Data Mining, ID2222 HT201

Homework Assignment 5

Frano Rajic, Ivan Stresec

December 7, 2020

1 Objective

The graph partitioning problem, sometimes referred to as the min-cut problem, is formulated as dividing a graph into a predefined number of components, such that the number of edges between different components is small. A variant of this problem is the balanced or uniform graph partitioning problem, where it is also important that the components hold an equal number of nodes. The examples of important applications include biological networks, circuit design, parallel programming, load balancing, graph databases and on-line social network analysis. The motivation for graph partitioning depends on the application. A good partitioning can be used to minimize communication cost, to balance the load, or to identify densely connected clusters.

The problem that we are solving in this homework is the distributed balanced k-way graph partitioning. The problem can be seen as coloring the graph with k colors in a balanced way so that the number of edge cut between nodes of different colors is the minimum possible.

2 Solution

We use the algorithm proposed by F. Rahimian, et al. in the paper JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning.

2.1 The Ja-Be-Ja algorithm

Let the energy of the system be the number of edges between nodes with different colors, and let the energy of a node be the number of its neighbors with a different color. Each color represents a partition of the graph, of which there are k. The basic idea of **Ja-Be-Ja** is to initialize colors uniformly at random, and then to apply heuristic local search to push the configuration towards lower energy states (min- cut). Note that a worse solution is one that increases the total edge cut (and thus the energy), whereas a better solution decreases it.

The local search operator is executed by all graph nodes in parallel. Each node iteratively selects another node from either its neighbors or a random sample and investigates the pair-wise utility of a color exchange. If the color exchange decreases the energy then the two nodes swap their colors. Otherwise, they preserve their colors.

When applying local search, the key problem is to ensure that the algorithm does not get stuck in a local optimum. For this purpose, **Ja-Be-Ja** uses a simulated annealing (SA) technique.

Note that **Ja-Be-Ja** is a heuristic algorithm, so it cannot be proven (or, in fact, expected) that the globally minimal energy value is achieved. Exact algorithms are not feasible since the problem is NP-complete, so we cannot compute the minimum edge-cut in a reasonable time, even with a centralized solution and complete knowledge of the graph.

2.2 Ja-Be-Ja implementation

We used the given Java scaffolding source code on github for **Ja-Be-Ja** simulation for one-host-one-node model. Besides Java code, we wrote Python scripts to automate running different algorithm configurations in parallel, to generate all plots and logs and collect all results into a .csv document.

We tried three different versions of simulated annealing. The first version (**v1**) uses a linear decrease in the temperature. The second version (**v2**) uses an exponential decrease in the temperature with a acceptance probability of a bad solution given as $e^{\frac{-\Delta E}{T}}$ which is always greater than one if the energy is negative a.k.a. decreased a.k.a. a better solution was found. This way of simulated annealing is described in this blog post by Katrina Ellison Geltman.

Bonus
The third version (**v3**) has also a exponential decrease in temperature, but the acceptance probability has been changed to use a sigmoid function of form $\frac{1}{1+e^{\frac{-\Delta E}{T}}}$.

* * *

All three versions do simulated annealing restarts after a certain amount of rounds in a cool-down state, as the algorithm converges very short after the cool-down and restarting the simulated annealing process makes the algorithm explore more solutions. This leads to better results as compared to having just one simulated annealing, without restarts.

3 Testing and evaluation

The algorithms were tested on the following graphs:

- 3elt
- 4elt
- add20
- data
- vibrobox
- Twitter
- Facebook

The graphs were taken from Walshaw's graph partitioning archive. Table 1 shows the best-known min cuts for four partitions, as reported in Walshaw's archive and in the **Ja-Be-Ja** paper.

Graph	Nodes	Edges	Edge cut
3elt	4720	13 722	201
4elt	15 606	45 878	327
add20	2395	7462	1203
data	15 606	45 878	327
vibrobox	12 328	165 250	19 245
Twitter	2731	164 629	41 040
Facebook	63 731	817 090	117 844

Table 1: The table shows which datasets were used and describes them with the number of vertices and edges. The last column shows the best known edge cut for four partitions, as reported in Walshaw's archive for all datasets but the last two. For Facebook and Twitter, the results is shown for the best result in F. Rahimian, et al. paper JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning.

The results of running all three versions of the **Ja-Be-Ja** implementation are shown in figure 1. Overall, **v2** performed the best among the three versions. Good parameters values depended on the graph used. We did not conduct an extensive parameter search, as the tests took a long time, so the parameters might not be very close to the optimum. From the results shown, good parameters for **v1** were $\alpha = 2$ and $T = 3$ and good parameters for **v2** were $\alpha = 2$ and $T = 2$. For **v3** the value of $\alpha = 2$ gave better results, but it is hard to conclude which value of temperature was the best.

For the Twitter and Facebook datasets, we conducted a smaller number of experiments. Out of these experiments, the best results were as follows:

- Twitter
  - edge-cut:41 139
  - temperature: 4
  - α: 2
  - delta: 0.003
  - RNSS: 3
  - URSS: 6
- Facebook
  - edge-cut:124 205
  - temperature: 10
  - α: 2
  - delta: 0.003
  - RNSS: 3
  - URSS: 6

A comparison of best results that we achieved with results of Rahimian et al. and of best results described in 1 is shown in table 2.

Graph	**Our best**	**Rahimian, et al.**	**Benchmark**
3elt	**371**	390	201
4elt	**1280**	1424	327
add20	1701	**1206**	1203
data	1351	**775**	327
vibrobox	24 910	**23 174**	19 245
Twitter	41 139	**41 040**	41 040
Facebook	124 205	**117 844**	117 844

Table 2: Comparison of our best results with the best results in F. Rahimian, et al. paper JA-BE-JA: A Distributed Algorithm for Balanced Graph Partitioning and with the best results described in 1 (last column) for different graphs.

Selected graphs showing edge cut, number of swaps and number of migrations with respect to iteration number are shown in figures 2, 3, 4 and 5. Notice how the occasional jumps in edge cut value occur, they are easy to see in figures 2 and 3. The jumps occur because of the simulated annealing process restarts, triggered after certain number of cycles in cool-down.

3elt

		Temp		
	v1	1	2	3
Alpha	2	1673	1070	1350
	2.5	2051	1075	1354
	3	2051	1000	1272

		Temp			
	v2	0.8	0.9	1	2
Alpha	2	841	708	600	371
	3	1482	1488	1395	1121

		Temp			
	v3	0.25	0.5	1	4
Alpha	2	592	608	714	654
	3	723	743	652	627

4elt

		Temp		
	v1	1	2	3
Alpha	2	6678	3811	4612
	2.5	6685	3606	4329
	3	6147	3602	4043

		Temp			
	v2	0.8	0.9	1	2
Alpha	2	2369	2075	2076	1280
	3	5378	5541	5138	3389

		Temp			
	v3	0.25	0.5	1	4
Alpha	2	2456	2218	2400	2133
	3	2159	2186	1968	2163

add20

		Temp		
	v1	1	2	3
Alpha	2	1849	1991	1971
	2.5	1897	1970	1993
	3	1981	2157	2142

		Temp			
	v2	0.8	0.9	1	2
Alpha	2	2064	1738	2018	2110
	3	2246	2225	2188	2086

		Temp			
	v3	0.25	0.5	1	4
Alpha	2	2075	2003	1701	1715
	3	1786	1740	1982	1782

data

		Temp		
	v1	1	2	3
Alpha	2	2076	1359	1419
	2.5	2253	1542	1461
	3	1889	1505	1450

		Temp			
	v2	0.8	0.9	1	2
Alpha	2	1905	1747	1625	1351
	3	2150	2158	2250	1978

		Temp			
	v3	0.25	0.5	1	4
Alpha	2	1630	1448	1399	1787
	3	1782	2207	2029	2284

vibrobox

		Temp		
	v1	1	2	3
Alpha	2	32151	28713	24910
	2.5	32478	28807	27014
	3	32539	29431	25798

		Temp			
	v2	0.8	0.9	1	2
Alpha	2	31571	32154	32472	32589
	3	32064	32135	33099	32518

		Temp			
	v3	0.25	0.5	1	4
Alpha	2	31762	32527	32987	30977
	3	31498	31091	31740	31517

Figure 1: Results of running all three versions of the algorithm for 10000 rounds on 5 different graphs.

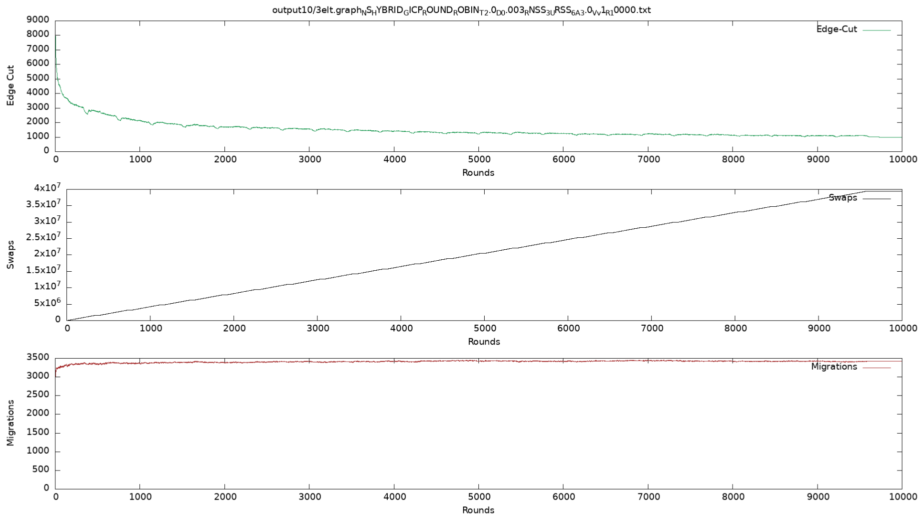

Figure 2: Graph showing edge cut, number of swaps and number of migrations with respect to the iteration (round). This run used **v1** algorithm on the *3elt* graph with following parameters: $\alpha = 3$, $T = 2$, delta= 0.003, RNSS= 3 and URSS= 6. This is the best run of **v1** on this graph from the results shown in figure 1.

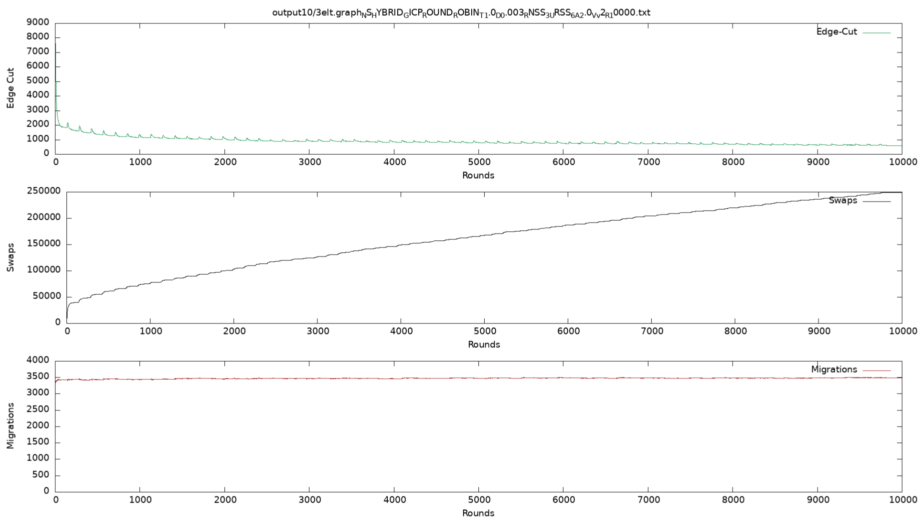

Figure 3: Graph showing edge cut, number of swaps and number of migrations with respect to the iteration (round). This run used **v2** algorithm on the *3elt* graph with following parameters: $\alpha = 2$, $T = 1$, delta= 0.003, RNSS= 3 and URSS= 6. This is the best run of **v2** on this graph from the results shown in figure 1.

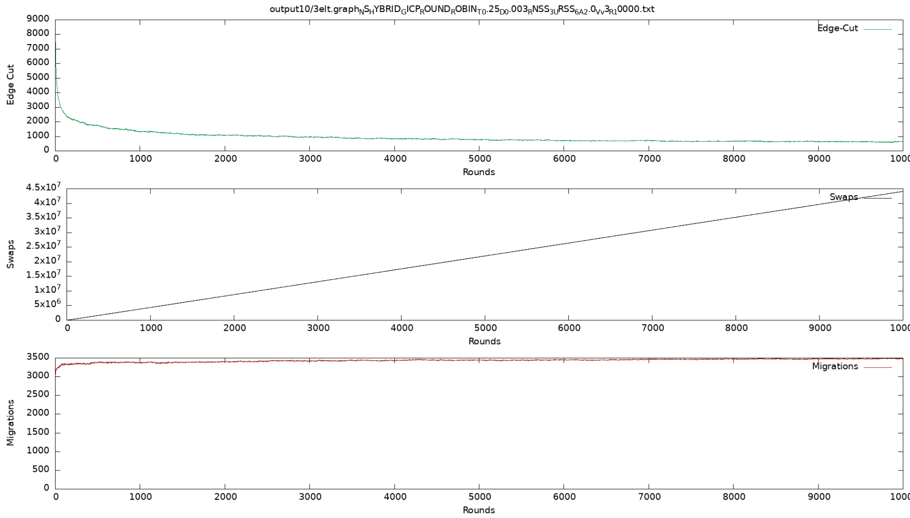

Figure 4: Graph showing edge cut, number of swaps and number of migrations with respect to the iteration (round). This run used **v3** algorithm on the *3elt* graph with following parameters: $\alpha = 2$, $T = 0.25$, delta= 0.003, RNSS= 3 and URSS= 6. This is the best run of **v3** on this graph from the results shown in figure 1.

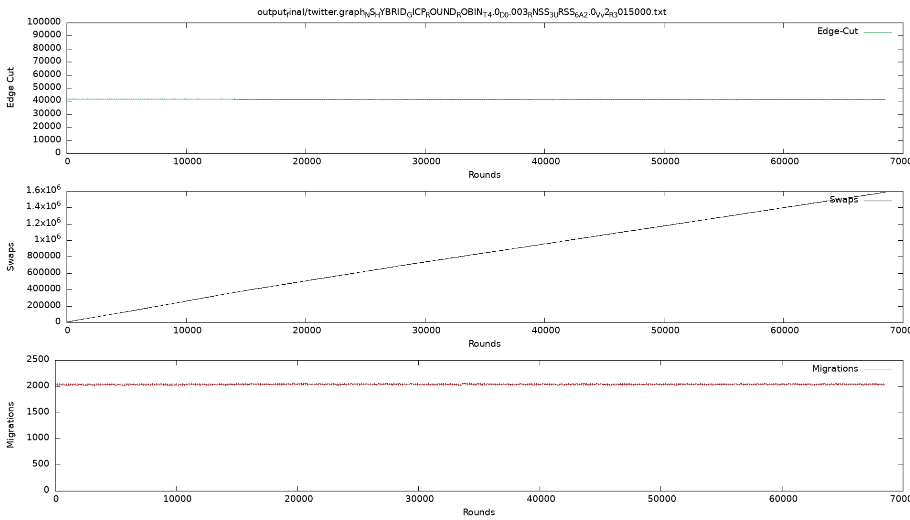

Figure 5: Graph showing edge cut, number of swaps and number of migrations with respect to the iteration (round). This run used **v2** algorithm on the *twitter* graph with following parameters: $\alpha = 2$, $T = 4$, delta= 0.003, RNSS= 3 and URSS= 6.

4 Conclusion

We tried solving the k-way graph partitioning problem using the **Ja-Be-Ja** algorithm by F. Rahimian, et al. We found it relatively easy to build upon the Java source code scaffolding given in the assignment and implement the needed versions of the **Ja-Be-Ja** algorithm. We have tested and generated a modest amount of results and synthesized them into tables and graphs. This process took quite a lot of time, but the results were quite good and satisfactory. The **Ja-Be-Ja** algorithm proved to be very effective in its aim to find k balanced partitions, based primarily on the edge-cut metric. It seems quite possible that further tweaking of parameters of the algorithm and its functions could get achieve even better results.