ID2222 Data Mining: Homework 5

Distributed Graph Partitioning with JaBeJa

Emanuele Minotti Margherita Santarossa

Group 33

December 8, 2025

Abstract

This report presents the implementation and analysis of the JaBeJa algorithm for balanced graph partitioning. The study is divided into two main tasks: the implementation of the standard algorithm with linear annealing (Task 1) and the optimization of the partitioning quality using an exponential simulated annealing approach with a restart mechanism (Task 2). The algorithms were tested on various datasets, including `3elt`, `add20`, and `twitter`, analyzing the impact of different configurations on the edge-cut metric.

Contents

1 Introduction

Graph partitioning is a fundamental problem in distributed computing, essential for processing large-scale graphs efficiently. The goal is to divide the graph nodes into k partitions of approximately equal size while minimizing the number of edges crossing the partitions (edge-cut). In this assignment, we implement **JaBeJa**, a fully distributed algorithm that uses local search and simulated annealing techniques. We explore two different variations of the algorithm:

1. **Task 1:** Standard implementation with linear temperature decay.
2. **Task 2:** Optimized implementation with Metropolis-Hastings acceptance probability, exponential decay, and a restart mechanism to escape local minima.

2 Task 1: Standard JaBeJa Implementation

2.1 Implementation Details

In the first task, the standard JaBeJa algorithm was implemented. The core mechanism relies on swapping nodes between partitions to minimize the energy of the system. The temperature T decreases linearly over time, and the acceptance of a swap is determined by the utility function multiplied by the temperature.

- **Cooling Schedule:** Linear. $T_{new} = T_{old} - \delta$.
- **Utility Function:** $U = (E_{new} \cdot T) - E_{old}$.
- **Acceptance:** Swap is accepted if $U > highestUtility$.

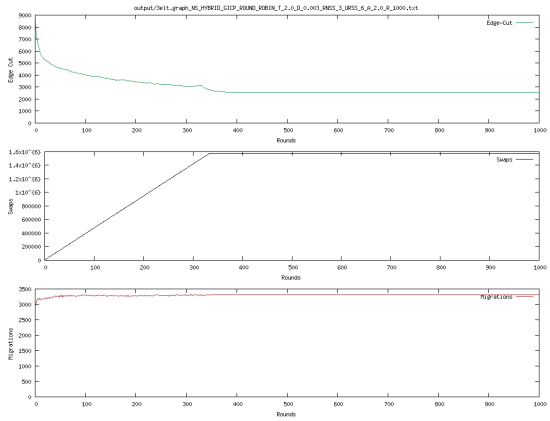
The relevant code snippet from `Jabeja.java` is shown below:

```
protected void saCoolDown(){
    if (T > 1)
        T -= config.getDelta();
    if (T < 1)
        T = 1;
}
```

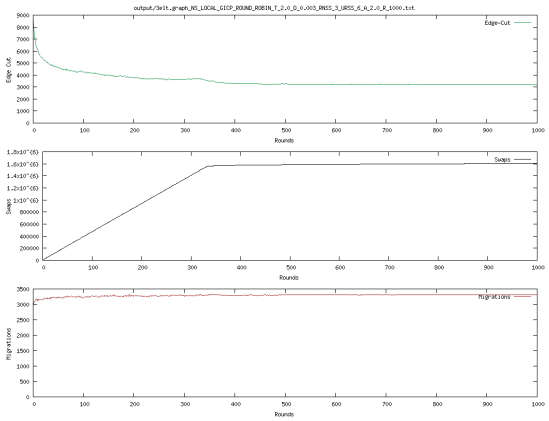
Listing 1: Linear Cooling in Jabeja.java

2.2 Results and Analysis

We compared the **LOCAL** and **HYBRID** node selection policies. The `HYBRID` policy, which combines local neighbors and random sampling, generally performs better as it allows nodes to explore a broader portion of the graph, reducing the likelihood of getting stuck in local optima.



(a) 3elt Hybrid Policy



(b) 3elt Local Policy

Figure 1: Comparison of node selection policies for the 3elt graph.

The plots in Figure 1 show the edge-cut reduction over time. The Hybrid policy achieves a lower final edge-cut compared to the Local policy, confirming the importance of random sampling in distributed partitioning. Similar considerations also apply to the add20 and twitter datasets, whose plots are available in the GitHub repository.

3 Task 2: Simulated Annealing Extensions

3.1 Modified Acceptance Probability and Cooling

In Task 2, we modified the simulated annealing mechanism to improve convergence and solution quality.

1. **Exponential Cooling:** The temperature decays geometrically ($T = T \cdot \delta$), allowing for a slower cooling at high temperatures and rapid stabilization at low temperatures.

2. **Metropolis Acceptance:** We adopted the standard probability function $P(accept) = e^{\frac{\Delta E}{T}}$.

3. **Restart Mechanism:** To further avoid local minima, the temperature is reset to the initial value every 400 rounds.

The implementation in `SA_Jabeja.java`:

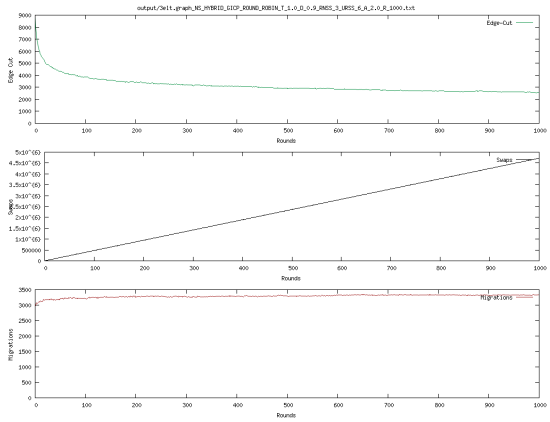
```
@Override
protected void saCoolDown() {
    int restartRound = 400;
    if (T > 0.001f)
        T *= config.getDelta(); // Exponential cooling
    else
        T = 0.001f;

    // Restart mechanism
    if (round > 0 && round % restartRound == 0) {
        T = config.getTemperature();
    }
}
```

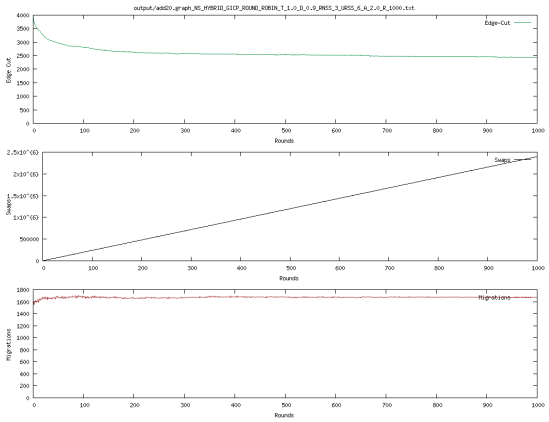
Listing 2: Exponential Cooling with Restart

3.2 Experimental Results

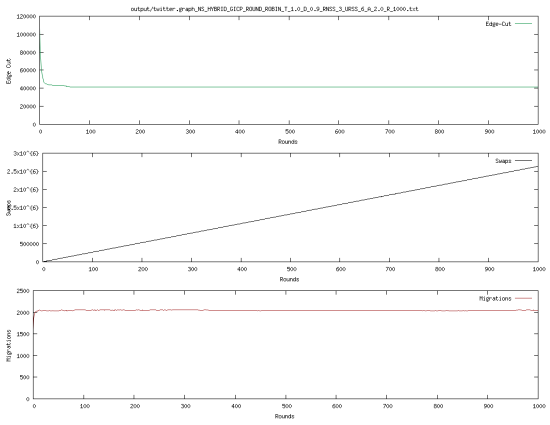
We tested the new configuration with $T = 1.0$ and $\delta = 0.9$ (as indicated by the filenames `T_1.0_D_0.9.png`). The restart mechanism (set at 400 rounds) creates periodic spikes in the exploration phase, allowing the algorithm to escape local minima that the standard linear approach would settle into.



(a) 3elt (SA + Restart)



(b) add20 (SA + Restart)



(c) twitter (SA + Restart)

Figure 2: Performance of the optimized SA algorithm with restart mechanism. The effect of reheating is visible as the edge-cut fluctuates before settling again.

As observed in Figure 2, the restart mechanism effectively "shakes" the system. While the edge-cut temporarily increases during the restart (high temperature), it often converges to a lower minimum in the subsequent cooling phase compared to the previous cycle.

4 Bonus Task: Alternative Acceptance Probability

For the optional task, we designed and implemented a **custom acceptance probability function** to change how the algorithm navigates the solution space.

4.1 Formulation

In Task 2, we utilized the standard Metropolis criterion, which accepts "bad" moves based on a probability calculated via an exponential function:

$$P(accept) = e^{\frac{U_{new} - U_{old}}{T}} > \text{random}(0, 1) \tag{1}$$

In this bonus task, we replaced this probabilistic approach with a deterministic threshold that evolves with the temperature. The new acceptance logic is defined as:

$$\frac{U_{new}}{U_{old}^{1/T}} > 1 \quad \Longrightarrow \quad U_{new} > U_{old}^{1/T} \tag{2}$$

4.2 Analysis of the New Function

This new formula radically changes the acceptance dynamic:

- **High Temperature ($T \approx 1$):** The exponent $1/T$ is close to 1. The condition resembles $U_{new} > U_{old}$, behaving similarly to a greedy approach but modulated by the remaining temperature factor.
- **Low Temperature ($T \to 0$):** As T decreases, the exponent $1/T$ grows significantly. This makes the denominator $U_{old}^{1/T}$ extremely large, effectively rejecting almost all swaps unless U_{new} provides a massive improvement, thereby "freezing" the configuration much harder than the standard exponential function.

4.3 Experimental Results

We tested this new function using specific parameters found in the bonus analysis: initial temperature $T = 1.0$, a delta $\delta = 0.01$, and a restart interval of 400 rounds.

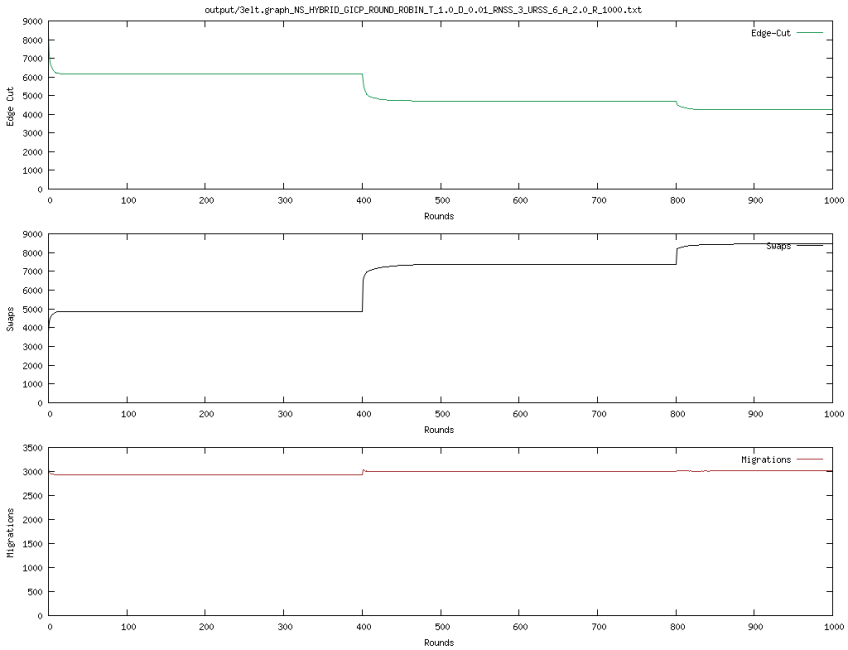


Figure 3: Performance of the custom acceptance probability function on the 3elt graph with restart enabled.

The plot in Figure 3 clearly illustrates the interaction between the custom acceptance function and the restart mechanism. Due to the aggressive nature of the power function term ($U_{old}^{1/T}$), the system "freezes" rapidly as T decreases, settling into a local minimum (visible as the flat plateau in the edge-cut between rounds 50 and 400). However, at round 400, the restart mechanism triggers, resetting the temperature to $T = 1$. This immediately relaxes the strict acceptance threshold, allowing the algorithm to resume swapping (as seen in the middle graph) and escape the local minimum. Consequently, the edge-cut drops significantly (from ≈ 6200 down to ≈ 4800). A similar step-wise improvement is observed at the second restart (round 800), confirming that periodic reheating is essential for this custom probability function to progressively find better global optima.

5 Conclusion

The experiments confirm that the JaBeJa algorithm is an effective distributed method for graph partitioning.

- **Task 1** demonstrated that the Hybrid policy outperforms the Local policy by avoiding local optima through random sampling.
- **Task 2** showed that replacing the linear cooling with an exponential schedule and implementing a **Restart mechanism** significantly improves the final solution quality. The restart mechanism allows the algorithm to explore the solution space more thoroughly, escaping local minima that trap the standard version.

The combination of the Metropolis acceptance criterion and the periodic restart proved to be the most robust configuration for minimizing the edge-cut across different graph topologies.