Data Mining Homework 5

K-way Graph Partitioning Using JaBeJa

Group 22 - Blanca Bastardés Climent, Alice Ciallella

Short description

The implementation of this assignment has been done in java. All the code is provided in the script **Jabeja.java** and the script **CLI.java** contains the hyperparameters used. The approach followed in this assignment consists in understanding distributed graph partitioning by means of implementing the Ja-Be-Ja algorithm.

Instructions to run the code

The following folders and files have the relevant information with the implementation of the code.

- **/src/main/java/se/kth/jabeja/Jabeja.java:** A javascript containing code for JaBeJa algorithm and the different tasks assigned.

Results

The results are divided into two sections. The three datasets studied are 3elt, add20 and Twitter.

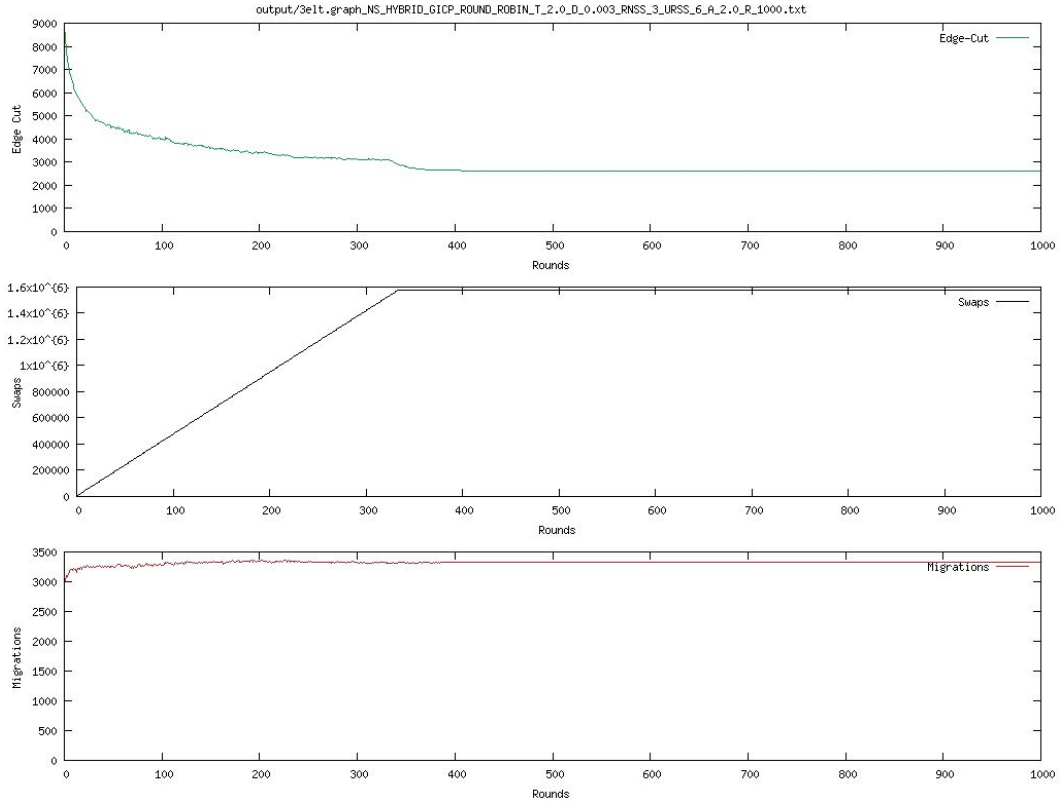
Task 1

In this task, the basic code of the algorithm is implemented following the one in the paper. Results for the three datasets with the default configuration can be seen in the next table and figures.

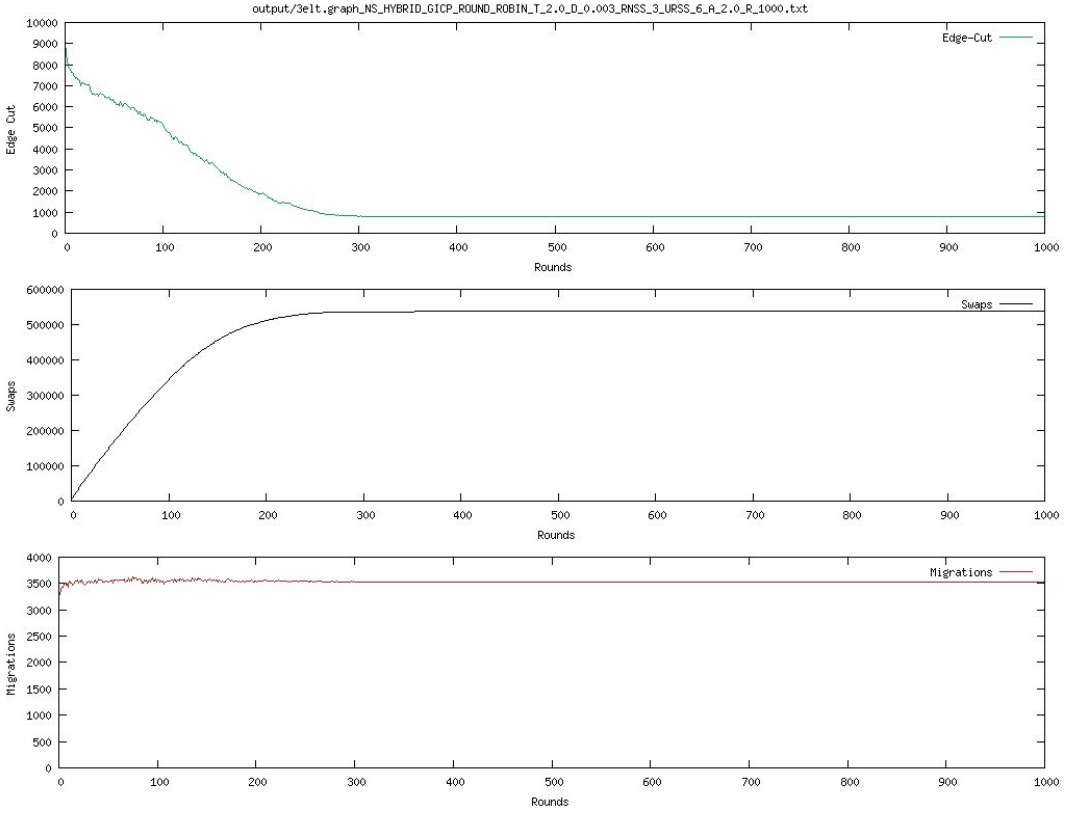
data	T	delta	rounds	edge-cut	swaps	migrations	convergence time
3elt	2	0.03	1000	2604	1580209	3328	472
add20	2	0.03	1000	2095	1090263	1751	827
twitter	2	0.03	1000	41156	899515	2049	813

With condition new != old:

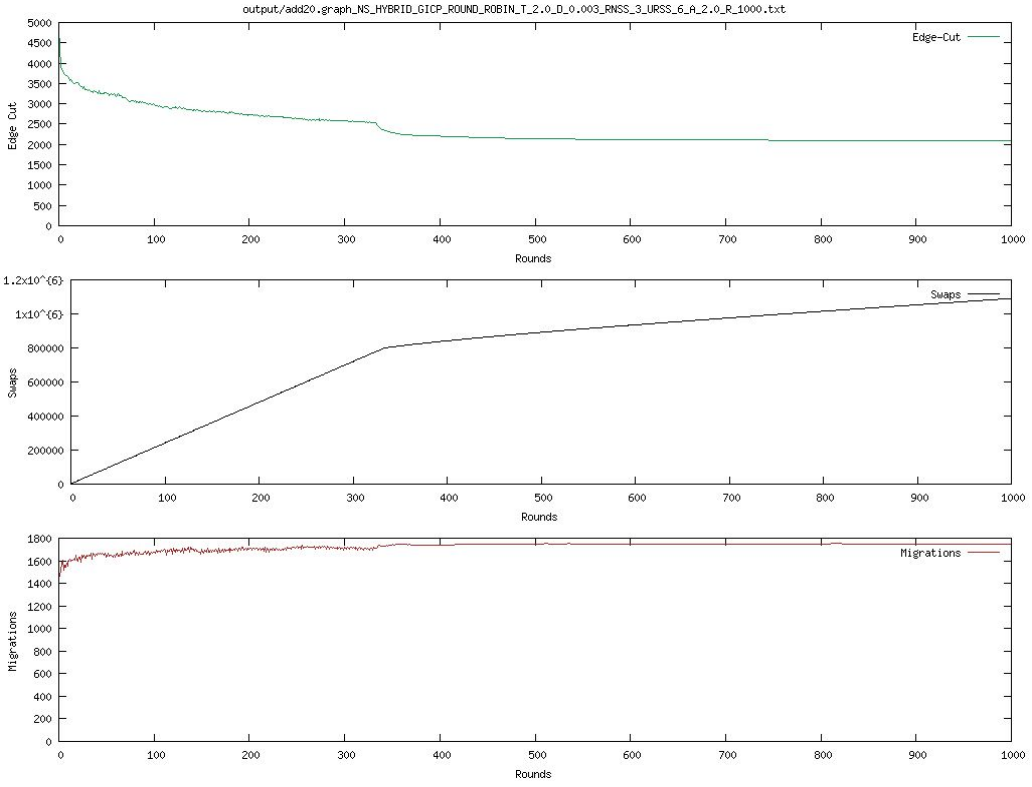
data	T	delta	rounds	edge-cut	swaps	migrations	convergence time
3elt	2	0.03	1000	776	536803	3530	382
add20	2	0.03	1000	2129	877029	1797	395
twitter	2	0.03	1000	41190	196328	2033	627



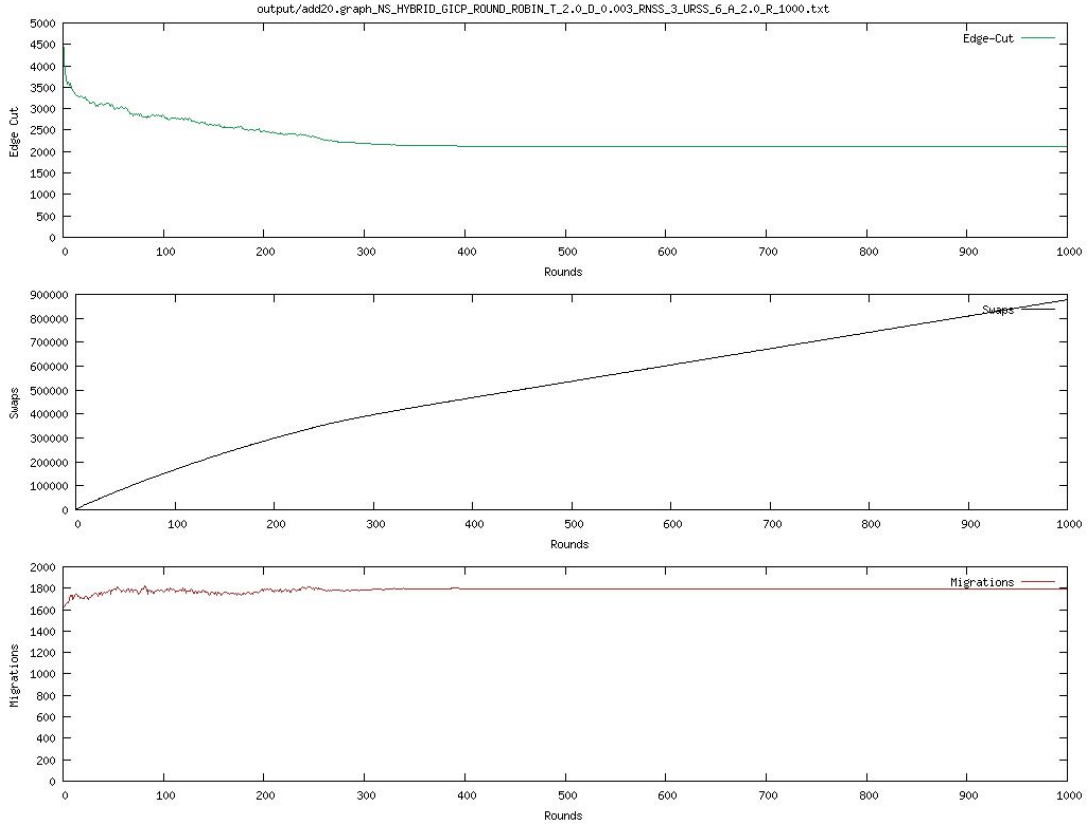
3elt without condition



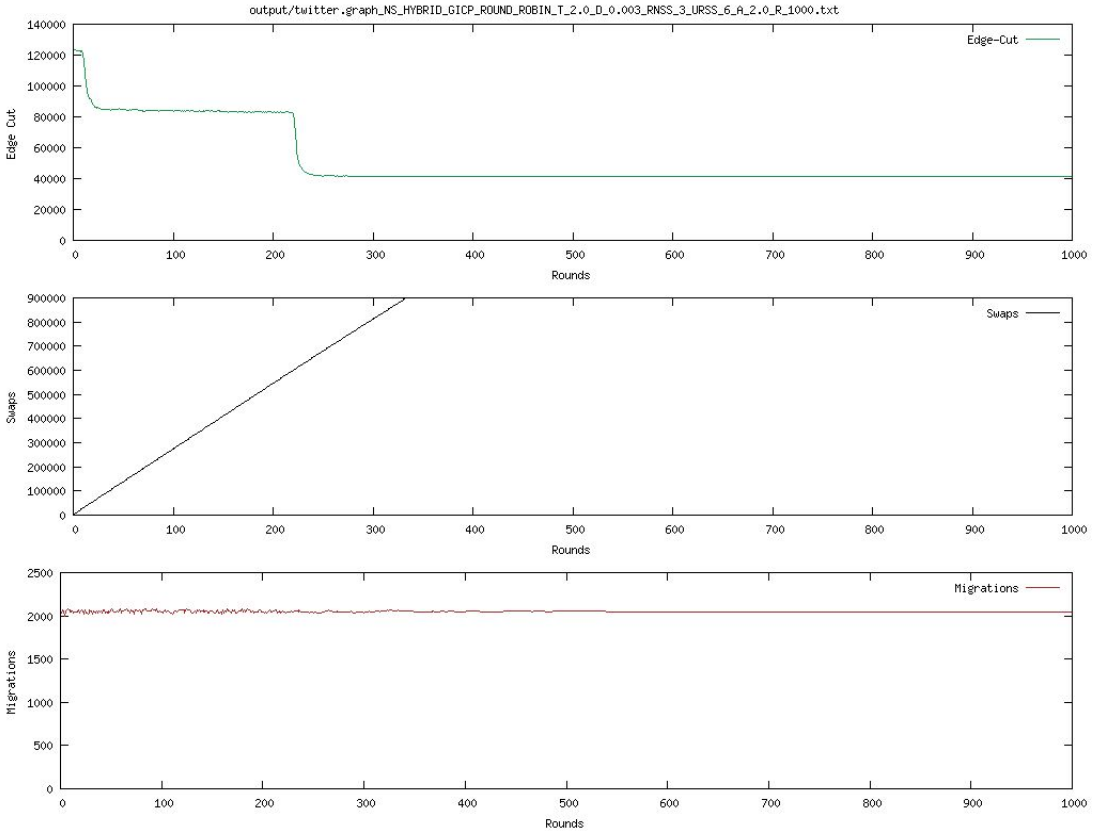
3elt with condition new =! old



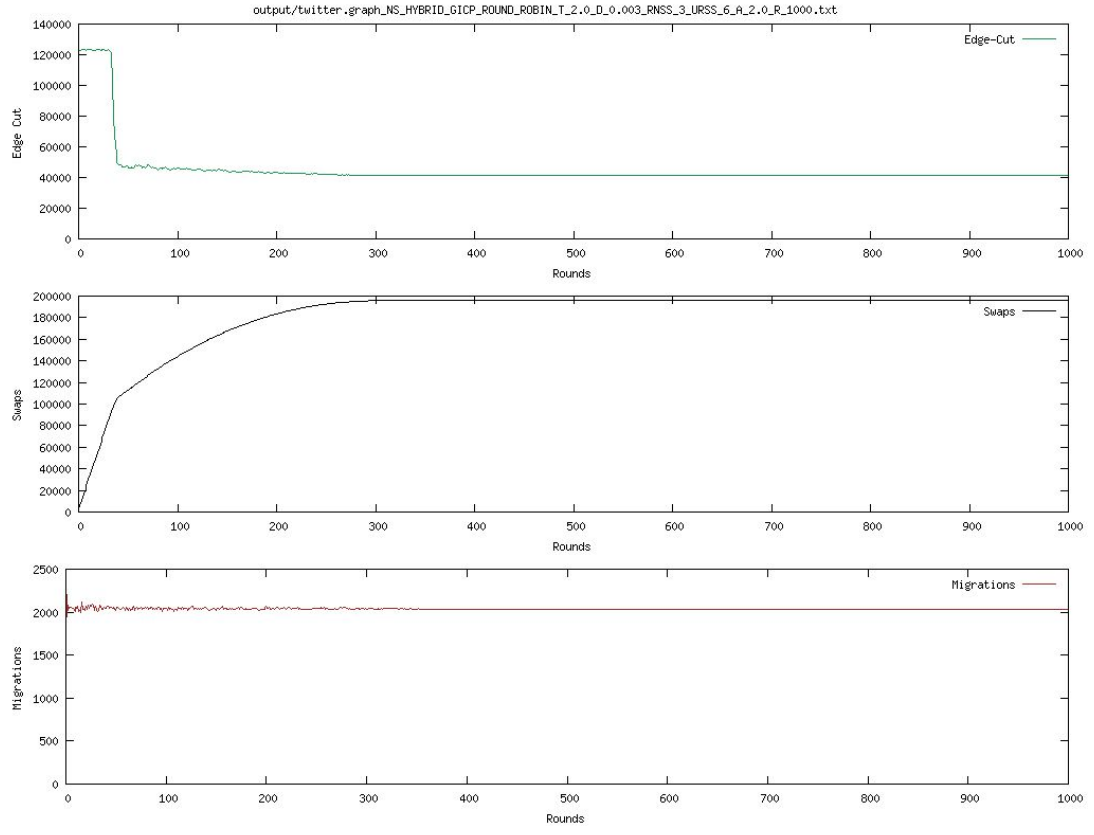
add20 without condition



add20 with condition



twitter without condition



twitter with condition

Task 2

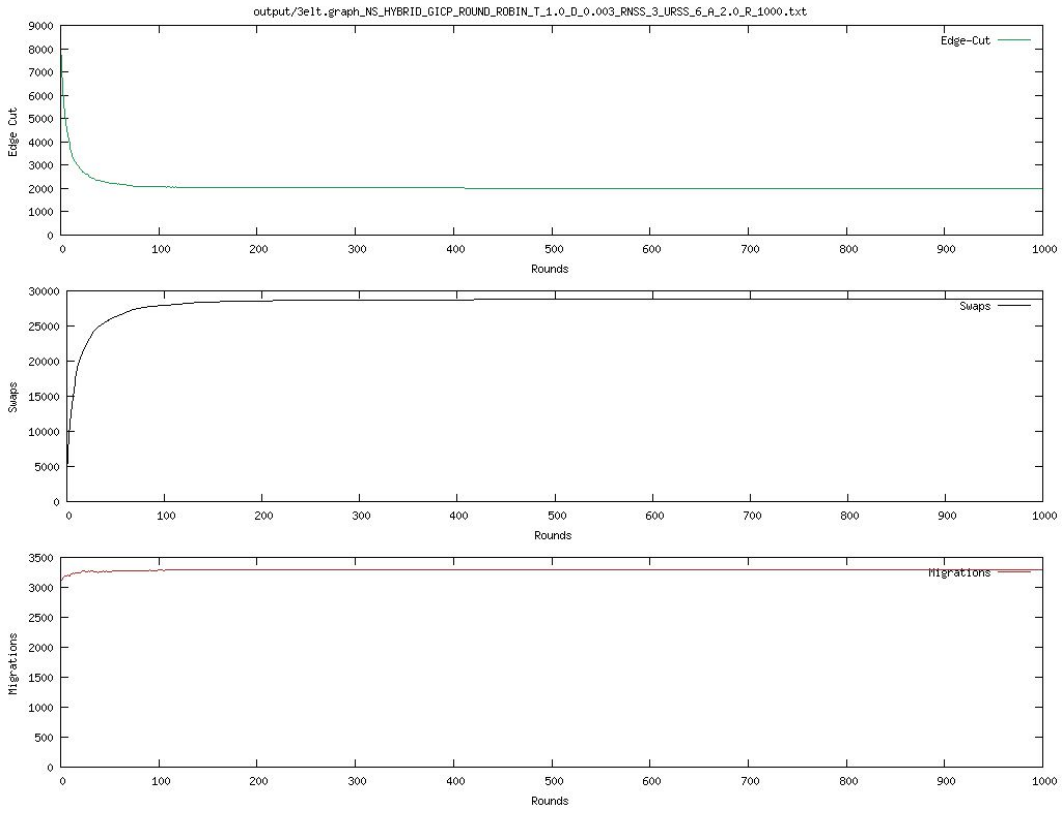
In this task, two different subtasks were done. For the first one, we explored a different mechanism to control the convergence of the algorithm. The original algorithm uses linear simulated annealing cooling function to accept a solution and keep track of the best one found so far; while in this case, we will explore an exponential approach. For this method, the maximum initial temperature is 1 and it will be decreased by multiplying it by *alpha* after each round. Typical choices for *alpha* values are between 0.8 and 0.99. We set it to 0.9.

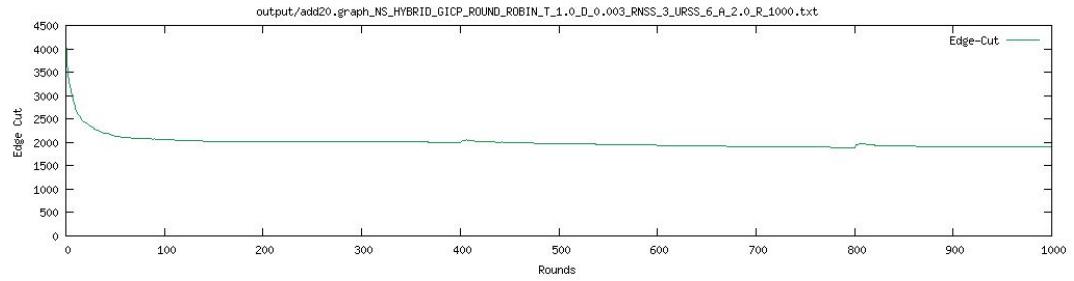
For the second subtask, the effect of restarting the simulated annealing after the algorithm has converged is explored. The frequency to be restarted is set to 400 rounds, preventing from getting stuck in local minima, accepting more swaps and looking for finding lower edge cuts.

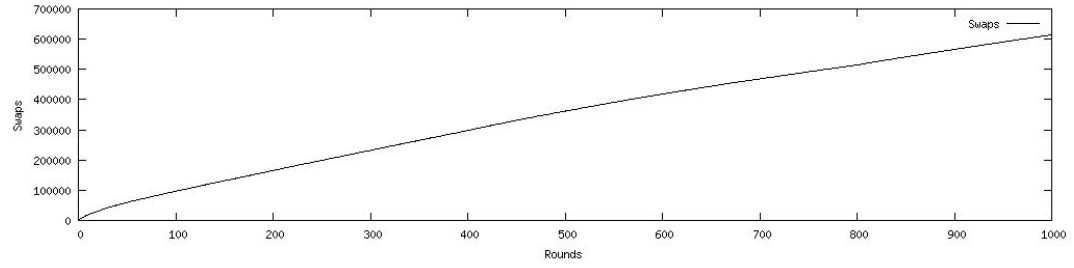
After this new approach, we can see that although for the add20 and twitter graphs there are no big differences, in all three fewer swaps are needed to start reaching the convergence and for the 3elt graph, we can see how the edge cut decreased.

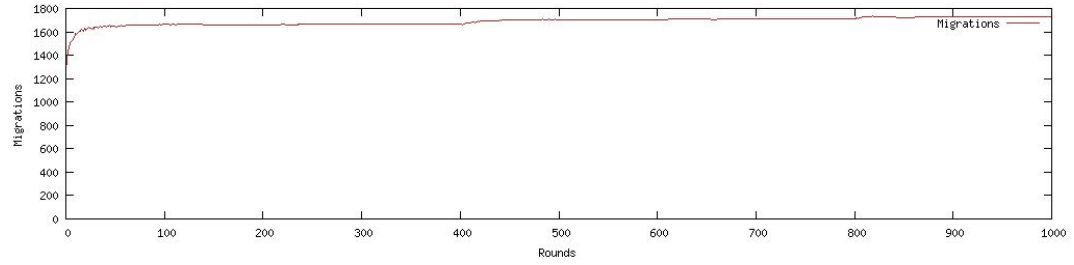
The results obtained are:

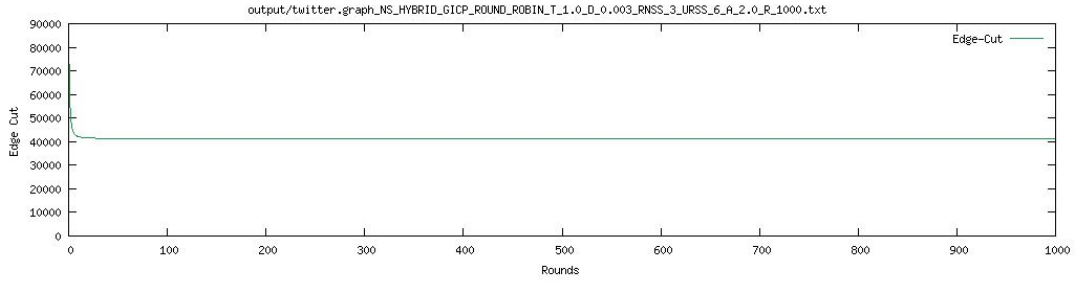
data	T	alpha	rounds	edge cut	swaps	migrations	convergence time
3elt	1	0.9	1000	1996	28794	3298	443
add20	1	0.9	1000	1901	613894	1729	989
twitter	1	0.9	1000	41273	6009	2019	848

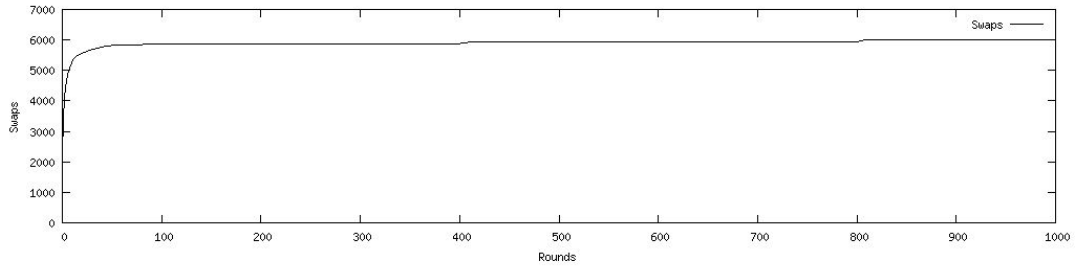


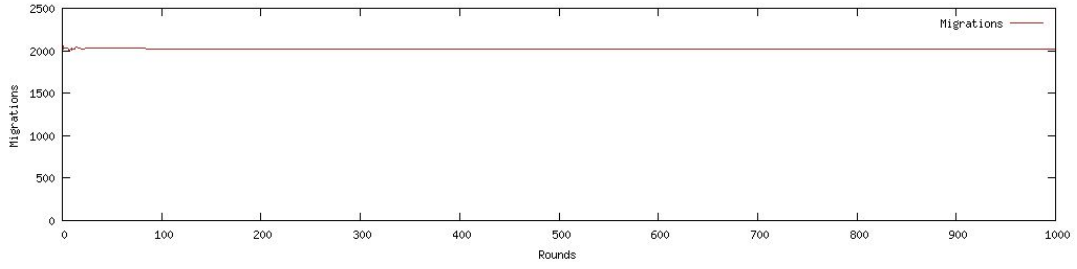

output/add20.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_1.0_D_0.003_RNSS_3_URSS_6_A_2.0_R_1000.txt
Edge-Cut
Edge Cut
Rounds


Swaps
Swaps
Rounds


Migrations
Migrations
Rounds


output/twitter.graph_NS_HYBRID_GICP_ROUND_ROBIN_T_1.0_D_0.003_RNSS_3_URSS_6_A_2.0_R_1000.txt
Edge-Cut
Edge Cut
Rounds


Swaps
Swaps
Rounds


Migrations
Migrations
Rounds

For the 3elt it helps to avoid convergence at a local minimum when comparing the results obtained from task1 and the same implementation but adding the reset option.

Optional

One possible approach to save computational time in case of finding no changes between several rounds consecutively could be to set a counter to keep track of the rounds with no more swaps and after no further improvement finishes the process. This could be beneficial for instance for the twitter graph since it reaches convergence very fast. The early stop has been set to 50 rounds. The Twitter graph reaches convergence at 41713 with 266 rounds. The reached value is not the minimum but is a good approximation.

We tried to reduce the number of swaps by introducing a new condition when evaluating the swap: the new situation is not equal to the old one. We tried this with Task 1 and the results are reported. The algorithm improves: the swaps decrease along with the convergence time, while the number of migrations are not heavily affected. The edge-cut stays the same (more or less) for the second and third graph, while is strongly reduced for 3elt.

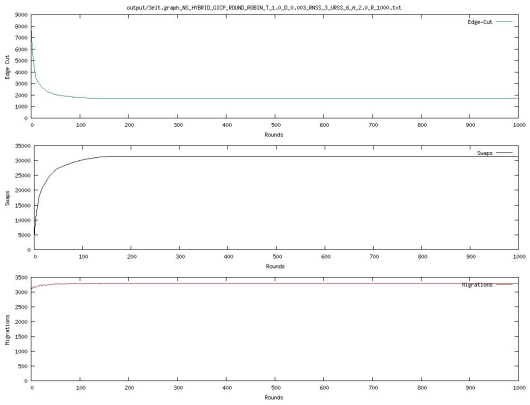
Experimenting with different parameters

We investigate how the different parameters, T, delta and alpha, affect the results for each of the graphs. We start with T, using the linear cooling function and with alpha = 2 and rounds = 1000.

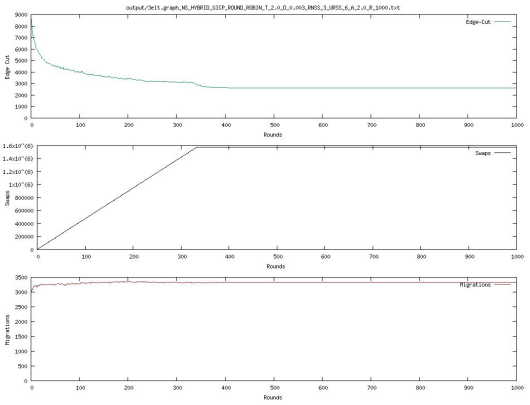
data	**T**	**edge cut**	**swaps**	**migrations**	**convergence time**
3elt	1	**1673**	31363	3287	176
3elt	2	2604	1580209	3328	472
3elt	3	2449	3149515	3362	740
add20	1	**1853**	582899	1697	906
add20	2	2095	1090263	1751	827
add20	3	2100	1787527	1779	851
Twitter	1	41675	6087	2032	636
Twitter	2	**41156**	899515	2049	813
Twitter	3	41224	1803820	2052	924

The Temperature influences the type of swaps made by the algorithm. In fact, with T>1 even bad swaps are allowed to increase the probability of making better swaps in the future. We can see that allowing no bad swaps (T=1) the graphs 3elt and add20 reaches a better edge-cut, instead with Twitter the best result is obtained with T=2. However, a lower T speed the algorithm and the convergence time decreases. Furthermore, the number of swaps are affected by T: when no bad swaps are allowed the number is evidently lower.

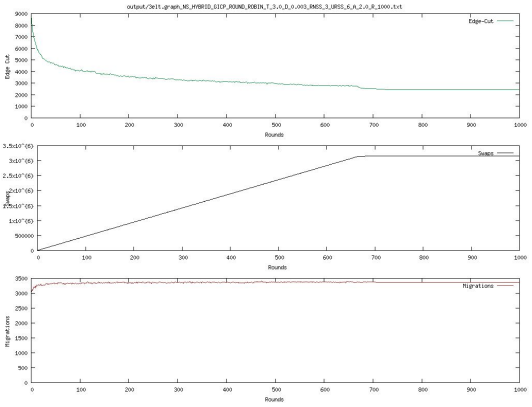
From the plots, we can observe that the convergence time is low with T=1.



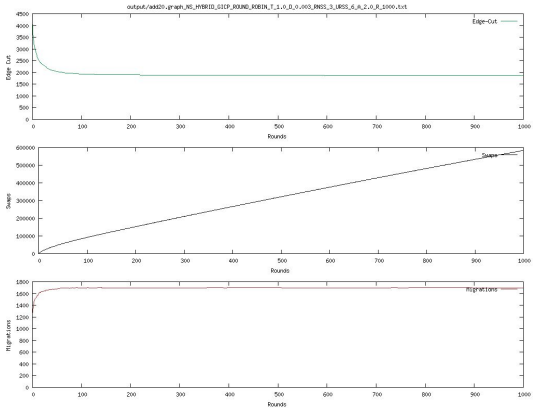
3elt T = 1, delta = 0.03



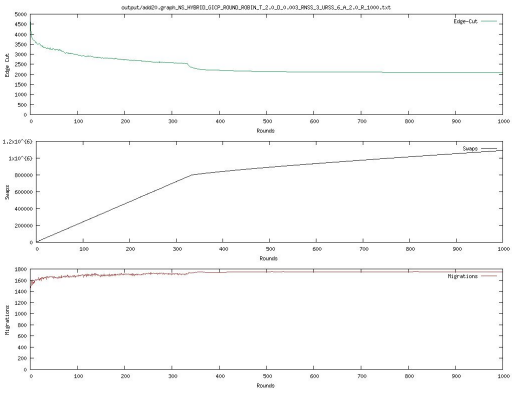
3elt T = 2, delta = 0.03



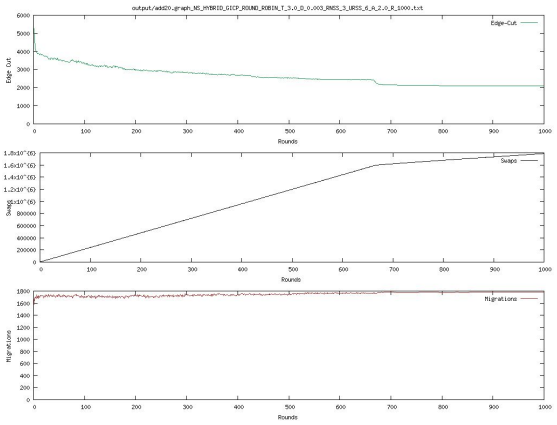
3elt T = 3, delta = 0.03



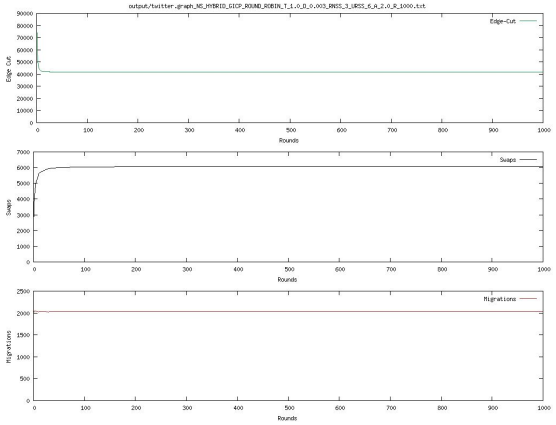
add20 T = 1, delta = 0.03



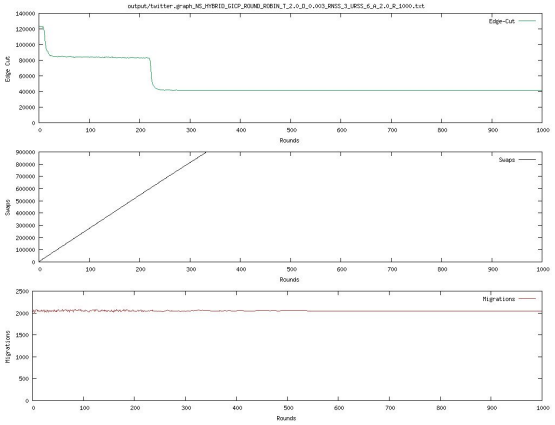
add20 T = 2, delta = 0.03



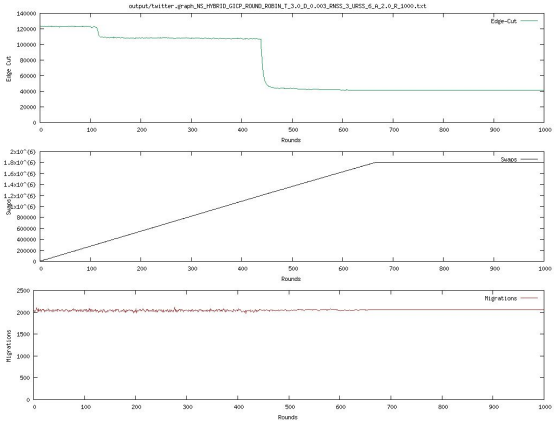
add20 T = 3, delta = 0.03



twitter T = 1, delta = 0.03



twitter T = 2, delta = 0.03



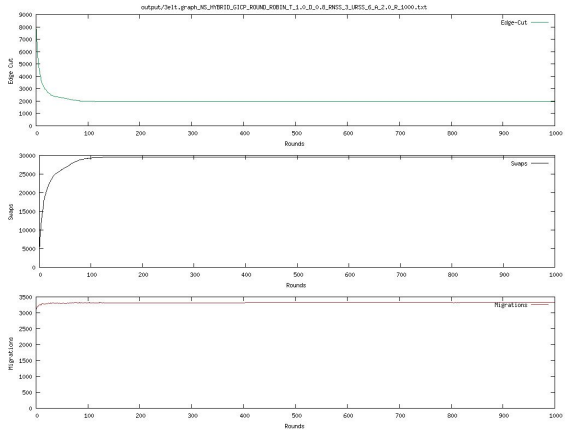
twitter T = 3, delta = 0.03

Then we tried with different deltas, using T = 1.

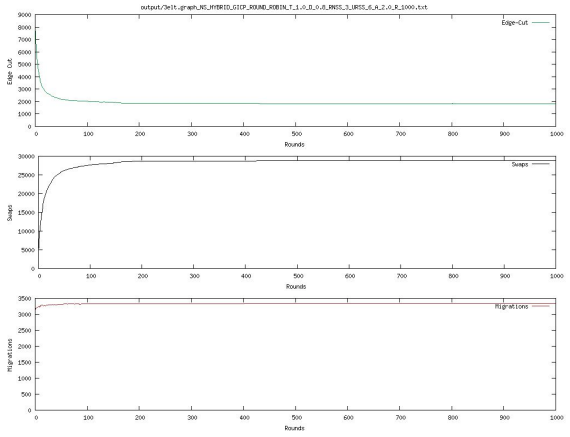
data	delta	edge cut	swaps	migrations	convergence time
3elt	0.8	1834	28802	3342	432
3elt	0.9	1780	31235	3323	294
3elt	0.99	**1685**	37096	3361	860
add20	0.8	**1851**	559293	1710	979
add20	0.9	1954	588454	1728	962
add20	0.99	2028	743159	1778	980
Twitter	0.8	**41154**	5869	2021	850
Twitter	0.9	41743	5917	2025	438
Twitter	0.99	41714	7723	2027	987

Delta affects the decreasing rate of T over the rounds, lower the delta higher the cooling rate. In simulated annealing, T is updated every round multiplying T with delta. T affects the probability of swapping: lower T higher the probability of swapping. For 3elt, the number of swaps increases while the edge cut decreases along with delta. For the other two graphs, is the opposite.

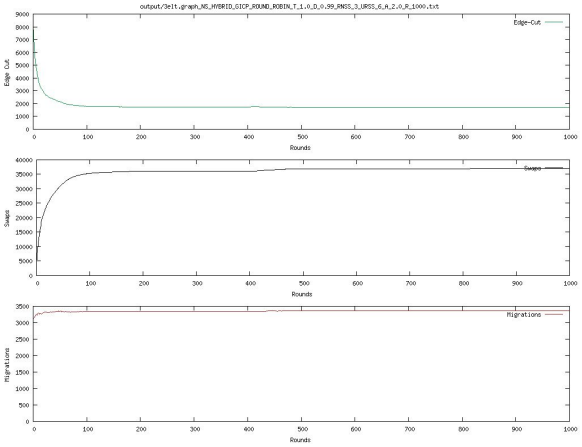
From the plots, we can observe that delta affects the convergence rate (this can be clearly seen in the swaps plot).



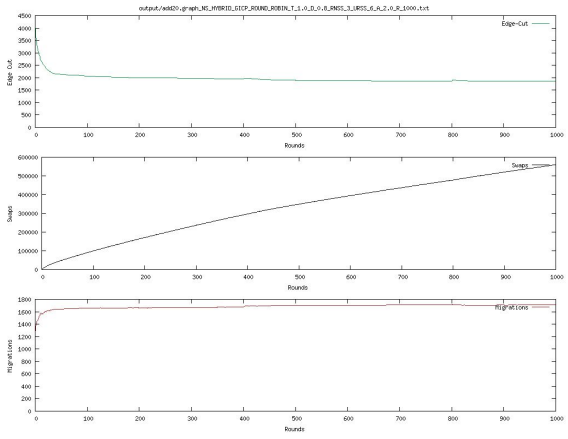
3elt delta = 0.8



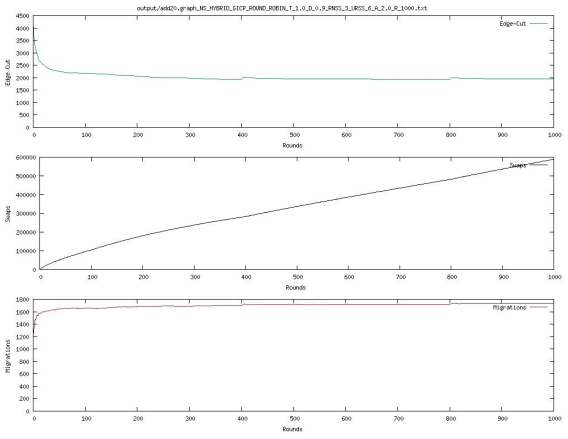
3elt delta = 0.9



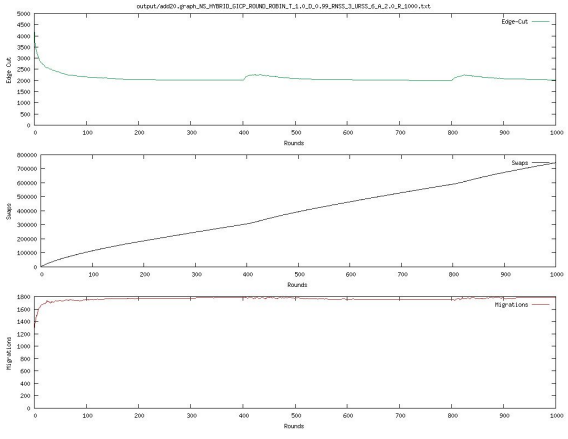
3elt delta = 0.99



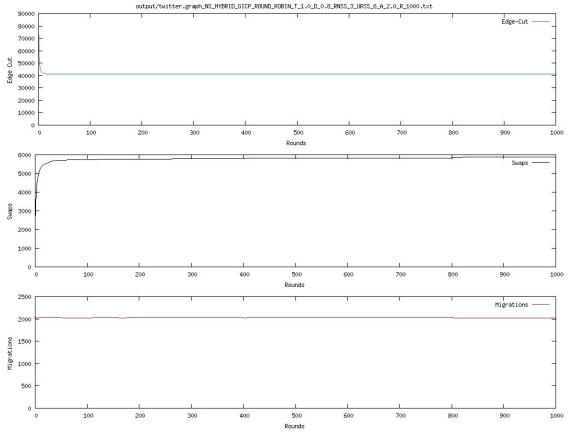
add20 delta = 0.8



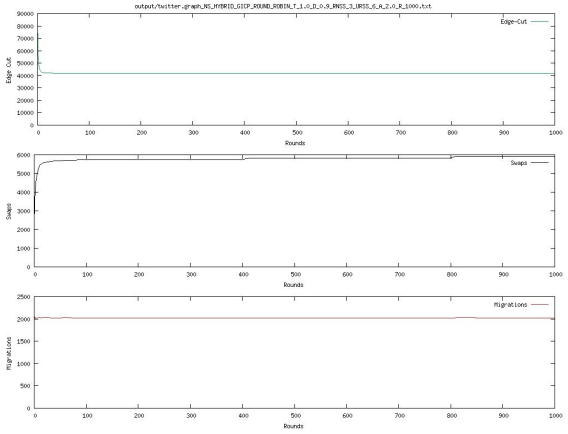
add20 delta = 0.9



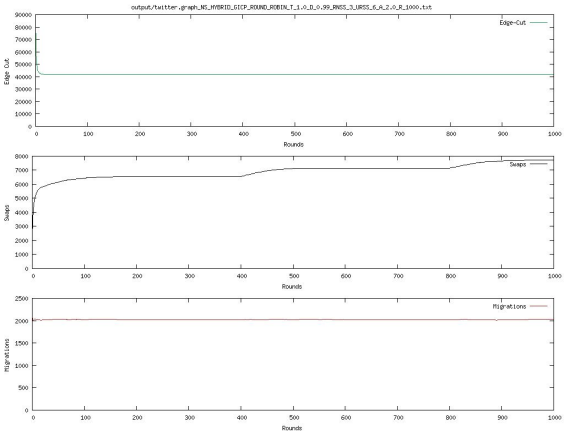
add20 delta = 0.99



twitter delta = 0.8



twitter delta = 0.9

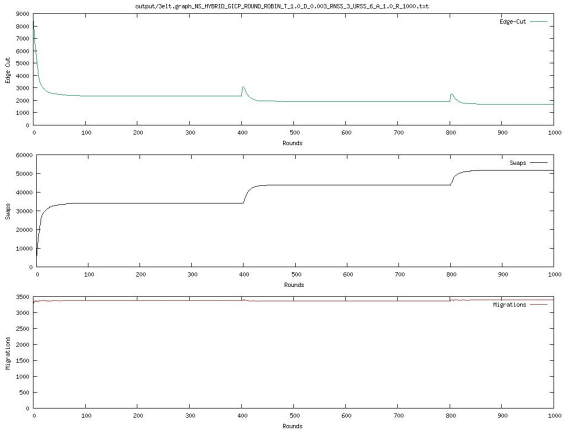


twitter delta = 0.99

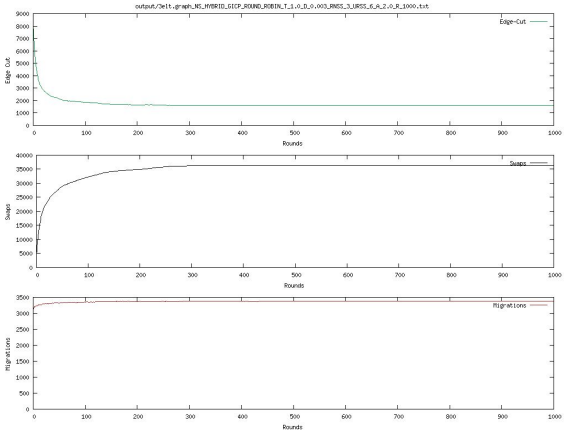
The last parameter is alpha. In this case, we used the exponential cooling function (simulating annealing) with T = 1.

data	alpha	edge cut	swaps	migrations	convergence time
3elt	1	1666	51911	3398	913
3elt	2	**1588**	36234	3371	342
3elt	3	1820	29978	3311	193
add20	1	1993	647113	1785	981
add20	2	**1869**	604907	1737	960
add20	3	2088	735507	1702	983
Twitter	1	**40844**	6059	2019	893
Twitter	2	41179	6060	2039	828
Twitter	3	42140	6071	2030	899

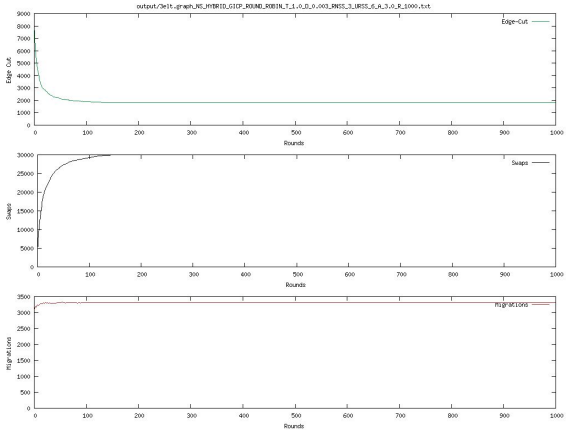
Alpha is used to calculate the value *new* and *old,* which are the number of same-color nodes connected before and after the swaps elevated by alpha. With alpha>1, the difference between *new* and *old* increases exponentially, so it is more likely to swap. This might cause a worsening of the edge-cut value. In the table, we can observe that with alpha=3, we have the worse results for all the graphs. With alpha=1, we swap only when *new* represents a relevant improvement (when the difference is low, the probability to swap is low as well). The best alpha value is not the same for all the graphs and we believe that the topology of the graphs is the cause of this.
From the plots, we can observe some spikes around 400 and 800 rounds when the T is restarted (the value is reset to the initial one). The edge-cut and swaps increase because when T is set to the initial value (T=1) even bad swaps are allowed.



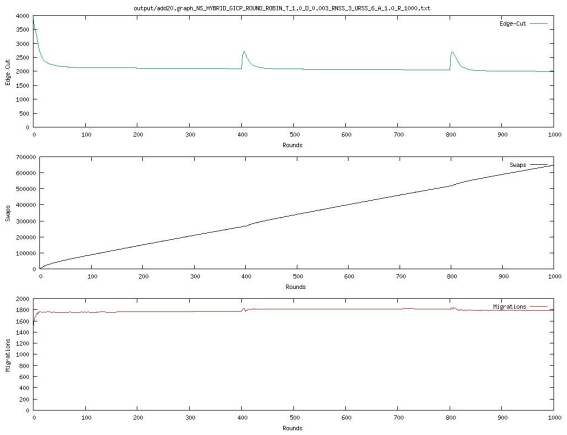
3elt alpha = 1



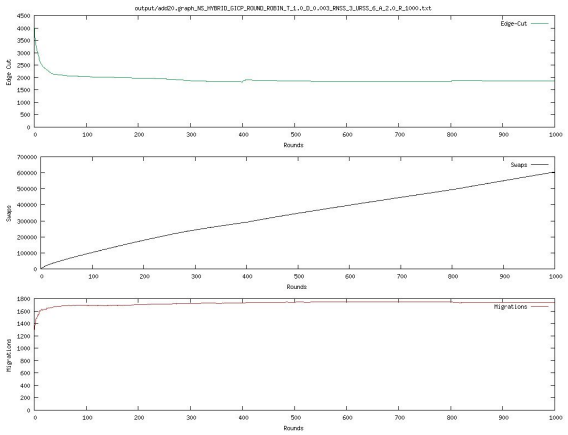
3elt alpha = 2



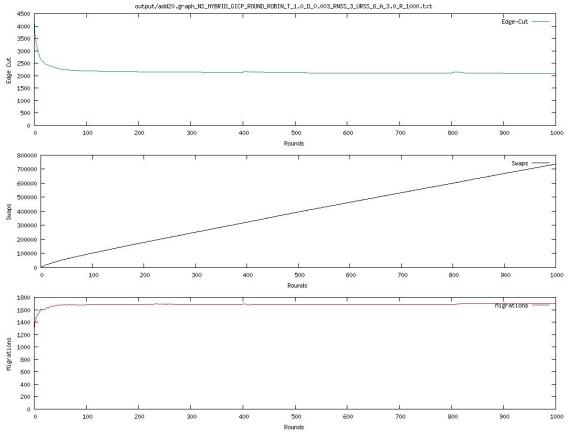
3elt alpha = 3



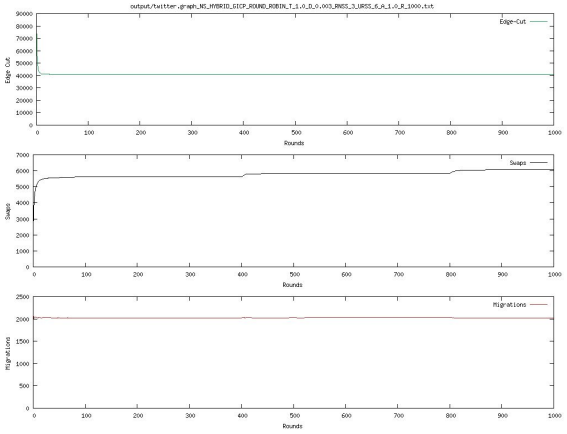
add20 alpha = 1



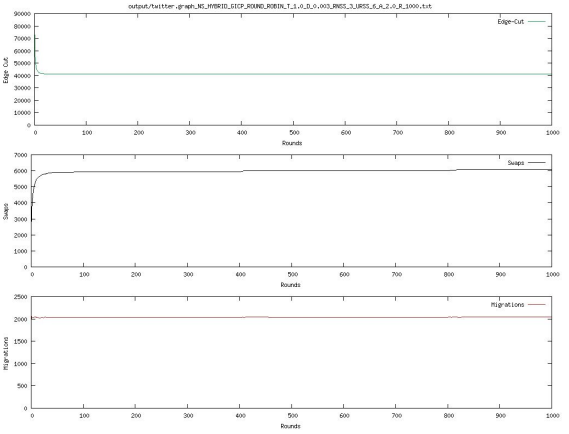
add20 alpha = 2



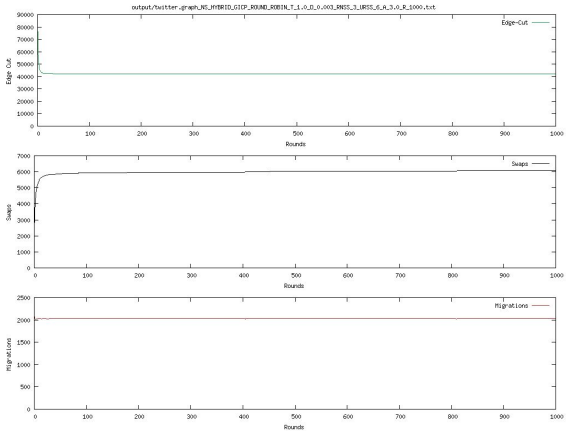
add20 alpha = 3



twitter alpha = 1



twitter alpha = 2



twitter alpha = 3