Data Mining, ID2222

Short report on lab assignment 5

Homework 5: K-way Graph Partitioning Using JaBeJa

Ivan Klabucar and Simon Langrieger

December 13, 2021

1 Short Introduction

In this report we look at JaBeJa - a distributed algorithm for graph partitioning. In particular, we investigate the influence of different approaches to perform simulated annealing during a run of the algorithm.

2 JaBeJa

First a high-level explanation of the algorithm itself. JaBeJa is a k-way partitioning algorithm where the number of partitions k and their respective sizes are chosen at the beginning of a run. Usually one assumes equal-sized partitions. Nevertheless, the sizes of communities can be adapted when necessary if the user has prior knowledge about the graph.

On start-up every node is assigned a (according to the partitioning) random color. All nodes with the same color will in the end belong to the same partition. The quality of a partitioning is therefore measured in the number of edges between nodes of different colors which are called edge-cuts.

The idea of JaBeJa is to optimise the edge-cuts locally by iteratively swapping the color of two nodes if the number of edge-cuts decreases after the swap. Formally defined, the rule is the following $d_p(C_p) + d_q(C_q) > d_p(C_q) + d_q(C_p)$ with C_p the color of node p and $d_p(C)$ the number of nodes connected to node p which don't have the color C (hence $d_p(C_p)$ is the number of edge-cuts around node p).

There are 3 different strategies to choose the nodes for a possible swap

1. Local: Only neighboring nodes are allowed to swap

2. Random: Every node selects another random node in the whole graph for swapping

3. Hybrid: Combination of Local and Hybrid. First the local nodes are tested if a swap would improve the edge-cuts. If this fails a random node in the graph is selected.

This node selection strategy allows the distributed fashion. Every computing node (not graph node) only has to know a small subset of neighboring nodes and some random nodes in the graph to work.

We modify the swapping rule a bit to make it more powerful. First we introduce a parameter $\alpha \geq 1$ which we add the following way: $d_p(C_p)^\alpha + d_q(C_q)^\alpha > d_p(C_q)^\alpha + d_q(C_p)^\alpha$. Bigger alphas favor an even distribution of edge-cuts. Easy example: two nodes have $1 + 3$ edge cuts and after the swap they would have $2 + 2$. Therefore the vanilla rule would not indicate to change. With $\alpha = 2$ the inequality would change to $1 + 9 > 4 + 4$ and therefore the colors will be swapped. For our experiments we always used $\alpha = 2.0$

3 Simulated Annealing

The second and for this report more important change is Simulated Annealing with the Temperature parameter T. The rule finally changes to

$$(d_p(C_p)^\alpha + d_q(C_q)^\alpha) \cdot T > d_p(C_q)^\alpha + d_q(C_p)^\alpha$$

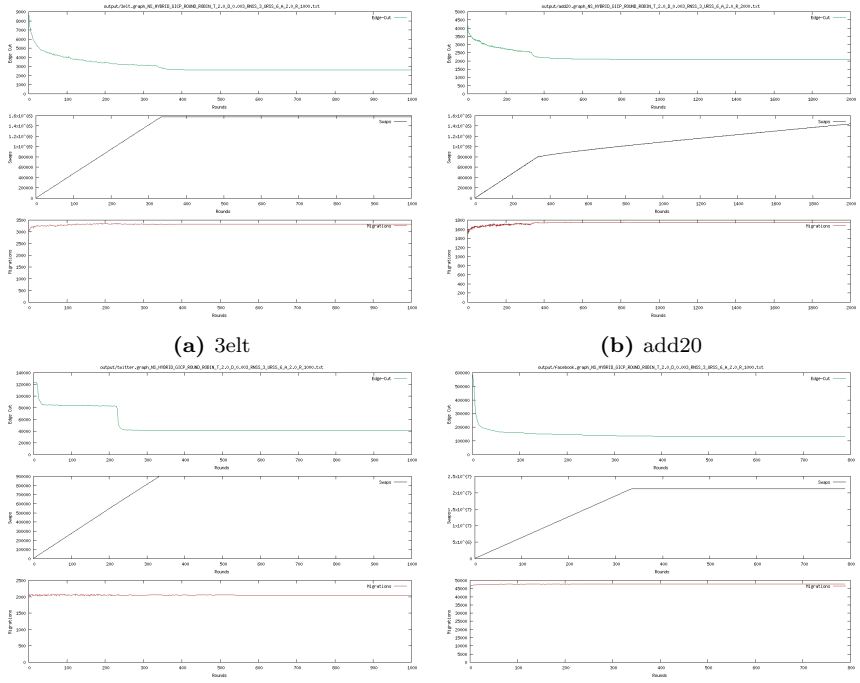
The problem with the vanilla rule is that it tends to solutions which are only local optima. With Simulated Annealing we temporarily might end up in worse configurations which however can finish in better solutions. The influence of Annealing gets reduced over time so at some point the algorithm runs as there is no parameter T at all.

3.1 Linear decrease of T

The first task was to implement JaBeJa with a very simple way for simulated annealing: we start with some starting Temperature (in our case $T = 2.0$) and then linearly decrease it by repeatedly subtracting a fixed delta until T reaches the value 1.0. This happened after around 320 steps in our case. We ran JaBeJa on four graphs: 2 smaller ones 3elt and add20 and two bigger examples from twitter and facebook. The results can be seen in Figure 1.

For every graph we plot 3 metrics:

1. edge-cuts, our quality measure



(a) 3elt **(b)** add20

(c) twitter **(d)** facebook

Figure 1: Results with linearly decreasing Temperature

2. swaps, metric which shows "movement" and also cost of the algorithm
3. migrations, the number of nodes that be migrated from their initial partition to their final partition

We will however only focus on the first two - so edge-cuts and swaps.

In all 4 runs we see very similar results. After the aforementioned ~ 320 steps either the algorithm stops swapping or slows down (add20). By this time the algorithm converged anyway, hence in the remaining steps nothing much happens.

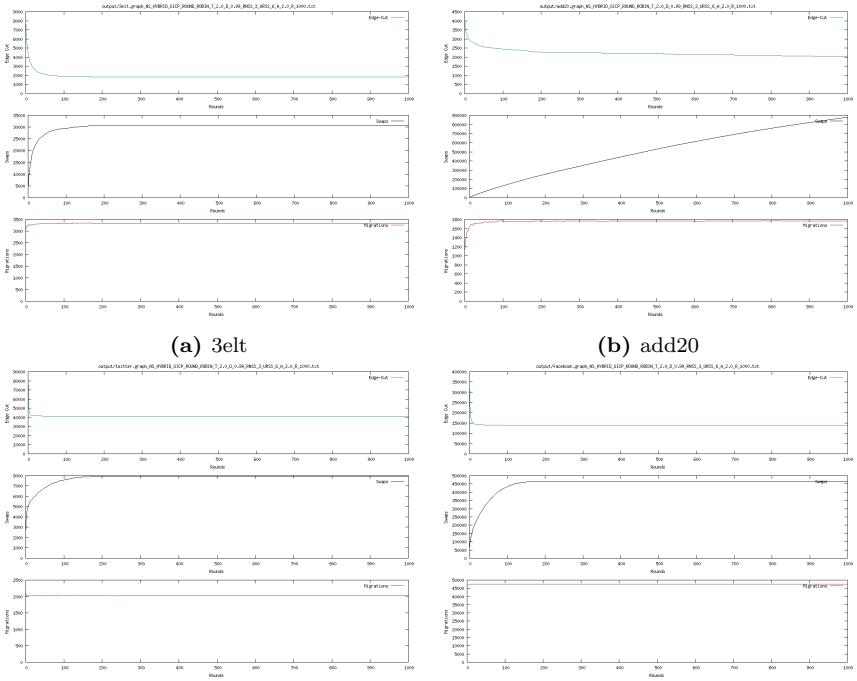
In the end we have ~ 2600 for 3elt, ~ 2100 for add20, ~ 41200 for twitter and ~ 134200 edge-cuts for facebook.

3.2 New approach to Simulated Annealing: Acceptance Probability

Now we change our implementation of Simulated Annealing. We remove the T from the swapping rule and formulate a new rule: if the edge-cuts get better then we definitely swap, if it gets worse we still might swap depending on a

probability a which we define as $a = e^{\frac{c_{new} - c_{old}}{T}}$ with c being the sum of the number of neighboring nodes of p and q with the same color as p and respectively q. The exponent is always negative as the quality of the swap is bad, hence we get a value between 0 and 1 which we interpret as a swapping-probability. Looking at the formula, we see that a high T increases the probability of still swapping while it is almost impossible to still swap with a T close to zero. In our experiments we start with $T = 1.0$ and decrease by repeatedly multiplying T with 0.99 until we reach a min-value which we set at 10e-5. This happens in around 1000 steps.

The results can be seen in Figure 2.



(a) 3elt **(b)** add20

(c) twitter **(d)** facebook

Figure 2: Results with Acceptance probability

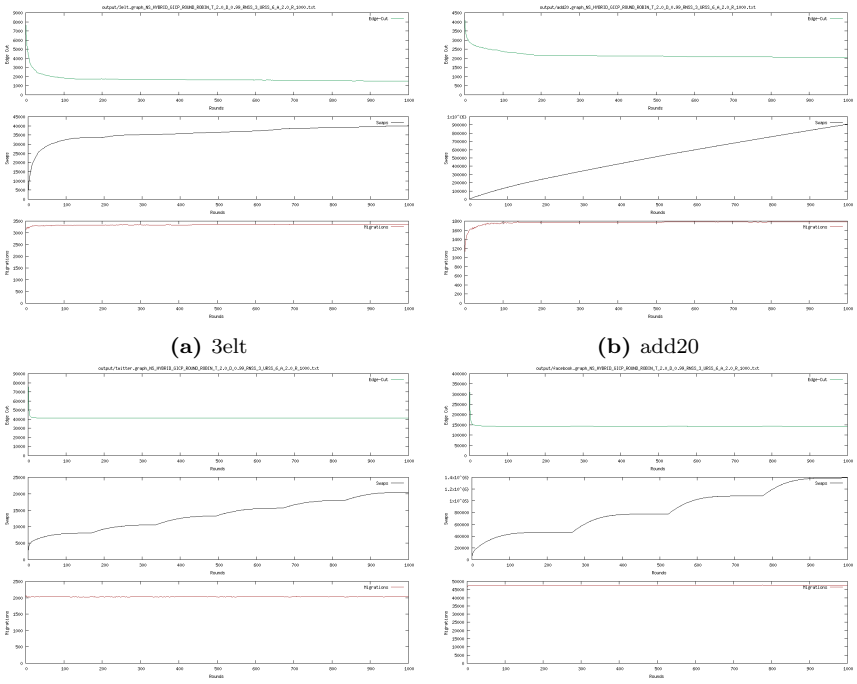
What we see is that now that the number of swaps also behaves exponential, just as T now changes exponentially. We again see that in the add20 case JaBeJa does not appear to really converge when we only look at the number of swaps while the edge-cut stays more or less constant.

Finally we ended up with ~ 1700 for 3elt, ~ 2000 for add20, ~ 41300 for twitter and ~ 137200 edge-cuts for facebook. Hence we see an improvement for the two smaller graphs while we see that the result got slightly worse for twitter and facebook.

3.3 Restarting Annealing

Now we don't reformulate the Annealing rule itself but if we see that there is no movement in the algorithm (which we recognize if the total number of swaps stops changing for some iterations) we reset the Temperature back to 1.0.

The results can be seen in Figure 4.



(a) 3elt **(b)** add20

(c) twitter **(d)** facebook

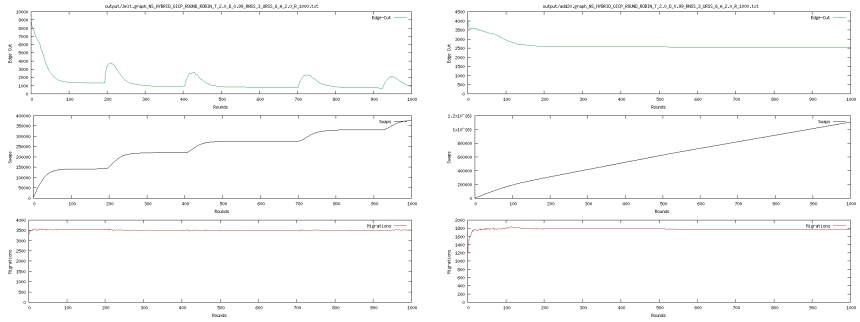
Figure 3: Results with Acceptance probability and resetting Annealing

We see in the number of swaps the effect of the reset - after an reset there are a lot of new swaps before stagnating again. It so-to-say gives the algorithm a push. We also see small spikes in the number of edge-cuts. In the short-term the quality of the partitioning worsens a little bit as the swapping probability is high again. However, this can lead to a new local optima which might be better than the previous one.

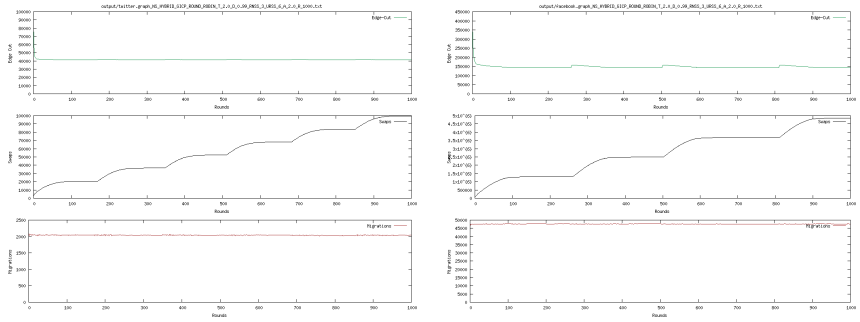
The final results are ~ 1500 for 3elt, ~ 2000 for add20, ~ 41100 for twitter and ~ 140000 edge-cuts for facebook. Hence, it can further improve the quality but it does not have to - as seen in the facebook example. This result is expected as the reset is quite random and can end up in a local optima which is actually worse than before.

3.4 Bonus: Changing the Acceptance rule

As a bonus task we were asked to add a modification to the acceptance rule. What we found is that the reset leads to a push - nevertheless often not a big enough one. Therefore we add a constant term C to the exponent which increases the probability of swapping - independent of the score difference. Hence the probability becomes $a = e^{C+\frac{c_{new}-c_{old}}{T}}$. We also decrease the value of C over time, just like the temperature, but do it in a linear fashion such that the influence of C is gone after 100 steps.



(a) 3elt **(b)** add20



(c) twitter **(d)** facebook

Figure 4: Results with Acceptance probability and resetting Annealing and independent increase of swapping probability

Our change has the expected effect, especially visible in the 3elt graph. In add20 we don't see any effect as the number of swaps never stagnates and the annealing therefore never resets. In the bigger graphs we don't see such a huge effect, however the biggest change is the "movement" in the algorithm. After 1000 steps we have four times as many swaps as in the run without adding the constant (twitter). This makes sense as the constant is independent of the selected nodes and is in the beginning big enough to always force a swap. A lot of movement is however not necessarily better as it also needs more operations (longer run-time) but more configurations are explored which can result in a better result. Since we introduce a lot of randomness it might be beneficial to keep track of the best configurations as we are more likely to loose good ones.

In our run we got $\sim$ 700 for 3elt, $\sim$ 2500 for add20, $\sim$ 41400 for twitter and

$\sim$ 143000 edge-cuts for facebook. We got a pretty big improvement in the 3elt graph but worsened the results in the other graphs. This is probably due to the initial increase in randomness. However at this point we have introduced a lot of parameters which influence the quality of the outcome - so other hyperparameters might work better for some graphs.