Data Mining - Homework 5

Robert-Andrei Damian and Alice De Schutter

K-way Graph Partitioning Using JaBeJa

December 13, 2021

Task

The goal of this assignment is to understand distributed graph partitioning using gossip-based peer-to-peer techniques, such as, JaBeJa. The assignment is divided into two tasks:

- **Task 1: Implement the Ja-Be-Ja algorithm by modifying the JaBeJa.java class.**
- **Task 2: Tweak different JaBeJa configurations in order to find the smallest edge cuts for the given graphs. Analyze how the performance of the algorithm is affected when different parameters are changed, specially the effect of simulated annealing.**
  1. **Implement a different simulated mechanism described here. Observe how this change affects the rate of convergence.**
  2. **Investigate how the Ja-Be-Ja algorithm behaves when the simulated annealing is restarted after Ja-Be-Ja has converged. Experiment with different parameters and configurations to find lower edge cuts.**
- **BONUS: Define your own acceptance probability function or change the Ja-Be-Ja algorithm (in order to improve its performance) and evaluate how your changes affects the performance of graph partitioning.**

1 Detailed Information

For **Task 1**, the sampleAndSwap(...) method and the findPartner(...) method were implemented as described in A Distributed Algorithm For Large-Scale Graph Partitioning.

For **Task 2.1**, the alternative simulated mechanism was implemented (Figure 1). As described here, the equation used for the acceptance probability is:

$$a = e^{\frac{c_{new} - c_{old}}{T}},$$

where $c_{new} - c_{old}$ is the difference between the new cost and the old one.

Thus, the acceptance probability:

- is always > 1 when the new solution is better than the old one.
- gets smaller as the new solution worsens.
- gets smaller as the temperature decreases.

```java
package se.kth.jabeja.annealing;

import se.kth.jabeja.rand.RandNoGenerator;

class Type2Annealing extends AbstractAnnealing {

    public Type2Annealing(double t, double delta) {
        super(Math.min(t, 1), delta);
    }

    public void update() {
        T = Math.max(minValue(), T * delta);
    }

    @Override
    public boolean shouldAcceptSolution(double currentValue, double potentialValue) {
        double acceptanceThreshold = RandNoGenerator.nextDouble();
        double acceptanceProbability = potentialValue > currentValue ?
                1 : Math.exp((potentialValue - currentValue) / currentTemperature());
        return acceptanceProbability > acceptanceThreshold;
    }

    protected double minValue() {
        return 1e-10;
    }

    @Override
    public String key() {
        return "TYPE2";
    }
}

```

Figure 1: Implementation of different simulated annealing mechanism as described here

For **Task 2.2**, we investigated how the Ja-Be-Ja algorithm behaves when the simulated annealing is restarted after Ja-Be-Ja has converged (Figure 2). We experimented with different parameters and configurations. The parameters that we played with were:

- T_0: Initial temperature.
- δ: Parameter that determines the speed of the cooling process. As described in the paper, the parameter represents a trade-off between **the number of swaps** and the edge-cut (a lower δ is associated with lower edge-cuts but a larger number of swaps).
  Of course, one should aim to minimize edge-cut. However, a number of swaps that is too large means both **a longer convergence time** and more communication overhead. Therefore, the edge-cut is a quality metric for partitioning, while the number of swaps defines the cost of the algorithm.
- α: Parameter of the swapping condition. It is often better to have α be greater than one. A higher alpha does not directly reduce the total edge-cut of the graph, but, in some cases, it increases the probability of future color exchanges. This could lead to a lower edge-cut. Nevertheless, if the parameter is set too high, nodes might overestimate the value of a swap and end up in an inferior energy state.

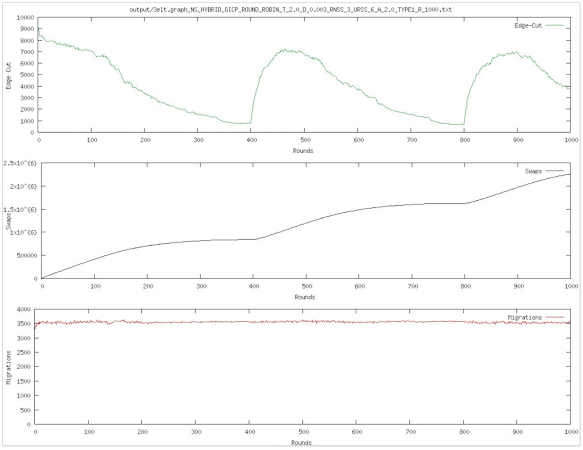

Figure 2: Graph with the annealing restarted every 400 rounds.

We experimented with different parameters and configurations to find lower edge cuts. This was done in Python (tester.py). Results are found in the folder called "output". Figure 3 shows the parameters and configurations that we experimented with.

```
params = [
    {
        "annealing": ["type1"],
        "temp": [1, 2, 3, 6, 10],
        "delta": [0.001, 0.01, 0.03, 0.1],
        "alpha": [0.5, 1, 2, 4]
    },
    {
        "annealing": ["type2"],
        "temp": [0.1, 0.5, 0.6, 0.8, 0.9, 1],
        "delta": [0.5, 0.7, 0.9, 0.99],
        "alpha": [0.5, 1, 2, 4]
    },
    {
        "annealing": ["type4"],
        "temp": [20, 10, 8, 5, 2],
        "delta": [1, 0.5, 0.1, 0.05],
        "alpha": [0.5, 1, 2, 4]
    }
]
```

Figure 3: Parameters and configurations used to experiment and to find lower edge cuts.

Note that the annealing types 1, 2, 3 and 4 correspond to:

- Type 1: original annealing type. Ja-Be-Ja uses a linear function to decrease the temperature and the temperature is multiplied to the cost function.
- Type 2: different simulated annealing mechanism as described here.
- Type 3: no annealing performed (we merely tried this out of curiosity).
- Type 4: our own simulated annealing implementation (for the BONUS).

We created a simple Python program (analyser.py) to find the lowest edge-cut for three different graphs and annealing Types 1, 2 and 4 (Figure 4).

```
{('3elt.graph', 'TYPE1'): (1494,
                           '3elt.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_3.0_D_0.03_RNSS_3_URSS_6_A_2.0_TYPE1_R_1000.txt'),
 ('3elt.graph', 'TYPE2'): (930,
                           '3elt.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_0.8_D_0.99_RNSS_3_URSS_6_A_4.0_TYPE2_R_1000.txt'),
 ('3elt.graph', 'TYPE4'): (909,
                           '3elt.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_2.0_D_0.05_RNSS_3_URSS_6_A_4.0_TYPE4_R_1000.txt'),
 ('add20.graph', 'TYPE1'): (1634,
                            'add20.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_1.0_D_0.001_RNSS_3_URSS_6_A_0.5_TYPE1_R_1000.txt'),
 ('add20.graph', 'TYPE2'): (1952,
                            'add20.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_0.1_D_0.5_RNSS_3_URSS_6_A_0.5_TYPE2_R_1000.txt'),
 ('add20.graph', 'TYPE4'): (1993,
                            'add20.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_10.0_D_1.0_RNSS_3_URSS_6_A_0.5_TYPE4_R_1000.txt'),
 ('facebook.graph', 'TYPE1'): (110230,
                               'facebook.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_3.0_D_0.03_RNSS_3_URSS_6_A_2.0_TYPE1_R_1000.txt'),
 ('facebook.graph', 'TYPE2'): (123321,
                               'facebook.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_0.9_D_0.5_RNSS_3_URSS_6_A_0.5_TYPE2_R_1000.txt'),
 ('facebook.graph', 'TYPE4'): (130530,
                               'facebook.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_20.0_D_0.05_RNSS_3_URSS_6_A_4.0_TYPE4_R_1000.txt')}
```

Figure 4: Output showing parameters that achieve lowest edge-cut for different graphs and different annealing types.

The dictionary is summarized in the table below.

Summary table			
Graphs	**Type 1**	**Type 2**	**Type 4**
3elt	1 494 $T_0 = 3.0$, $\delta = 0.03$, $\alpha = 2.0$	930 $T_0 = 0.8$, $\delta = 0.99$, $\alpha = 4.0$	**909** $T_0 = 2.0$, $\delta = 0.05$, $\alpha = 4.0$
add20	**1 634** $T_0 = 1.0$, $\delta = 0.001$, $\alpha = 0.5$	1 952 $T_0 = 0.1$, $\delta = 0.5$, $\alpha = 0.5$	1 993 $T_0 = 10.0$, $\delta = 1.0$, $\alpha = 0.5$
facebook	**110 230** $T_0 = 3.0$, $\delta = 0.03$, $\alpha = 2.0$	123 321 $T_0 = 0.9$ $\delta = 0.5$, $\alpha = 0.5$	130 530 $T_0 = 20.0$, $\delta = 0.05$, $\alpha = 4.0$

For each graph and each annealing type, the table shows the lowest edge-cut that was achieved, along with the parameters that were used to obtain the result. The lowest edge-cut of each graph is marked with blue/bold text.

From the table, Type 1 is the annealing type which, on average, performs best. However, this statement is based solely on the analysis of three graphs, the conlusion thus remains uncertain.

For the **BONUS**, we defined our own (fairly simple) acceptance probability function. It is shown in Figure 5.

T is defined as the acceptable difference for which a swap is accepted. This acceptable difference drops over the passage of rounds, eventually reaching 0 (and settling at 0). The step of decreasing is defined by the parameter delta.

Although our own simulated annealing implementation is fairly simple and might not always perform better than the other implementations, it is quite efficient. As seen in the previous table, for the 3elt.graph, our implementation (Type 4) performed better than the other two annealing types (Type 1 and 2).

```
class Type4Annealing extends AbstractAnnealing {

    public Type4Annealing(double t, double delta) {
        super(Math.min(1, t), delta);
    }

    @Override
    public void update() {
        T = Math.max(0, T - delta);
    }

    @Override
    public boolean shouldAcceptSolution(double currentValue, double potentialValue) {
        return potentialValue + T >= currentValue;
    }

    @Override
    public String key() {
        return "TYPE4";
    }
}
```

Figure 5: Our own simulated annealing implementation

Furthermore, we also improved the performance of all annealing types by storing the color counts in the neighborhood of each node that we investigate. In that way, the nodes do not have to be investigated more than once. This was implemented with the interface ColorEventListener and by adding a Map<Integer,Integer> in the node class. The run-time was improved ($\approx 3.5\times$ improvement):

Execution time (improved version): 466 072 ms
Execution time (old version): 1 571 729 ms