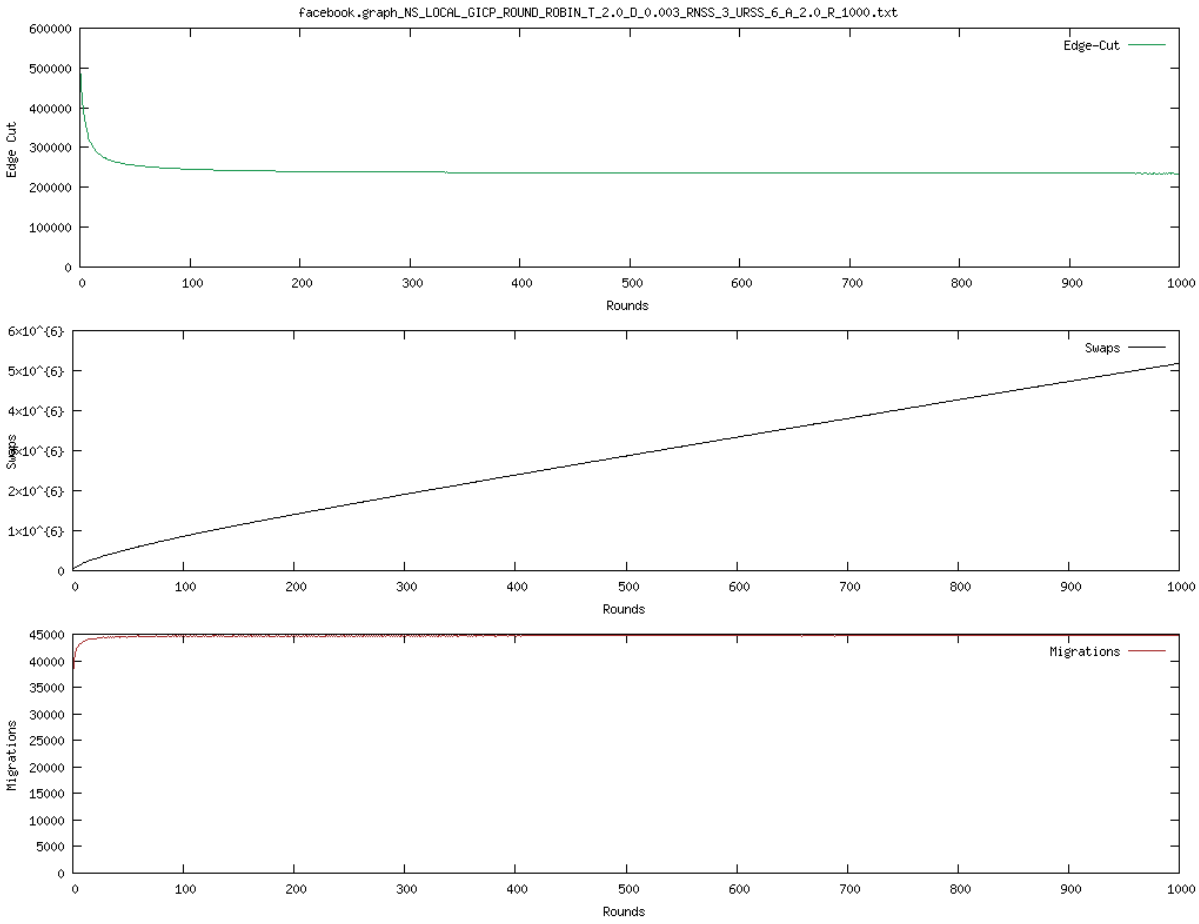

Figure 6: Facebook, Local node selection

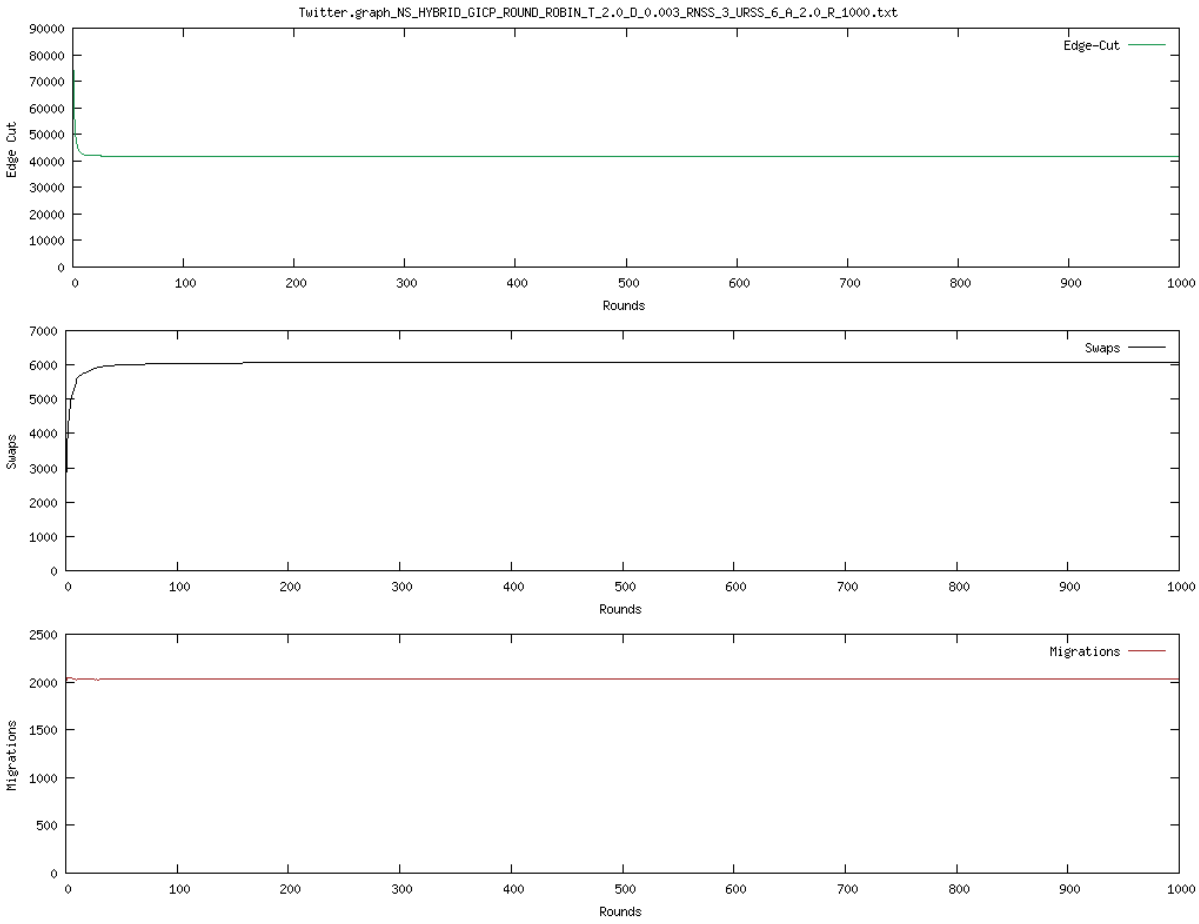

Figure 7: Twitter, Hybrid NS

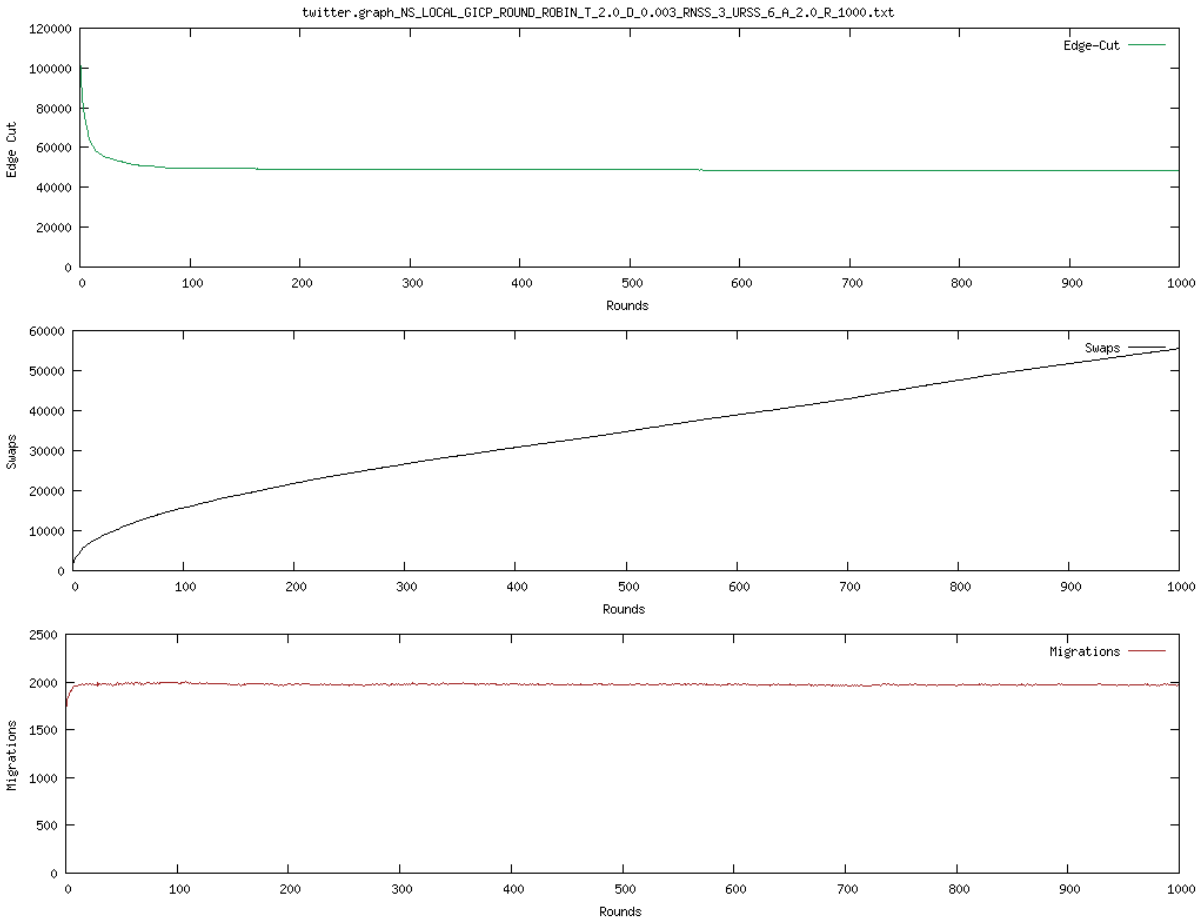

Figure 8 : Twitter, Local NS

3.6 Task 2 Results

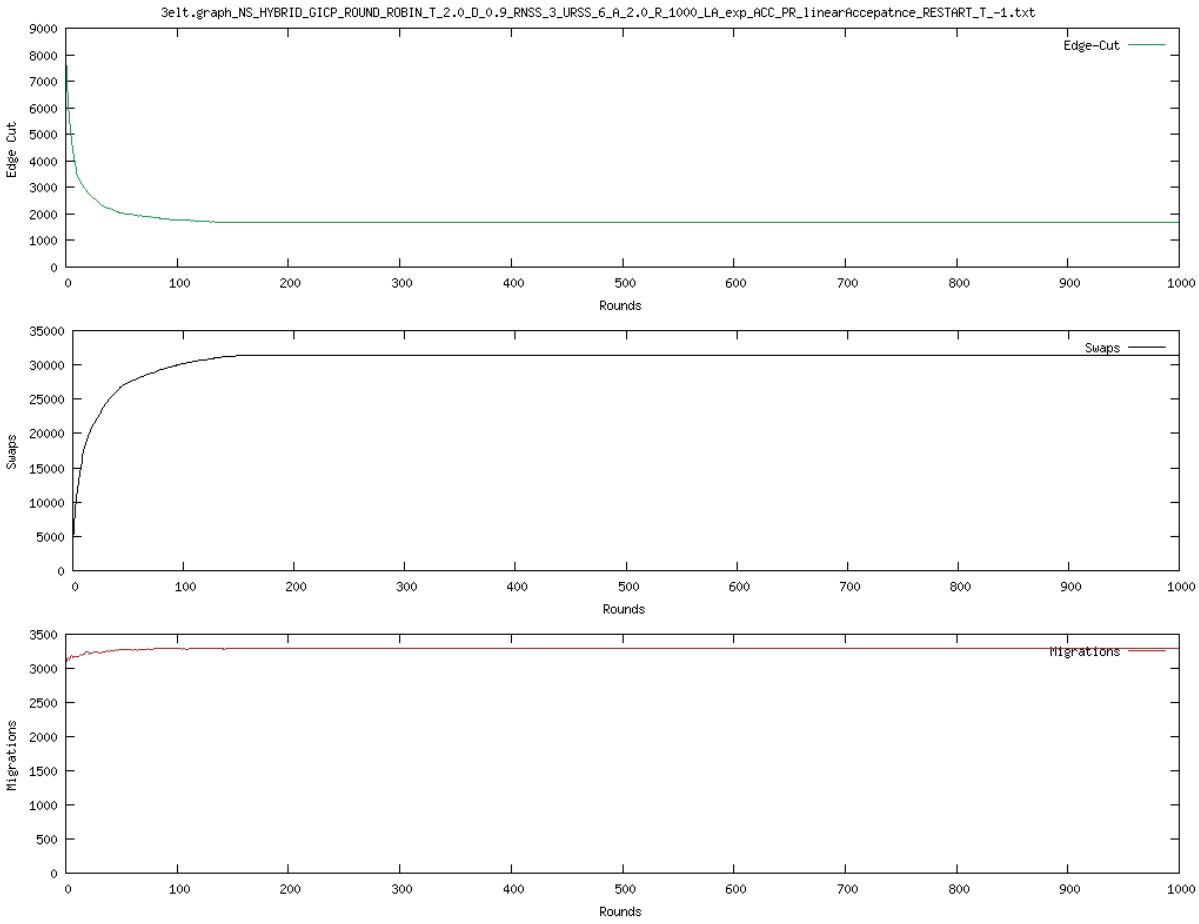

Figure 9: 3elt graph with exp annealer, temperature = 2, delta = 0.9, and without restart

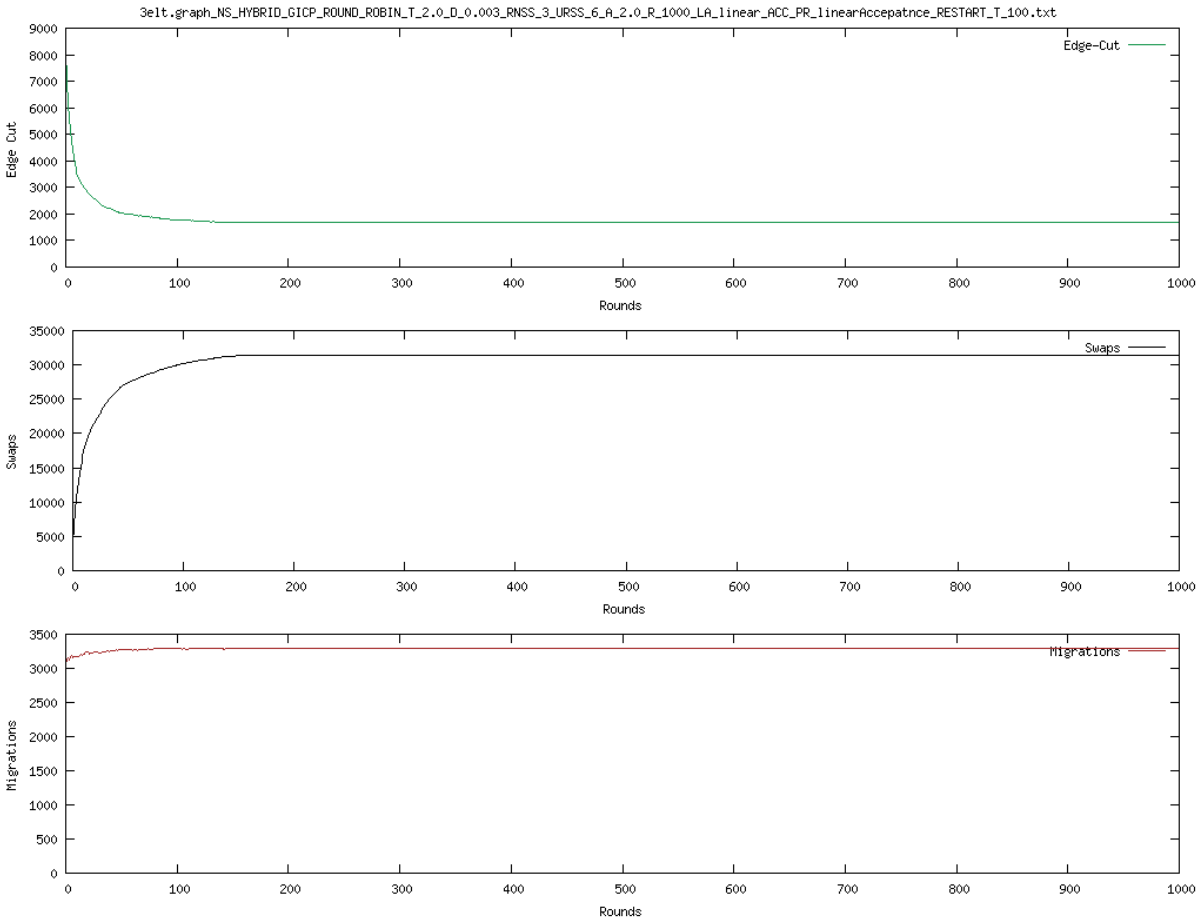

Figure 10 3elt graph with linear annealer, temperature = 2, delta = 0.9, and with restart after 100

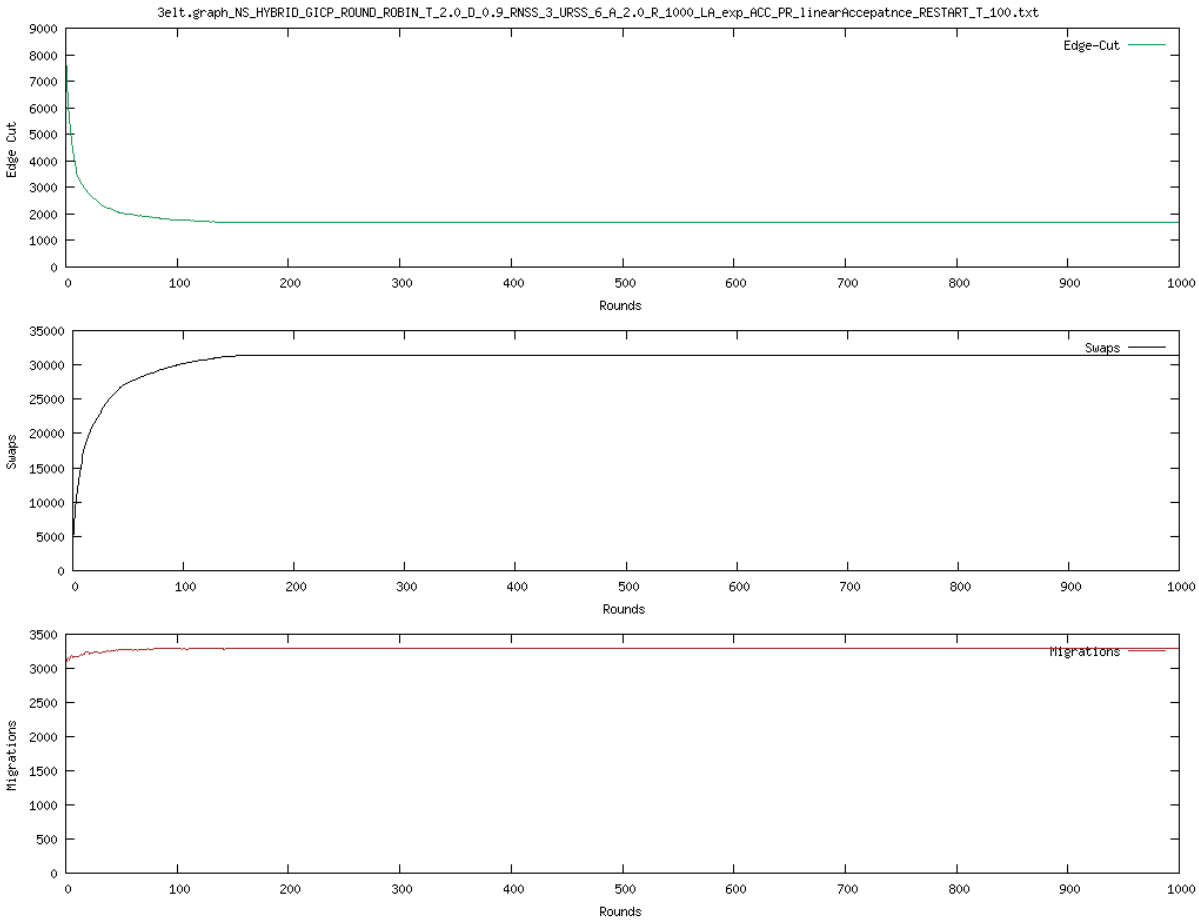

Figure 3elt graph with exp annealer, temperature = 2, delta = 0.9, and with restart after 100

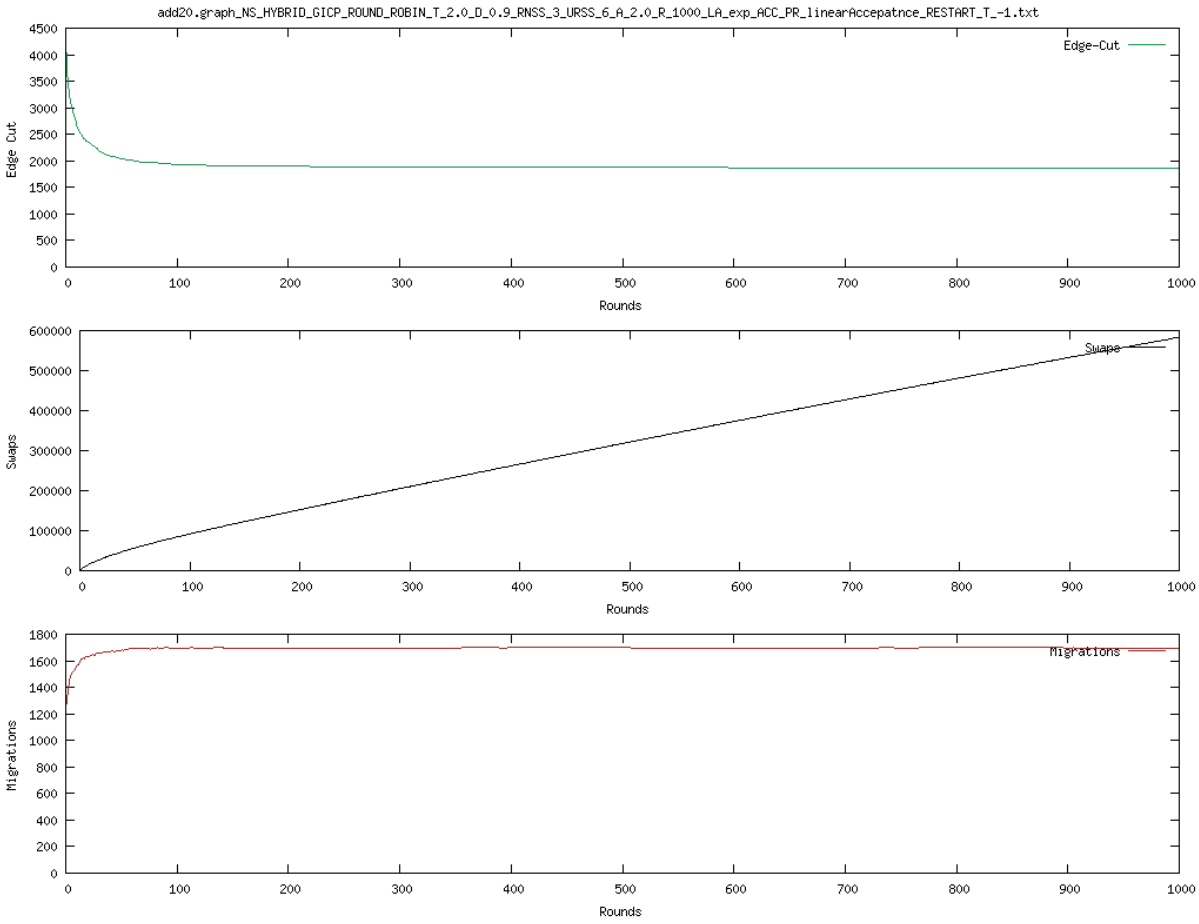

Figure 11: add20 graph with exp annealer, temperature = 2 delta = 0.9, and without restart

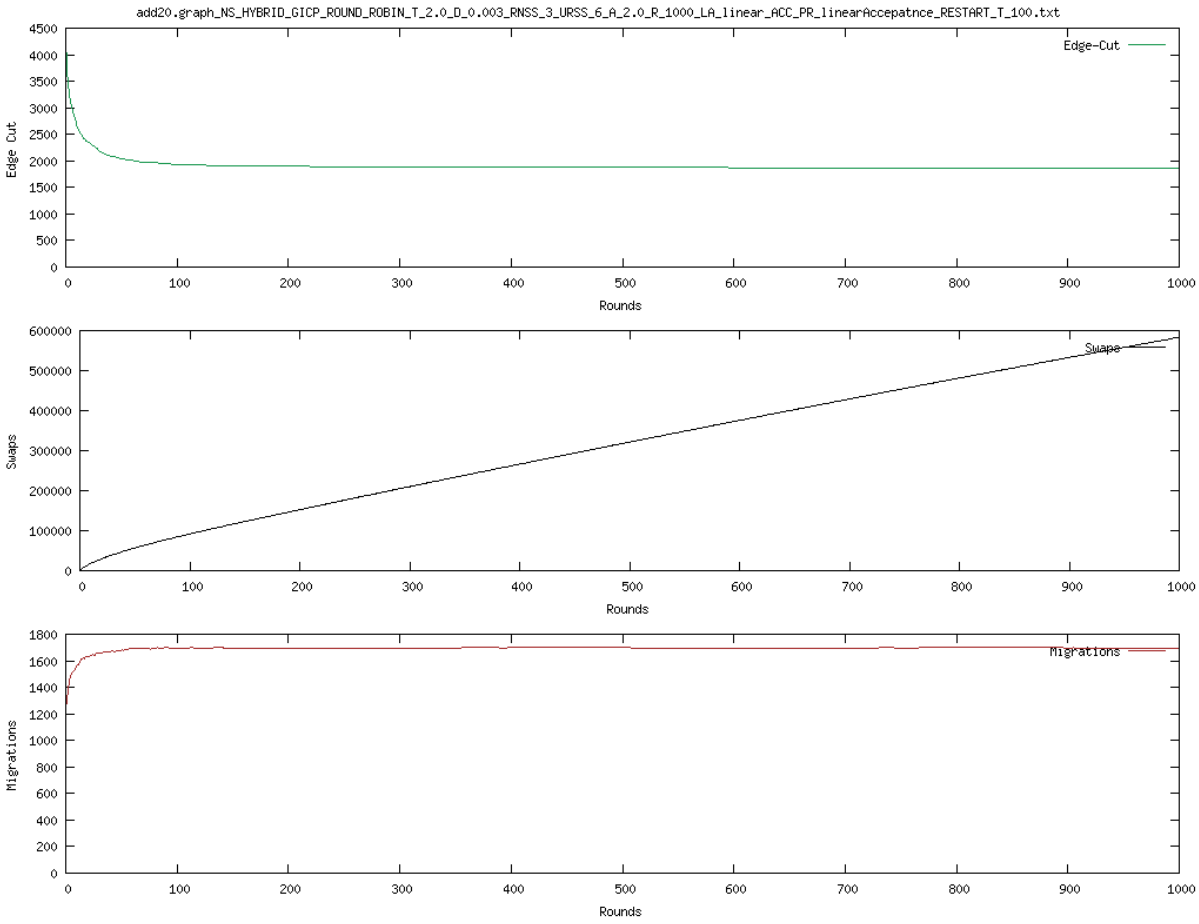

Figure 12 add20 graph with linear annealer, temperature = 2 delta = 0.9, and with restart at 100

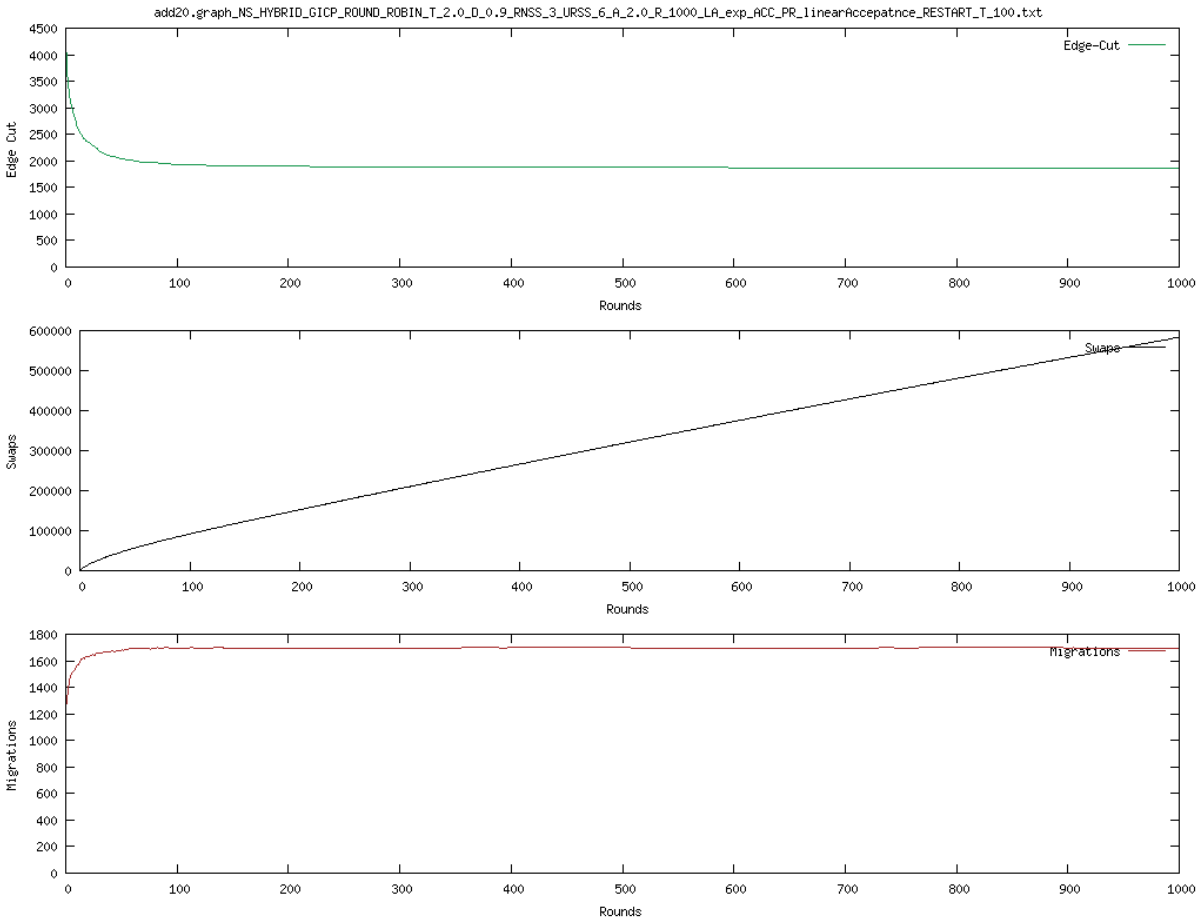

Figure 13: add20 graph with exp annealer, temperature = 2 delta = 0.9, and with restart at 100

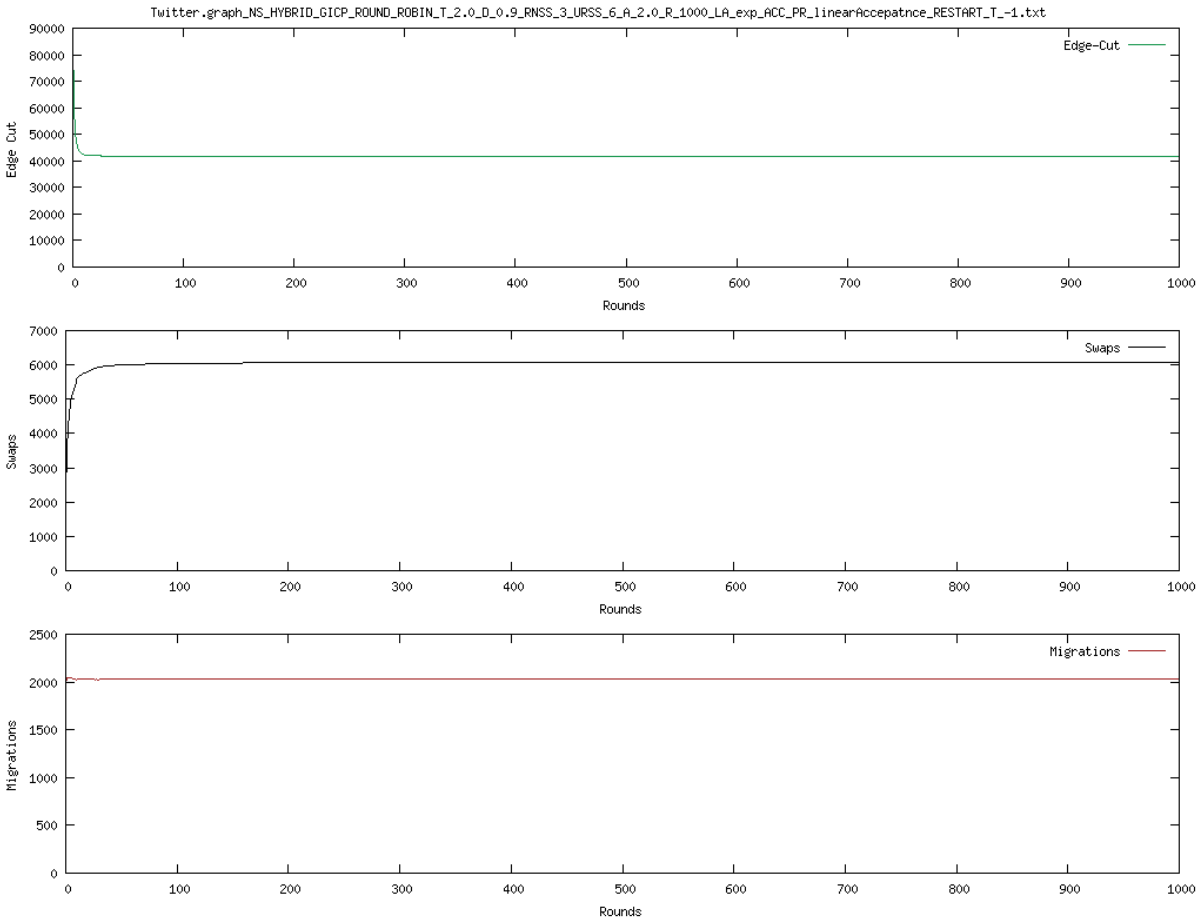

Figure 14: Twitter graph with exp annealer, temperature = 2 delta = 0.9, and without restart

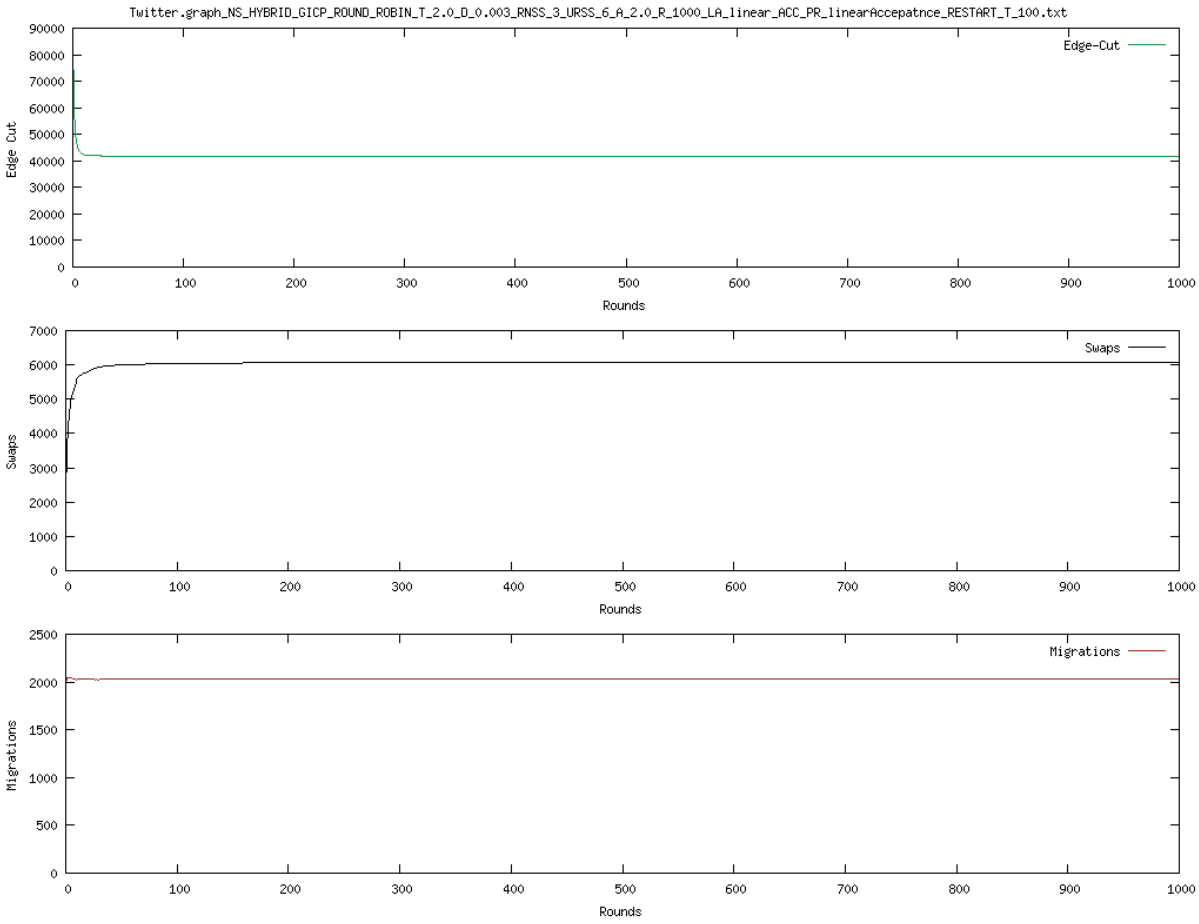

Figure 15: Twitter graph with linear annealer, temperature = 2 delta = 0.9, and with restart at 100

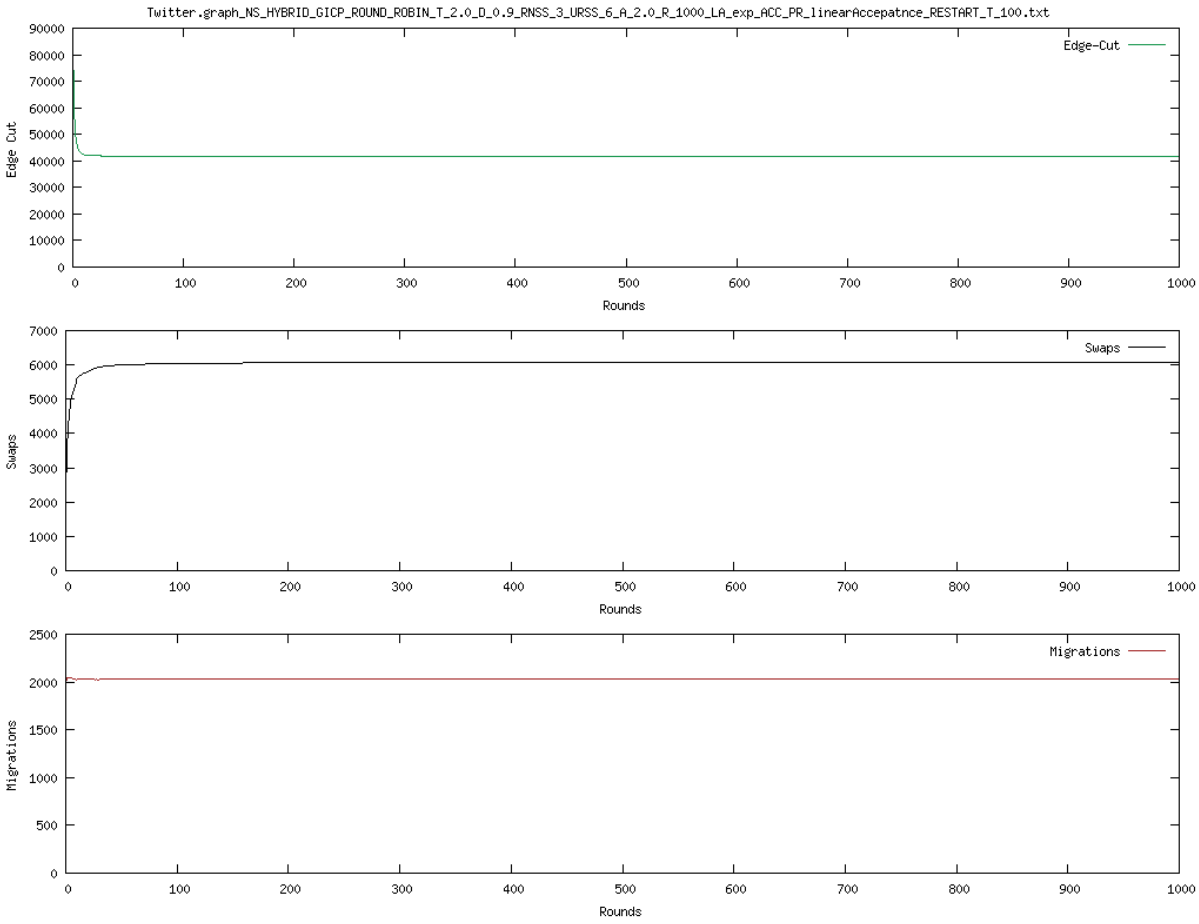

Figure 16: Twitter graph with exp annealer, temperature = 2 delta = 0.9, and with restart at 100

3.7 Task 3 Results

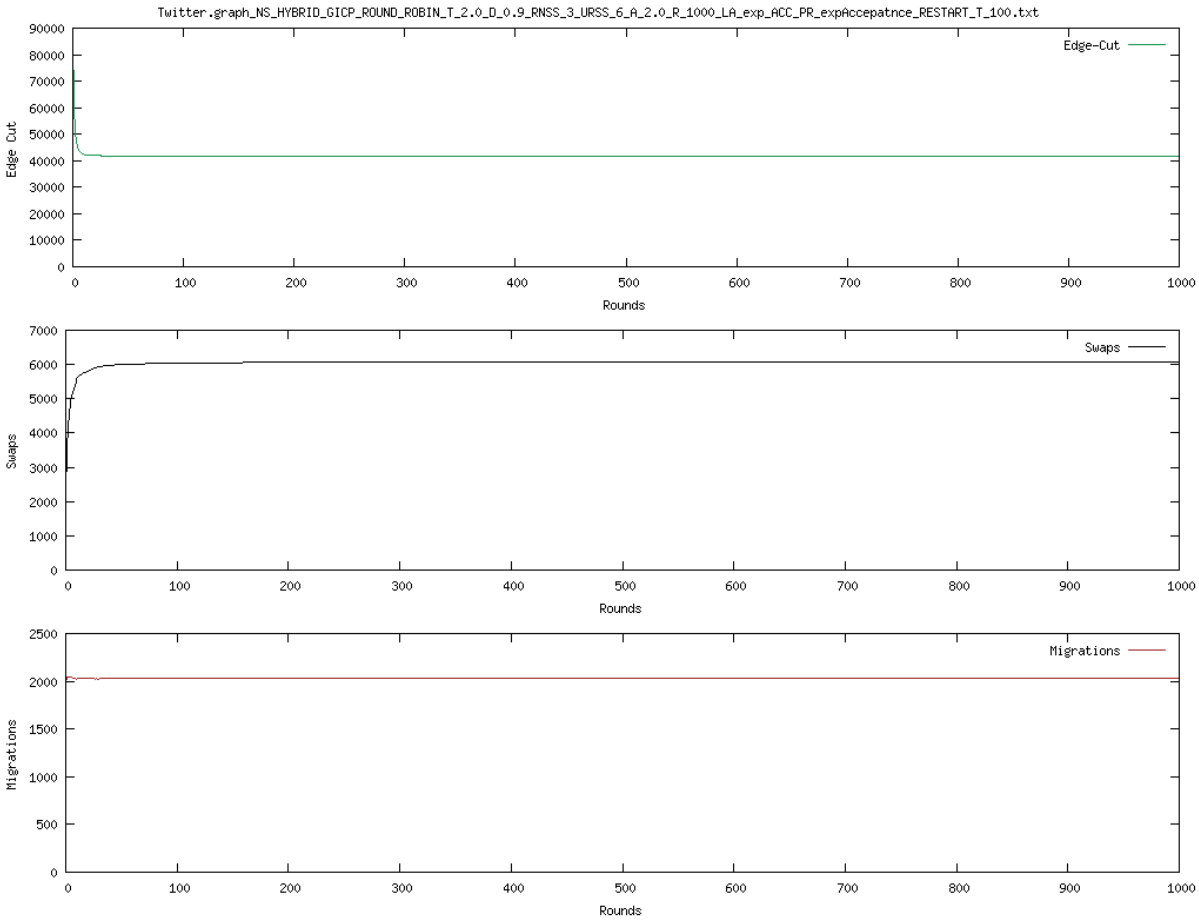

Figure 17: **Twitter graph with exp annealer, Acceptance Prob exp , and with restart at 100**

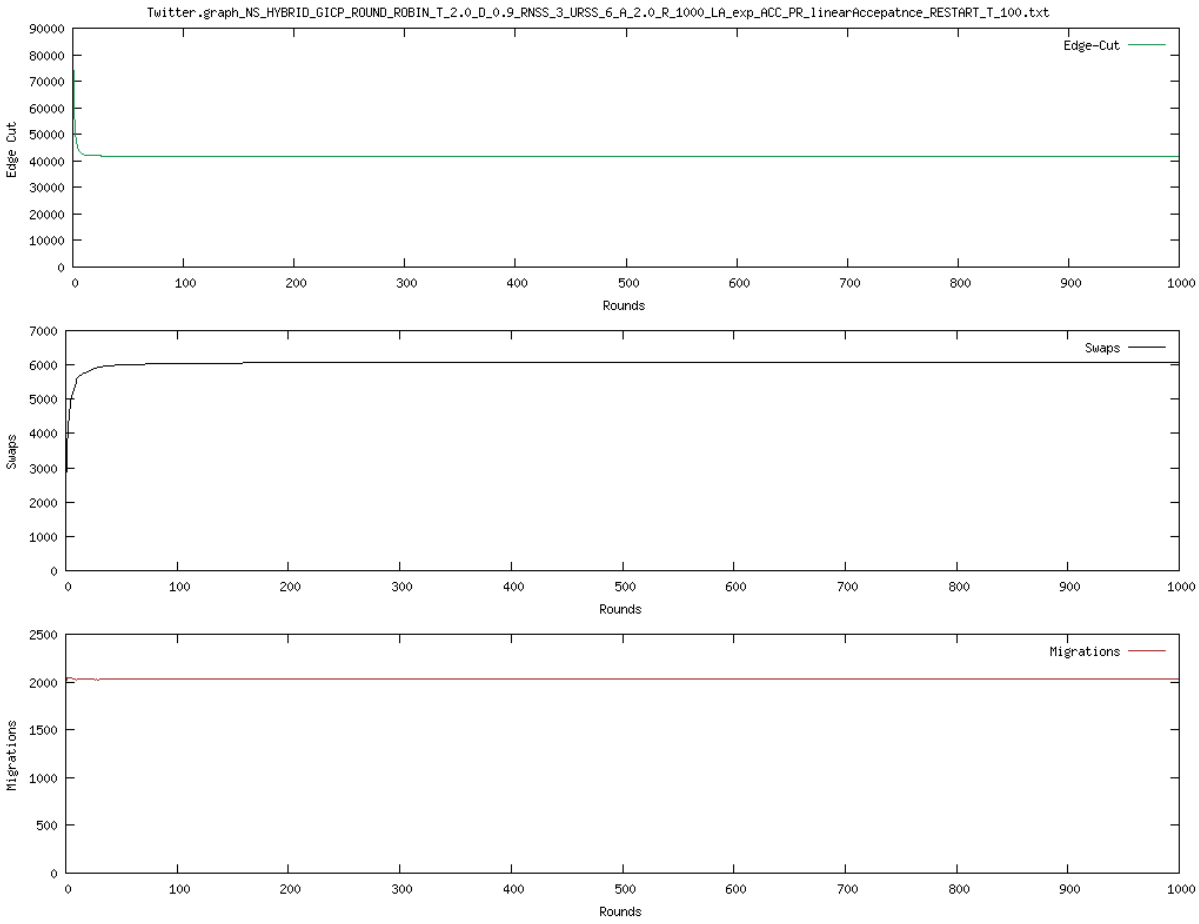

Figure 18: **Twitter graph with exp annealer, Acceptance Prob linear , and with restart at 100**

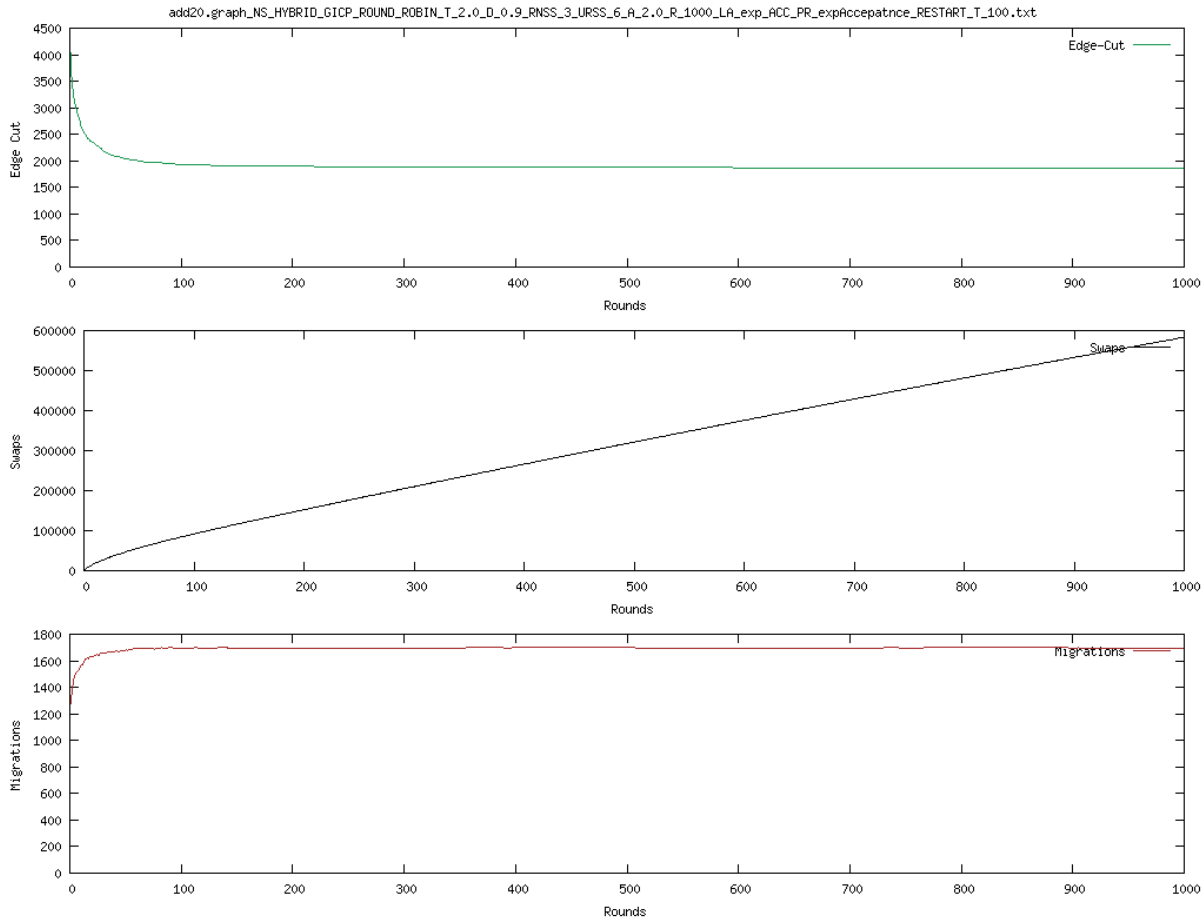

Figure 19: **add20 graph with exp annealer, Acceptance Prob exp , and with restart at 100**

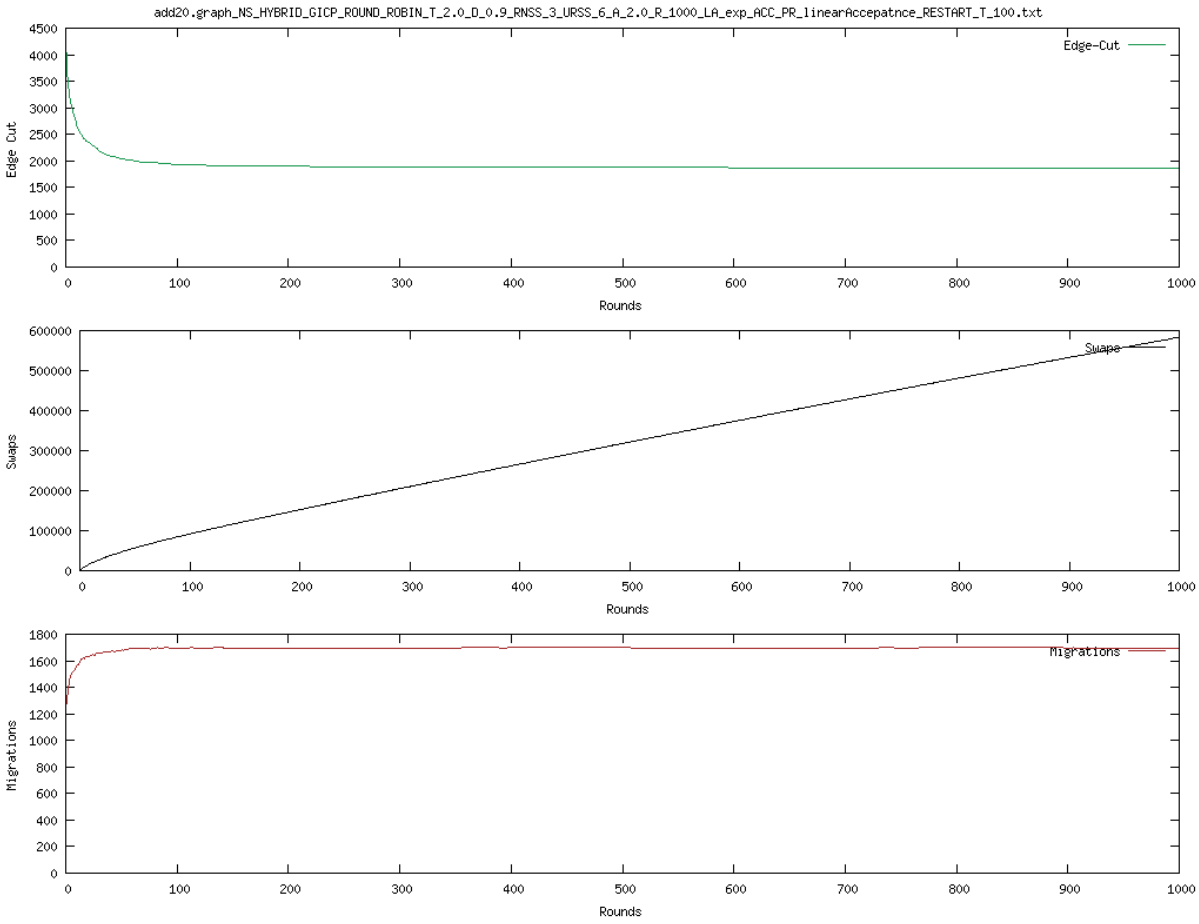

Figure 20: **add20 graph with exp annealer, Acceptance Prob linear , and with restart at 100**