Data mining Homework 5: JA-BE-JA

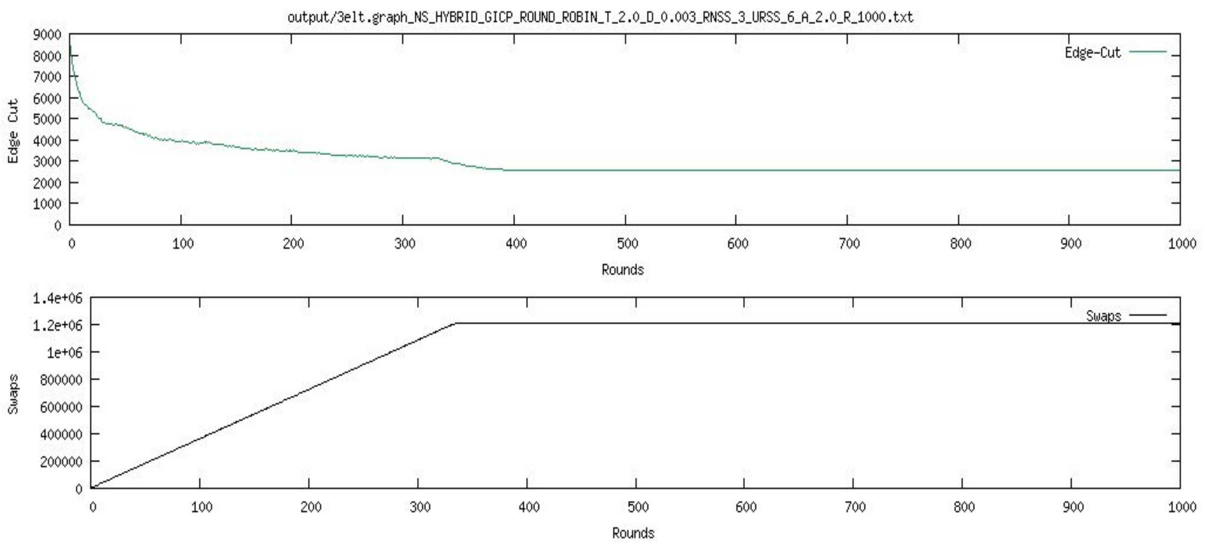
Braulio Grana Gutiérrez & Adrián Ramírez del Río

For this assignment we implemented part of the JA-BE-JA algorithm using Java and tested its results on different graphs when tweaking parameters. All the code can be found at github repository: https://github.com/AdrianRamirezRio/id2222

Algorithm performance

In this section we will discuss the JA-BE-JA's performance with different parameters, acceptance probability functions and simulated annealing algorithms. All results shown in this section are from the 3elt graph.

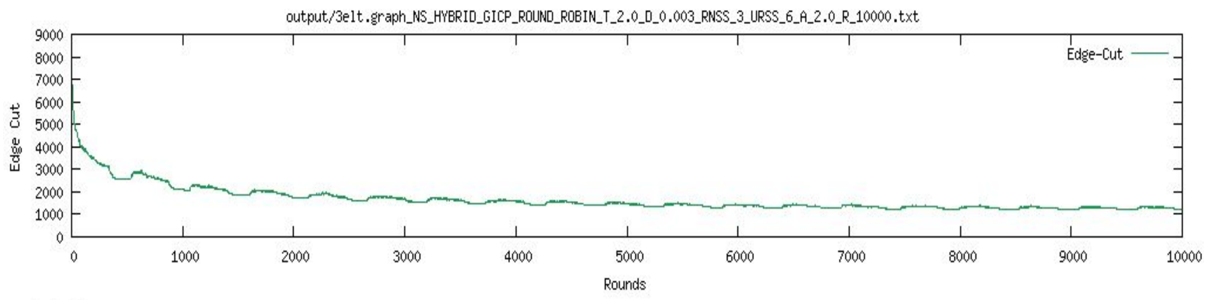
The first and simplest configuration we used a linear simulated annealing algorithm with the acceptance linear function described in the paper: $(new * T > old)\ \&\ (new\ >\ highest)$. Other parameters in this test are 0.003 delta, 2 starting temperature and 1000 rounds. In the following figure we can see the result:



We can see the algorithm converges around round 350 and the edge-cut descends steadily. But to really measure how it has performed with these parameters we will compare it against other configurations.

For the second part of the assignment we restarted the temperature of the algorithm when it hit 1. For this, we tried 3 different ways: first, we restarted T immediately after it reached 1, but this proved to be troublesome as the algorithm wouldn't have time to converge after having reached the minimum temperature; then we tried delaying the restart for a fixed amount of rounds to give the algorithm time to converge, but this wasted many rounds on a repeated value; finally, we delayed the restart by looking at the last 10 values of the edge-cut, if they were the same, we assumed the algorithm had converged and restarted.

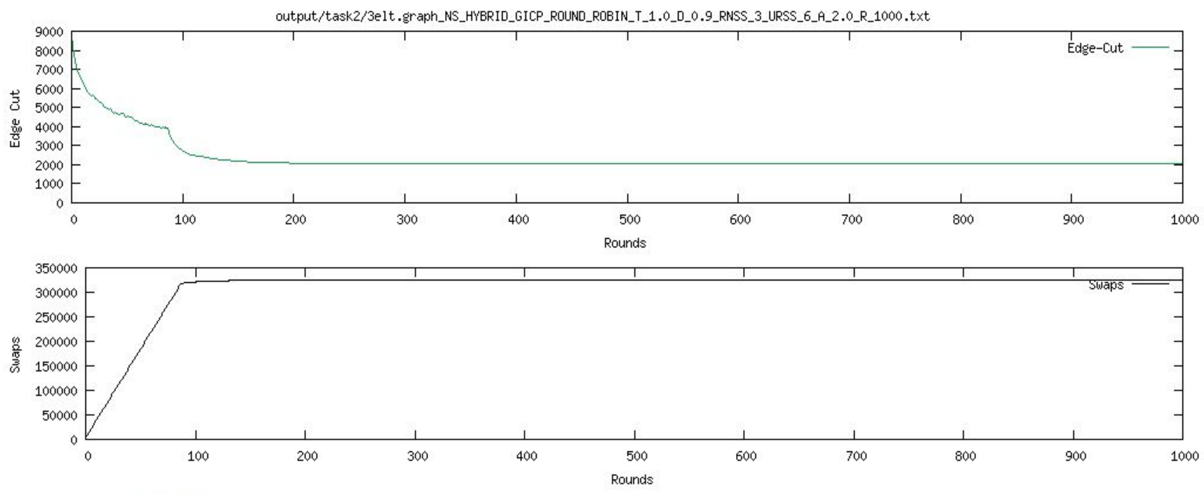
This technique does not show any improvement when tested with 1000 rounds. However, it allows the algorithm to keep improving if more rounds are given which wasn't possible without the T restarts. Here we can see a figure with the edge-cut evolution over 10000 rounds using T restarts, 0.003 delta and 2 as starting temperature:



As it can be seen, there are small growths of the edge-cut every time T is restarted but it keeps decreasing in the long term (final edge_cut is around 1000).

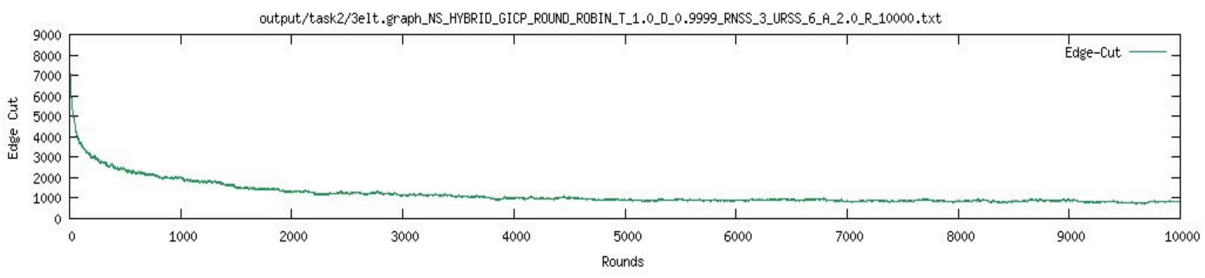
Lastly, and also for the second task, we had to change JA-BE-JA to use a different approach for the simulated annealing that would decrease T exponentially ($T_{k+1} = T_k * \delta$) instead of linearly and use a different acceptance probability function: $e^{\frac{Enew - Eold}{T}}$ (where E stands for energy).

For this part we always use 1 as starting temperature (as the maximum allowed) and, in this case, a delta of 0.9 so the algorithm has enough flexibility to search the solutions space but



not too much so that it takes too long to converge. The following figure shows the result:

As you can see the convergence rate of the algorithm is faster and the results are slightly improved. One good thing we saw with this method is that increasing the delta parameter to near 1 values (allowing a lot of space search) makes the edge-cut drop but it also takes



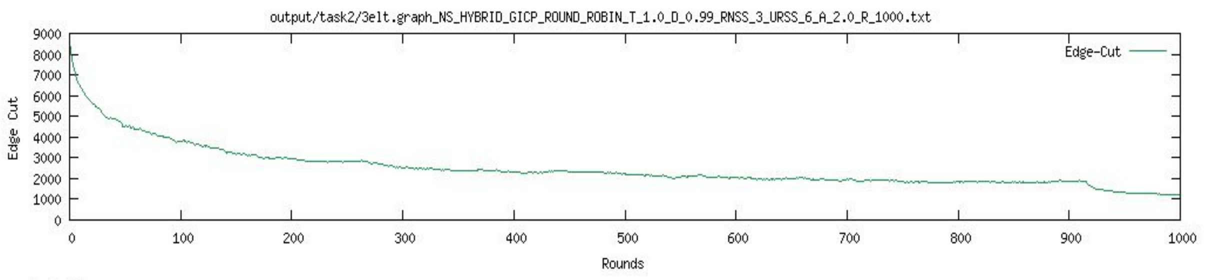
more time to converge as you can see in the figure of the edge-cut evolution using same method but with delta 0.9999 and max rounds 10000.

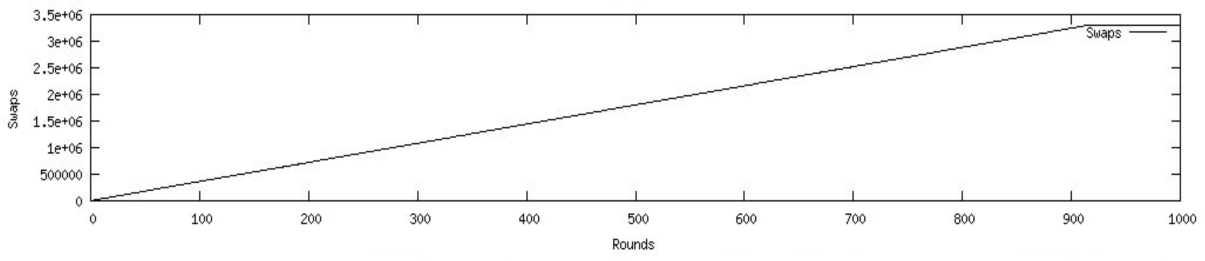
Graphs analysis

In this section we will analyze 4 graphs in terms of time to converge, number of swaps and minimum edge-cut observed with the methods aforementioned. All tests in all graphs were executed with the same number of partitions (4) as we only wanted to compare the results of different parameter values.

3elt graph

This graph was already analyzed in the previous section and the best combination of parameters we found was using exponential simulated annealing (SA) algorithm and acceptance probability (AP) function and parameters 0.99 delta and 1000 rounds.

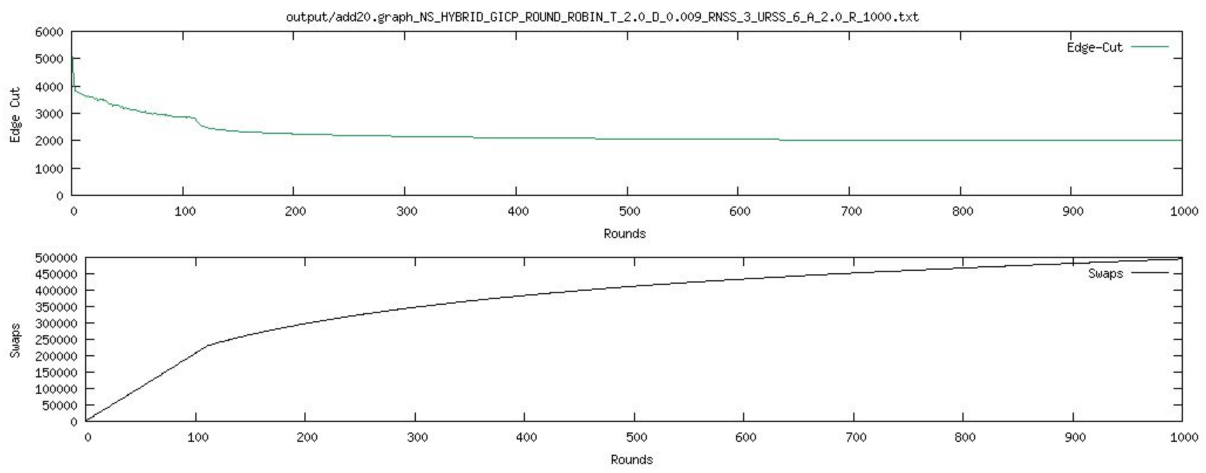




With these parameters we observed a convergence time of about 980 rounds and a minimum edge-cut of 1100. Other parameter combinations granted better minimum edge-cut but at the cost of a higher amount of swaps and much more rounds (edge cut of 800 with 10000 rounds).

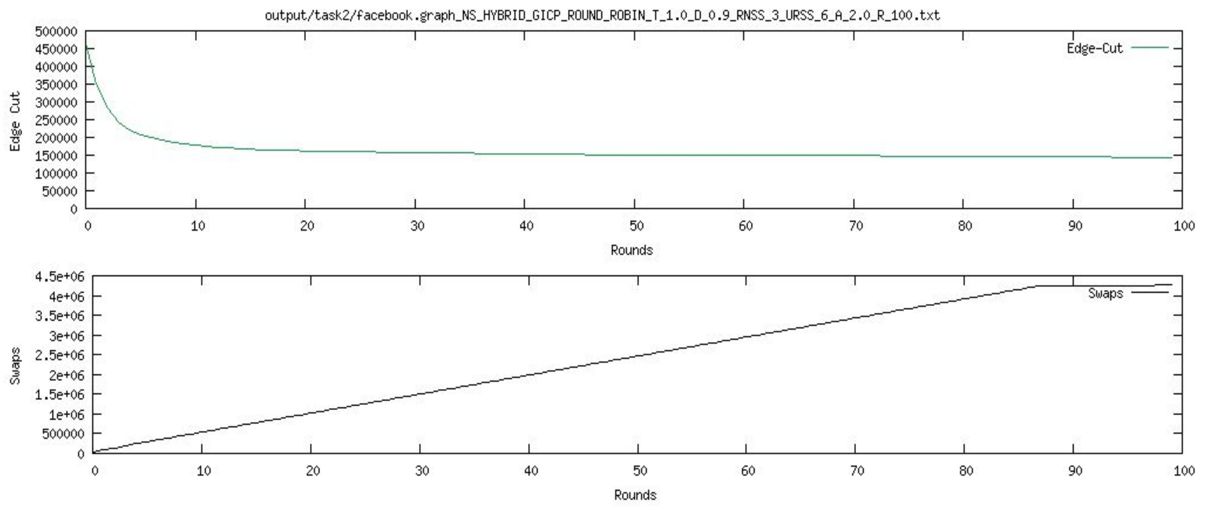
add20 graph

The add20 graph is the only one of the 4 that showed better results (if only slightly better) using a linear SA and the paper's AP function. For this, we used the parameters 0.009 delta and 1000 rounds. The minimum observed edge-cut was around 2000.



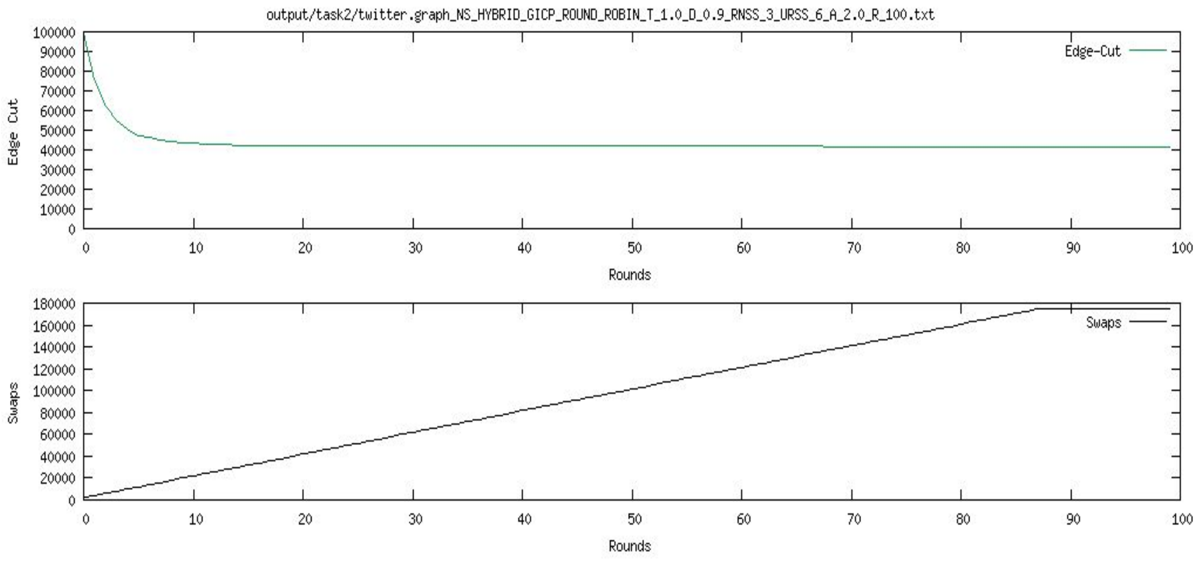
Facebook graph

This is the biggest graph of the 4 and, therefore, we could not test it with a high amount of rounds as it would take several hours to complete a single test. The best result for this graph was using the exponential SA algorithm and the second AP function as well as parameters 0.9 delta and 100 rounds. The reason for such a slow amount of rounds is that when we stretched it to 1000 and 10000 it didn't show much improvement but required a lot more time to converge and a bigger number of swaps. The minimum edge-cut found was around 143000.



Twitter graph

Lastly, the twitter graph had a similar behavior to the facebook graph in the sense that no configuration managed to improve the results of the others in terms of minimum edge-cut so we went for the configuration that showed the least amount of swap and the lowest convergence time. For this we used the exponential SA and the second AP function and parameters 0.9 delta and 100 rounds. The minimum edge-cut observed was around 41000.



Optional work

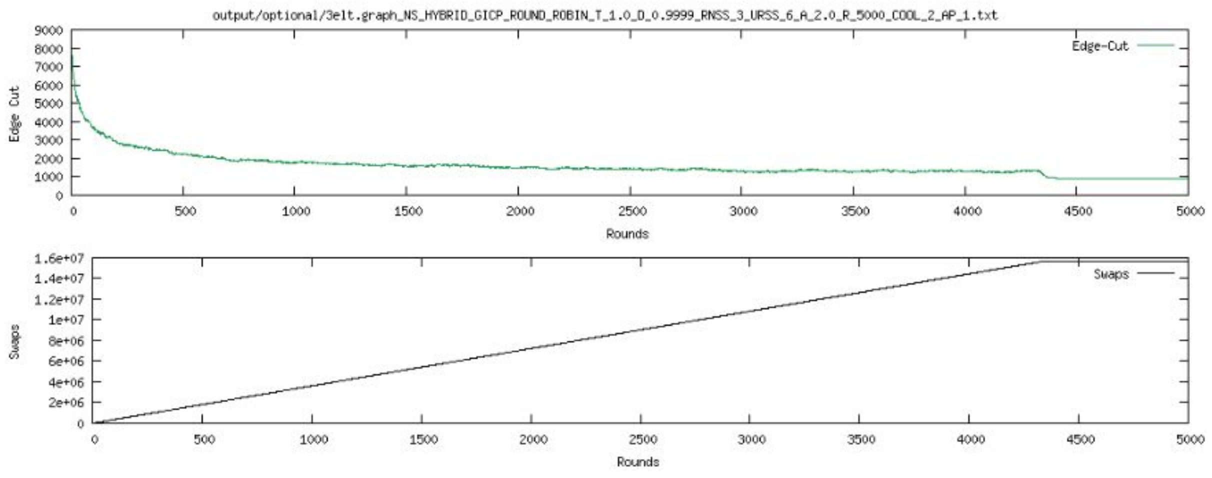
As optional work, we decided to try out a different acceptance probability that we found to be used very widely by the community: $\frac{1}{1 + e^{\frac{Eold - Enew}{T}}}$

This acceptance function has the peculiarity that when Eold > Enew (the considered solution yields to a worse state) the maximum acceptance probability for the new solution is at most 0.5, so there are considerably less jumps to worse solutions than there were with the previous acceptance functions.

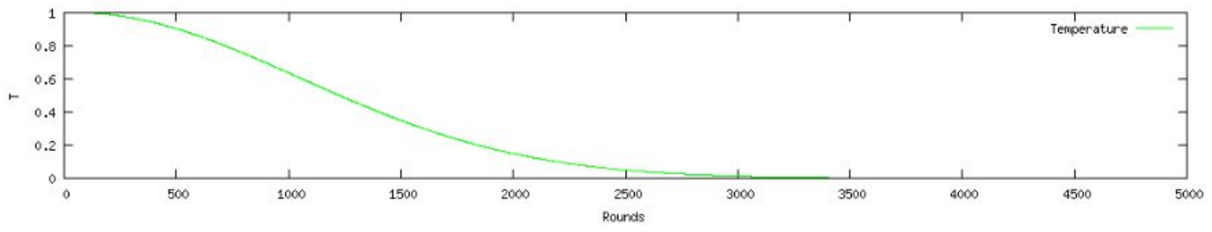
We also tried new cooling functions, motivated by those used in neural networks to update the learning rate, the exponential decay: $T_{k+1} = T_k * \delta^{k/100}$ and the inverse time decay: $T_{k+1} = \frac{T_k}{1 + \delta^k}$.

The motivation behind the inverse time decay is to try to boost the cooling in comparison to the previously applied exponential decay, so that we can try parameters to allow more solution space search whilst keeping the convergence times. However, we found the algorithm to reach equilibrium too fast (even for delta values as low as 0.05) so the algorithm didn't have enough time to search the solution space.

On the other hand, the exponential decay seems to achieve exactly the behaviour we wanted since the temperature decays faster in the middle, as can be seen in the figure below where we repeated the experiment on 3elt graph with delta 0.9999 but changing the cooling to the exponential decay approach:

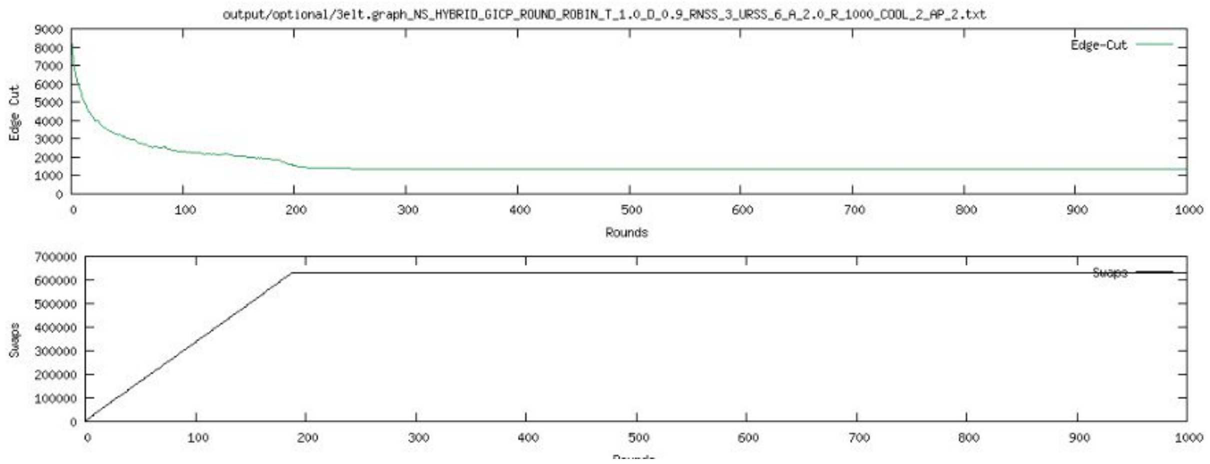


As we see, the convergence time falls from nearly 10000 rounds (shown in the first section) to roughly 4250 and an edge cut value of 895. But this improvement does not comes free, it can be seen that the number of swaps presented here is much higher than in previous examples. This can be explained if we look at the evolution of the temperature shown below:



We see that at the beginning of the algorithm, the cooling is very soft, so we are allowing the algorithm to make a lot of bad movements, that in the long run compensate with better results but with a lot more swaps.

Finally, we found that combining the exponential decay with the new acceptance probability (which limits the number of bad jumps since the maximum value is 0.5) yields the best results in terms of time convergence and edge cut as shown below for initial temperature 1 and delta 0.9:



In just 250 rounds the algorithm achieves an edge cut value of 1360, though the number of swaps is still very high (one order of magnitude) compared to previous experiments.

Note that the exponential decay cooling adds an extra parameter (the magnitude that divides k) which we set to 100. If one was to run the algorithm for a different graph with less iterations (as we did for facebook/twitter graph in previous sections), this value should also be changed.