DATA MINING - HOMEWORK 5
K-way Graph Partitioning Using JaBeJa

SERGHEI SOCOLOVSCHI, ANGEL IGARETA

7 December 2020

Introduction

This project aims to study distributed graph partitioning techniques by implementing the JaBeJa algorithm [1]. The problem of balanced graph partitioning is a well-known NP-complete problem with applications in numerous fields such as in Cloud Infrastructure. This algorithm uses local search and simulated annealing techniques and it is massively parallel, which avoids strict synchronization.

The algorithm would be implemented using a scaffolding source code written in Java for simulating it in a one-host-multiple-node model, available in Github. Once the implementation is complete, a hyper tuning will be performed by modifying the parameters that affect the graph partitioning metrics, specially edge-cut. Finally, some modifications of the algorithm would be tested in order to achieve better performance.

Tools

In order to implement the proposed algorithm, the Java programming language was used. Regarding the visualization and analysis of the results, both Gnuplot and Excel were used.

Implementation

The algorithm begins with the execution of the *'startJabeja'* method, which runs the *'sampleAndSwap'* procedure in a loop for the number of specified rounds. As simulated annealing is being used, also after each round, the temperature is updated correspondingly.

During the sample and swap stage, a local search is performed to find the neighbors for the current node, according to the node selection policy. After this, if the best candidate is found, the colors of the nodes among the graph will be swapped.

The search for the best candidate to swap can depend either on the acceptance probability of simulated annealing or by taking the node which maximizes the sum of the node degrees of the graph.

Execution

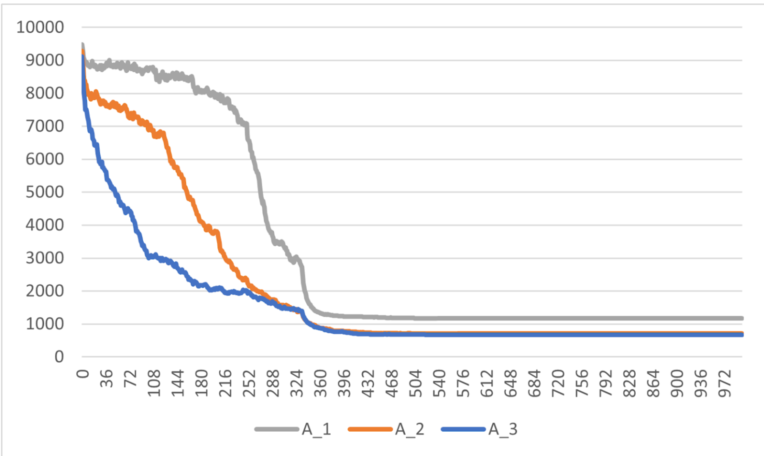
The code can be run using the helper scripts *'compile', 'run -graph <graph>',* and *'plot <output/result.txt>'.* For the second script, some parameters can be passed in order to configure the execution (see the following figure). Note that acceptance and reset parameters have been added from the original code.

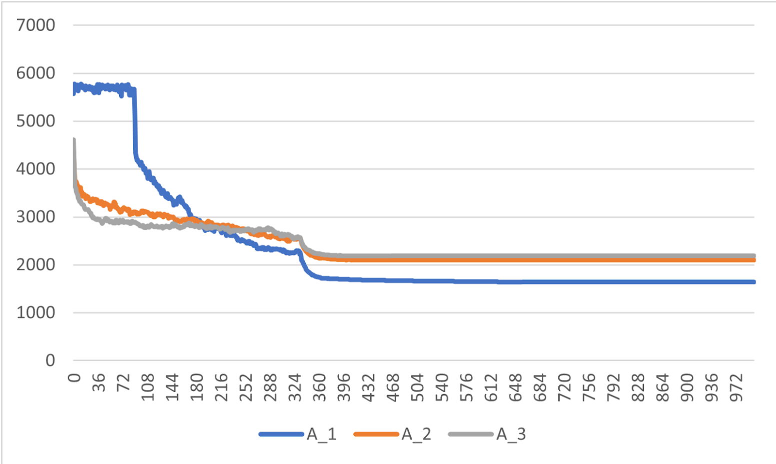
```
-acceptance                        : Use acceptance probability function.
                                     (default: false)
-alpha N                           : Alpah parameter (default: 2.0)
-delta N                           : Simulated annealing delta. (default: 0.01)
-graph VAL                         : Location of the input graph. (default:
                                     ./graphs/ws-250.graph)
-graphInitColorSelectionPolicy VAL : Initial color celection policy.
                                     Supported, RANDOM, ROUND_ROBIN, BATCH
                                     (default: ROUND_ROBIN)
-help                              : Print usages. (default: true)
-nodeSelectionPolicy VAL           : Node selection plicy. Supported, RANDOM,
                                     LOCAL, HYBRID (default: HYBRID)
-numPartitions N                   : Number of partitions. (default: 4)
-outputDir VAL                     : Location of the output file(s) (default:
                                     ./output)
-randNeighborsSampleSize N         : Number of random neighbors sample size.
                                     (default: 3)
-reset N                           : Reset SA each X iterations (no reset: 0).
                                     (default: 0)
-rounds N                          : Number of rounds. (default: 1000)
-seed N                            : Seed. (default: 0)
-temp N                            : Simulated annealing temperature.
                                     (default: 2.0)
-uniformRandSampleSize N           : Uniform random sample size. (default: 6)
```

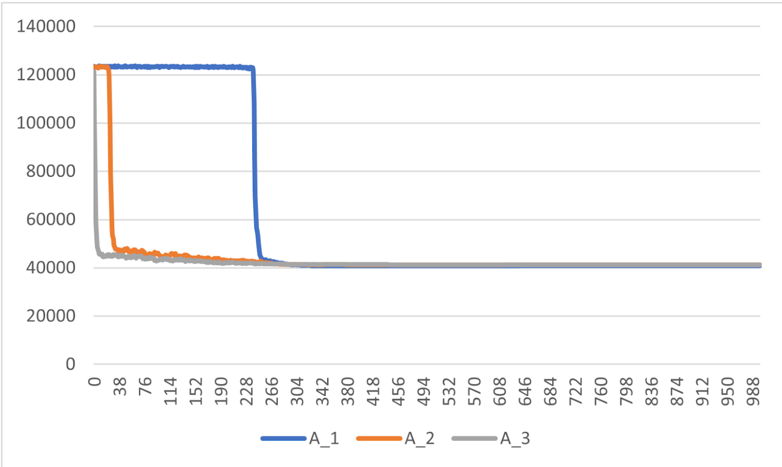
Results

The first experiment consisted of analyzing how the parameter alpha influences the performance of the algorithm. The experiment was conducted by setting delta = 0.003 and using linear simulating annealed technique, described in the paper [1]. As can be seen from the results, the lowest edge-cut was achieved by alpha equal to 1, with the exception of the 3elt dataset.

Dataset	α = 1	α = 2	α = 3
3elt	1172	711	**675**
add20	**1642**	2102	2191
Twitter	**40837**	41312	41339

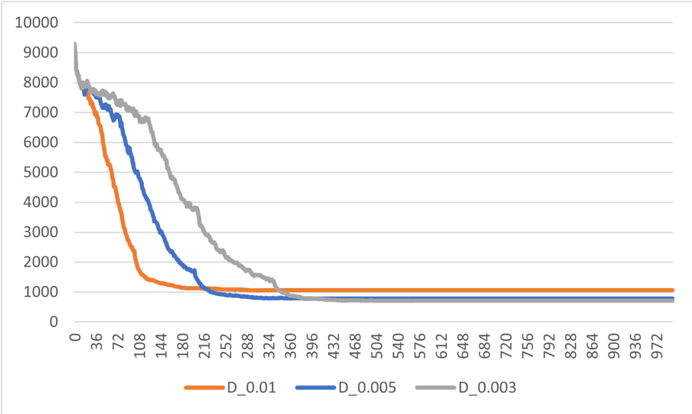


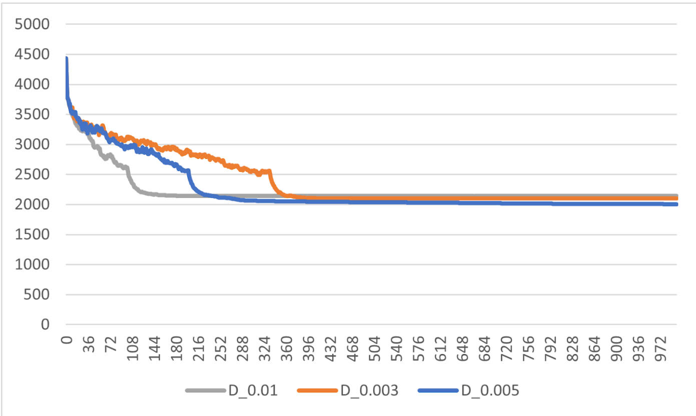


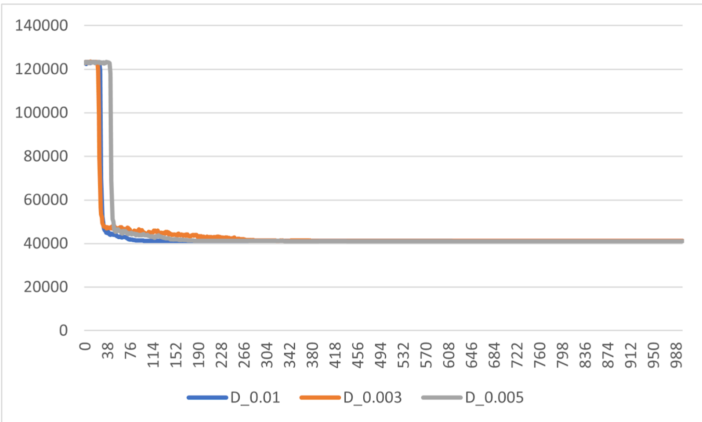


Secondly, the parameter delta was tested for the simulated annealing algorithm used in the paper. The best results were achieved with the parameter set to d = 0.005, except for the 3elt dataset.

Dataset	d = 0.01	d = 0.005	d = 0.003
3elt	1057	783	**711**
add20	2145	**2007**	2102
Twitter	41171	**41132**	41312







Furtherly, the proposed simulated annealing algorithm was tested by resetting the temperature throughout the computation and observed if the edge-cut could be reduced more. As the result, it was proved that by resetting the temperature during the computation it is possible to obtain better results.

Dataset	SA-Reset	SA-No-Reset
3elt	**497**	1057
add20	**1610**	2145
Twitter	**40792**	41171

The alpha parameter was also tested using another version of the simulated annealing algorithm, and from the experiment results, it can be concluded that the alpha = 2 is the best parameter, even if the Twitter dataset achieves the best performance with alpha = 1.

Dataset	α = 1	α = 2	α = 3
3elt	8546	**1567**	2260
add20	2850	**2548**	2616
Twitter	**40829**	41252	41853

The following experiment was studying how different values of the delta parameter influence the performance of the simulated annealing algorithm from Task 2. As a result, it was shown that d = 0.999 achieved the best results with 3elt and add20 datasets, while d = 0.9 was the optimal one for the Twitter one.

Dataset	d = 0.8	d = 0.9	d = 0.999
3elt	1573	1456	**1430**
add20	2458	2463	**2402**
Twitter	41180	**41163**	41351

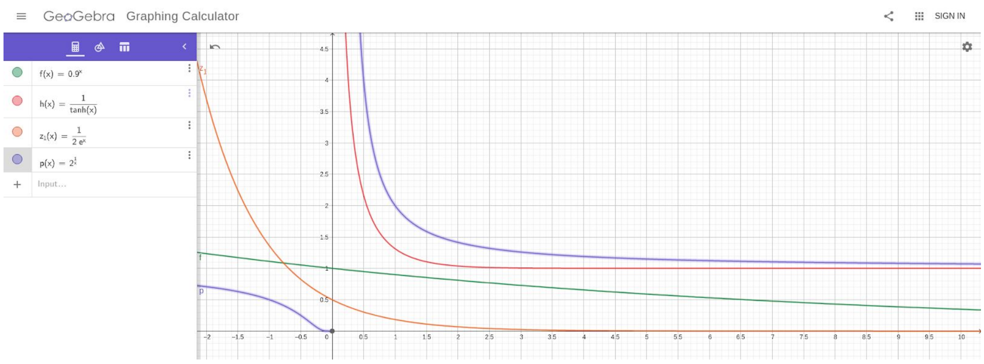
Lastly, the Jabeja algorithm was tested using the simulated annealing algorithm from Task 2 with resetting the temperature through the computation. As can be seen from the table, the algorithm performed better in case the annealing algorithm was reset multiple times.

Dataset	SA-Reset	SA-No-Reset
3elt	**1462**	1880
add20	**2397**	2445
Twitter	**40803**	41809

Modifications

The modifications tested were the following:

1. Avoiding simulated annealing. This modification resulted in fast convergence and poor results.
2. Instead of maximizing the sum of node degrees, take also into account the degree increment compared to the total number of neighbors of the node.
3. Modifying the acceptance probability function to '*newEdgeCut / oldEdgeCut*'.
4. Modify the simulated annealing to accept the solutions every time they are better and use the acceptance probability function when they are worse.
5. Modifying the acceptance probability function to '*Math.pow(2, 1/(round+1))*' and the minimum temperature to *1.0001*, so that it does not follow an exponential distribution as the normal simulated annealing proposes. Also, other functions were tested for the same aim but the results were very similar.

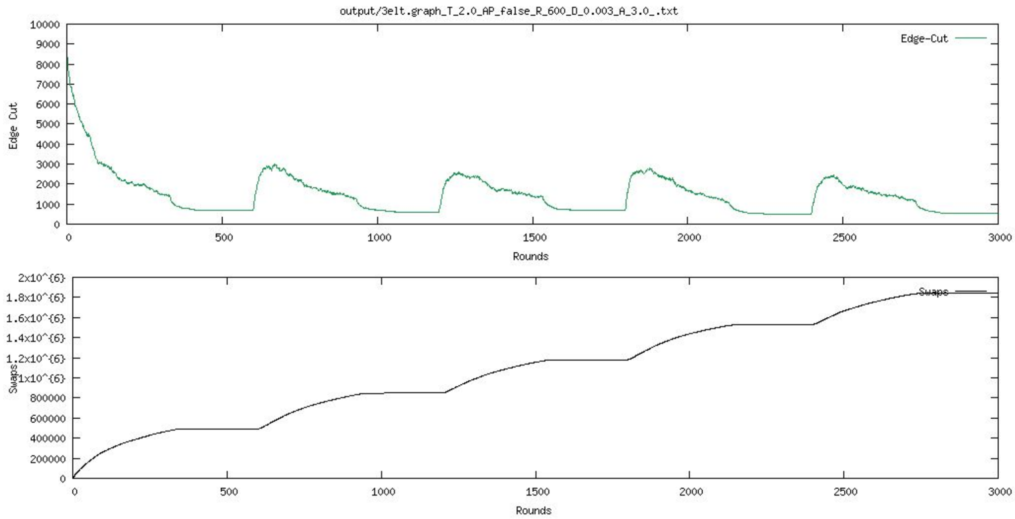


The best results modifying the original solutions were obtained with the 4th modification, the results were the following:

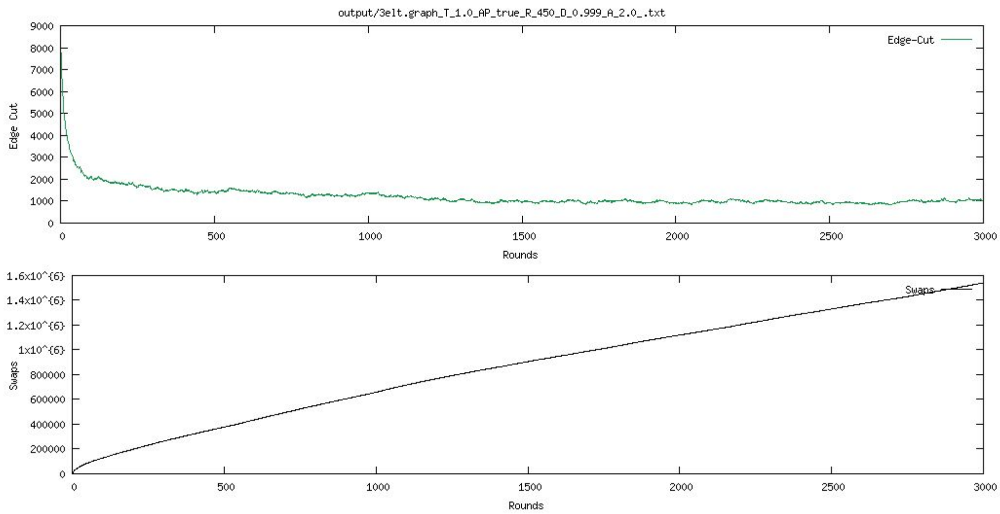
Dataset	SA-Reset	SA-No-Reset
3elt	**823**	1306
add20	**2187**	2207
Twitter	**40941**	40944

Comparision 3elt

The best results obtained with the original solution regarding the time to converge, edge cut, and the number of swaps was:



On the other side, the best results obtained with the modification using simulated annealing presented in the previous section were:



Regarding the edge-cut, the original solution obtained 497 as the best result compared to 823 in the modification. In addition, the number of swaps is similar.

References

[1] Rahimian, Fatemeh, et al. "Ja-be-ja: A distributed algorithm for balanced graph partitioning." *2013 IEEE 7th International Conference on Self-Adaptive and Self-Organizing Systems*. IEEE, 2013.