Homework 5 Report – Graph Partitioning JaBeJa

Data Mining – ID2222 – Homework Group 23

Adriano Mundo (mundo@kth.se), Edoardo Realini (erealini@kth.se)

December 7, 2020

1. Introduction

The goal of the *homework assignment 5* was to implement and test with different parameters setting the distributed graph partitioning algorithm with peer-to-peer techniques, called JaBeJa [1]. We had to implement our own version of the *sampleAndSwap* and the *findPartner* methods in the codebase provided on Canvas. Moreover, we had to implement the *simulated annealing* mechanism described in the article which describes a multiplicative cooling schedule and the related acceptance probability [2]. Indeed, the main objective was to analyse how it affects the performance of the algorithm by tuning different parameters. Another requirement for the assignment was to implement a restart mechanism after the algorithm as converged. For the *optional* task, we designed our own acceptance probability function which improves the performance of JaBeJa.
The datasets used for experimentation were: 3elt, add20, twitter. We preferred the twitter graph over the Facebook graph for computation time reason. The following report explains the implementation, the experimental results and how to run the code for the tasks described above. The implementation has been done by using Java. We leveraged on the knowledge from the paper [1] and the lectures [3].

2. Implementation

The code organization was provided by the codebase, we had to fill with the TODO section with our code.

- ***sampleAndSwap**:* the function sample a node from the graph, find the partner for the swapping phase and finally swap the colours of the two nodes. The code of this function is divided in the three parts, two for node selection and one for swapping.
    - The first two checks the node selection policy. Firstly, it tries the local policy hence by using the *getNeighbors* function retrieves a number of random neighbours and then by using *findPartner* try to find the right partner for swapping among the group of neighbours retrieved from the previous function. If the local policy fails, it uses the random policy by using *getSample*, so it retrieves a uniform sample of neighbours from the graph among whom is found the partner for the swapping.
    - The last part of the function retrieves from the references the information about the colours of the two nodes and then swaps the colours by leveraging on the *getColor* and *setColor*. Finally, it updates the number of swaps.

- ***findPartner**:* the function shows how the partner is selected and returns the selected Node object. We implemented the code described in the paper, which iterates over all the nodes (input to the function), retrieves the information about the colours of the p node and the candidate node (node q). Then, by using *getDegree* calculates the number of neighbours with the specified colour. The method has to calculate the two values needed to compute the energy of the system,

which we called new and old benefit. Then, there are different cases depending on the method the user selects for identifying the best partner and the highest benefit for swapping.
    - Standard Ja-Be-Ja selection which implements the line 26 of the pseudo-code provided in the paper and compares the new benefit with the old benefit and multiplying it for the temperature value.
    - Different simulated annealing mechanism which calculates the acceptance probability with the exponential formula and then it compares the computed value with a random value between 0 and 1 to deviates from local optima solution.
    - Different simulated annealing mechanism with custom acceptance probability function.

- ***saCoolDown**:* the function implements the simulated annealing mechanism in the two cases of linear decreasing described by the paper [1] and the multiplicative case described by the article [2]. In the second case, the initial temperature should be set to 1 and the

The program has been tested by using the three datasets available: 3elt, add20, twitter graphs. Some aspects of the previous functions can be set from the user by using command line arguments. In later sections, we describe in detail how to properly use them.

3. How to Run the Code

In this section it is explained how to run the code provided with this short report. The recommended environment is Ubuntu >= 18.04 LTS. It is also requested to have installed on the machine Java 8 and the *gnuplot* command line graphic utility.

If there are some modifications to the code and before running the algorithm, it is necessary to compile the code

```
$ ./compile.sh
```

To run the algorithm, it is necessary to specify the location of the graph file and use the run command. There are different command line parameters which it is possible to set with this command, the standard one way to substitute the NAMEGRAPH with the name of the file and use following command:

```
$ ./run.sh -graph graphs/NAMEGRAPH.graph
```

The following list is a description of the custom parameters that can be specified by the user with the run.sh command that we added to the CLI options. There exist in the codebase implementation other command line arguments.

- *saActivation:* for simulated annealing activation, default equals to 0.
- *custom*: activation for custom acceptance probability function, default equals to 0.
- *restart*: activation for restart, default equals to 0.
- *restartRounds*: number of rounds for the restart, default equals to 400.

To plot the algorithm, so to obtain a .png file with the plots of swap, edge-cut and migrations, it is mandatory to use the following command, where OUTPUTFILE is the name of the file obtained as output from the algorithm.

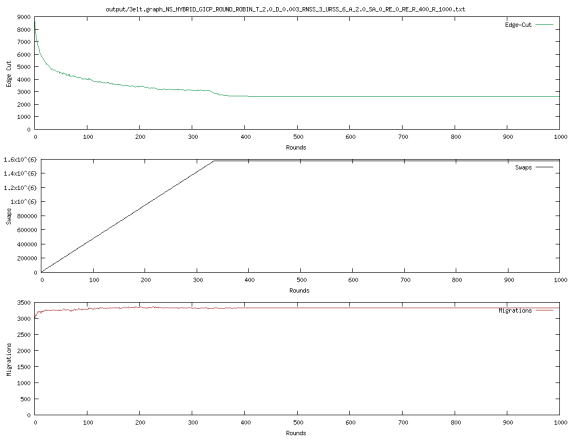
```
$ ./plot.sh output/OUTPUTFILE.txt
```

4. Experimental Results

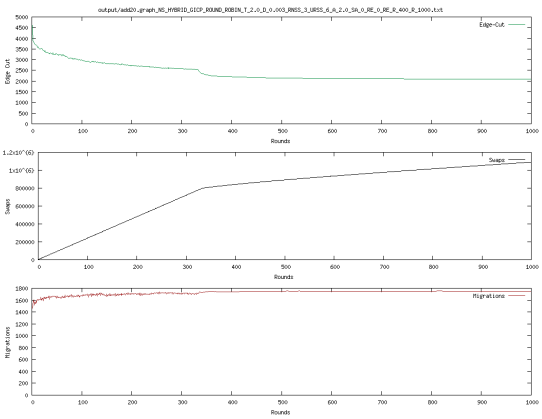
Task 1

In the task 1 we had to implement the two functions for the Ja-Be-Ja algorithm. We show the results for three chosen datasets (3elt, add20, twitter) with the parameters specified through the command line arguments and the *linear cooling* and *hybrid selection policy* as described in the paper. All the task below are run with k = 4 (number of graph partitioning).

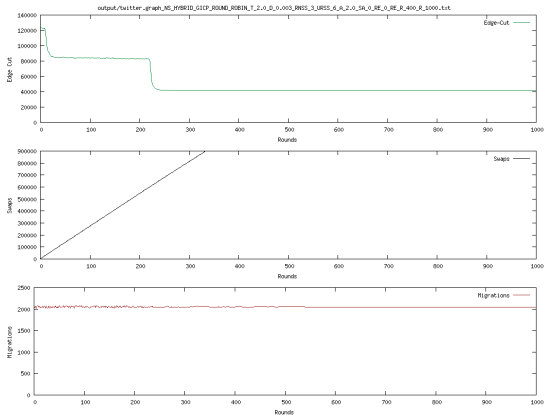
```
$ ./run.sh -graph graphs/NAMEGRAPH.graph -temp 2 -alpha 2 -delta 0.003 -rounds 1000
```



3elt.graph



add20.graph



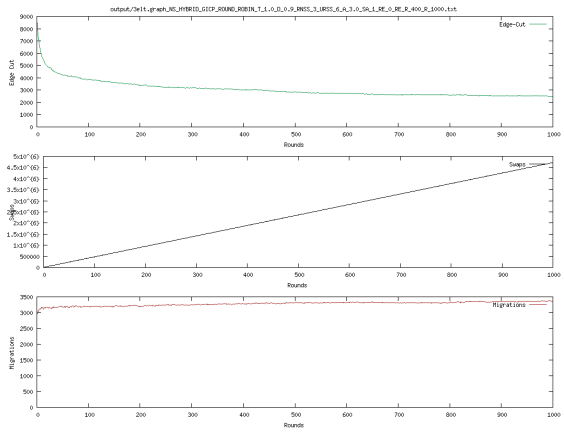
twitter.graph

Task 2.1

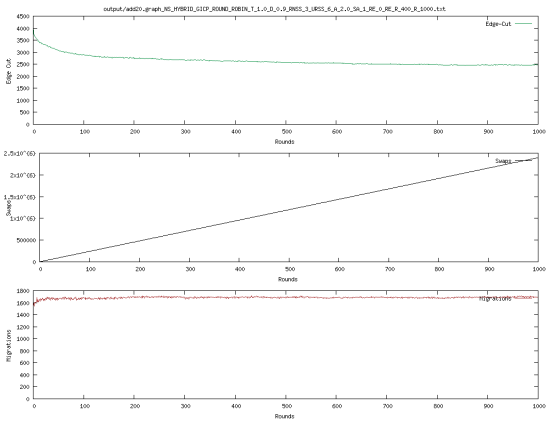
In the task 2.1 we had to implement the simulated annealing mechanism described in the article [2]. We show the results for three chosen datasets (3elt, add20, twitter) with the with the parameters specified through the command line arguments and the *multiplicative cooling schedule* and simulated mechanism with the *exponential acceptance probability*, and a *hybrid selection* policy.

Firstly, we show the standard scenarios obtained with the following command.

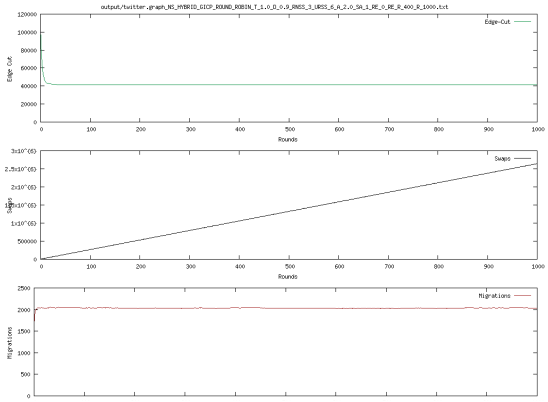
```
$ ./run.sh -graph graphs/NAMEGRAPH.graph -temp 1 -alpha 2 -delta 0.9 -rounds 1000 -
saActivation 1 -graphInitiColorSelectionPolicy ROUND_ROBIN
```



3elt.graph



add20.graph

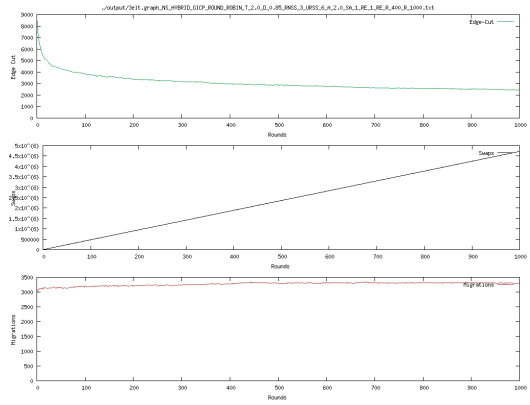


twitter.graph

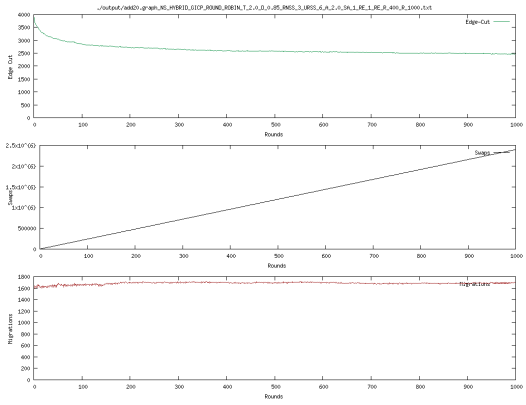
Task 2.2

In the task 2.2 we had to leverage on simulated annealing mechanism implemented in the task 2.1 but in addition we had to implement a restart mechanism for the algorithm after it reaches the convergence. We show the results for three chosen datasets (3elt, add20, twitter) with the with the parameters specified through the command line arguments and the *linear cooling schedule* and the *hybrid selection* policy. Firstly, we will show the base case.

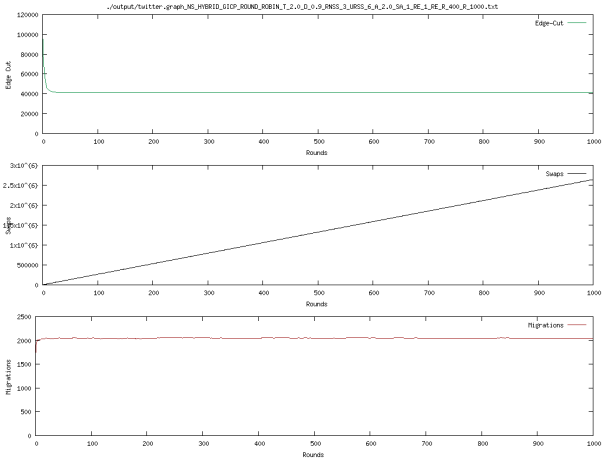
```
$ ./run.sh -graph graphs/NAMEGRAPH.graph -temp 2 -alpha 2 -delta (0.8-0.9) -rounds 1000 -
saActivation 0 -graphInitiColorSelectionPolicy ROUND_ROBIN -restart 1 -restartRounds 400
```



3elt.graph



add20.graph

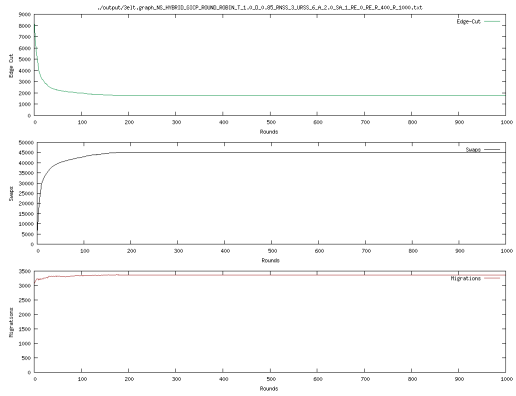


twitter.graph

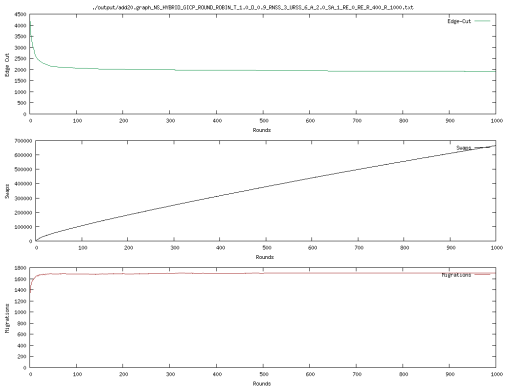
Task Optional

In the optional task we implemented our own acceptance probability function as $e^{\frac{new-old-1}{T}}$.
We show the results for the three chosen datasets (3elt, add20, twitter) with the with the parameters specified through the command line arguments and the *multiplicative cooling schedule* and the *hybrid selection* policy and the custom function.

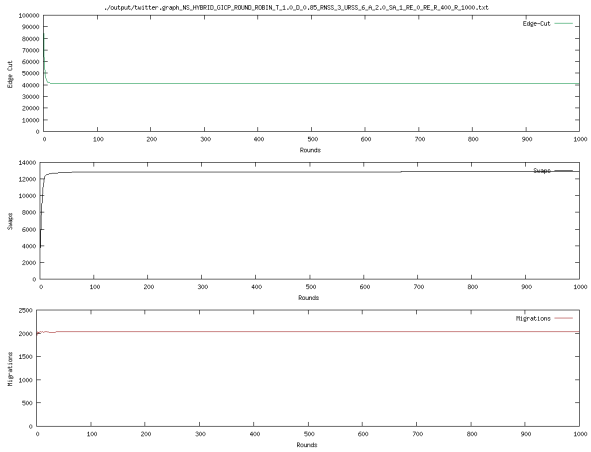
```
$ ./run.sh -graph graphs/NAMEGRAPH.graph -temp 1 -alpha 2 -delta (0.8-0.9) -rounds 1000 -
saActivation 1 custom 1 -graphInitiColorSelectionPolicy ROUND_ROBIN
```



3elt.graph



add20.graph



twitter.graph

Summary

Here we will present, in form of tables, the results obtained with various runs of the algorithm. The task_id can be used to retrieve the standardized parameters that were used. The parameters that were changed the most are specified in the tables.

All the runs were performed automatically using the python script **parameter_search.py**. In this script we implemented a way of trying the algorithm multiple times in sequence, with different parameters and saving the results into files. This helped us to run many tests automatically, without having to enter parameters at each run. In a second moment, another script, **gen_tables_plot.py** uses the results obtained by parameter_search.py in the form of csv files to generate the already presented plots that show the steps of the algorithm and the following summarizing tables.

Table 3elt.graph

	Task	Init_Policy	Delta	EdgeCut	Swaps	Migrations
0	1	BATCH	0.003	381	1578675	121
1	1	ROUND_ROBIN	0.003	2604	1580209	3328
2	2.1	BATCH	0.9670000000000001	422	4719427	130
3	2.1	ROUND_ROBIN	0.8	2392	4714765	3332
4	2.2	BATCH	0.8	411	4719341	142
5	2.2	ROUND_ROBIN	0.85	2439	4715019	3302
6	Opt	BATCH	0.9	368	38084	131
7	Opt	ROUND_ROBIN	0.85	1767	45083	3368

Table add20.graph

	Task	Init_Policy	Delta	EdgeCut	Swaps	Migrations
0	1	BATCH	0.003	2000	1070237	1249
1	1	ROUND_ROBIN	0.003	2095	1090263	1751
2	2.1	BATCH	0.867	2318	2392995	1010
3	2.1	ROUND_ROBIN	0.99	2422	2392985	1719
4	2.2	BATCH	0.9	2277	2392907	971
5	2.2	ROUND_ROBIN	0.85	2473	2393096	1695
6	Opt	BATCH	0.8	2207	476853	1132
7	Opt	ROUND_ROBIN	0.9	1915	664948	1706

Twitter.graph

	Task	Init_Policy	Delta	EdgeCut	Swaps	Migrations
0	1	BATCH	0.003	41260	889032	1463
1	1	ROUND_ROBIN	0.003	41156	899515	2049
2	2.1	BATCH	0.9670000000000001	41288	2636632	1312
3	2.1	ROUND_ROBIN	0.9329999999999999	41121	2636398	2037
4	2.2	BATCH	0.8	41327	2637466	1295
5	2.2	ROUND_ROBIN	0.9	41106	2639352	2045
6	Opt	BATCH	0.85	41317	12312	1284
7	Opt	ROUND_ROBIN	0.85	41148	12868	2031

References

[1] Rahimian, Fatemeh & Payberah, Amir H. & Girdzijauskas, Sarunas & Jelasity, Mark & Haridi, Seif. (2013). JA-BE-JA: A distributed algorithm for balanced graph Partitioning. International Conference on Self-Adaptive and Self-Organizing Systems, SASO. 51-60. 10.1109/SASO.2013.13.

[2] The simulated annealing algorithm, available at: http://katrinaeg.com/simulated-annealing.html (2014), accessed: 7 December 2020

[3] Graph fundamentals, Graph models, Link analysis, Spectral Clustering lectures, ID2222, Data Mining, KTH Royal Institute of Technology