KTH - Kungliga Tekniska högskolan - DATA MINING 2017

Assignment 5: K-way Graph Partitioning Using JaBeJa

Filippo Boiani <boiani@kth.se>
Riccardo Sibani <rsibani@kth.se>

9 Dec 2017

NOTE
The code can be found at the following link: https://github.com/filippoboiani/data-mining

Introduction

The aim of this assignment is to implement the Ja-Be-Ja algorithm starting form the provided scaffolding file. In particular we implemented the swap and find partner functions. After the implementation we present the outputs of different Ja-Be-Ja configurations.

Approach

TASK 1

We changed the code according to the paper. For more information have a look at the code.

TASK 2

Part 1
For the first part, we added a new CLI parameter `-improvedAcceptance` and we switch the linear annealing cooling function with a logarithmic annealing one. The changes followed the article *The Simulated Annealing Algorithm*. The only difference was on the way the annealing was calculated. In fact, the paper makes a difference in terms of cost that should be minimised, while in our case we are maximising the difference between old and new value. Therefore we switched the difference using the following:

```
acceptance = Math.exp((new - old) / T) > Math.random();
```

Part 2
For the second part we modified the linear annealing function in the `saCoolDown()` method.

We first tired a sawtooth approach with the frequency related to delta and the number of rounds

```
// extra annealing for task 2.2
if (config.getExtra()) {
  // reset the temperature back to the start after it has cooled
down
  int roundToConverge = (int) (T / config.getDelta());
  if (T== 1 && round != roundToConverge && round%roundToConverge
== 0) {
    T = config.getTemperature();
  }
}
```

Then we tried an easier approach with a fixed number (trigger = 400)

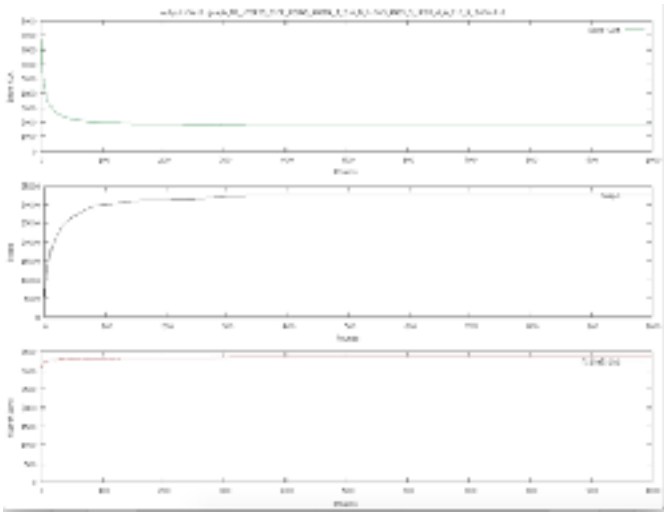
```
// extra annealing for task 2.2
if (config.getExtra()) {
  // reset the temperature back to the start after 400 rounds
  int trigger = 400;

  if (T == 1 && round % trigger == 0) {
    T = config.getTemperature();
  }
}
```

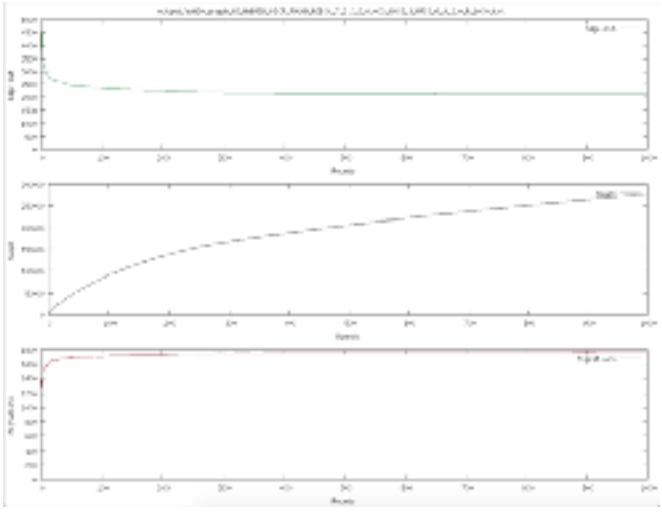
Analysis

TASK 1

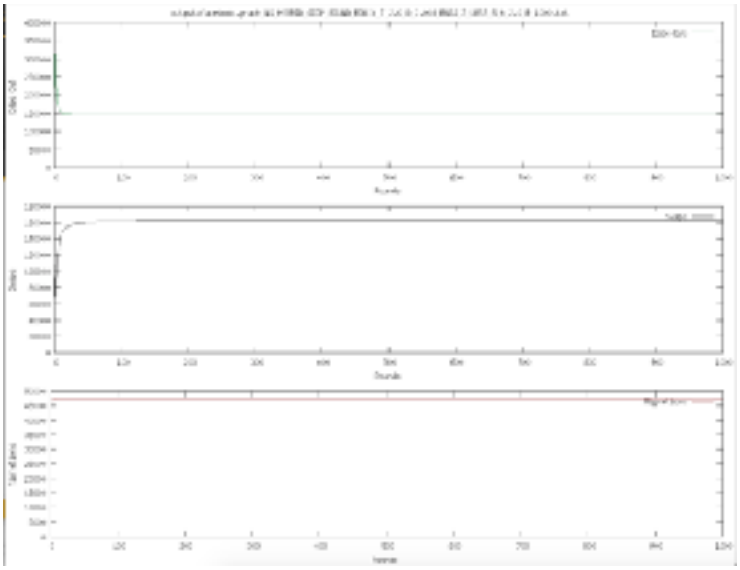
graphs/3elt.graph



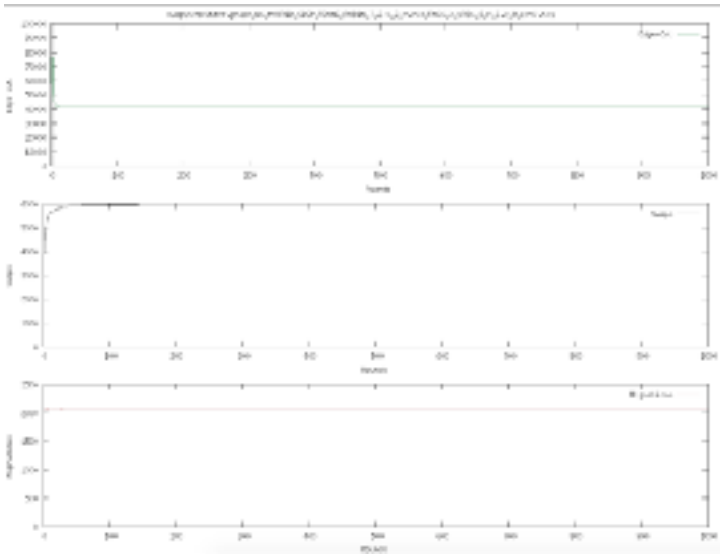
graphs/add20.graph



graphs/facebook.graph



graphs/TwitterGraph.net



TASK 2

Part 1

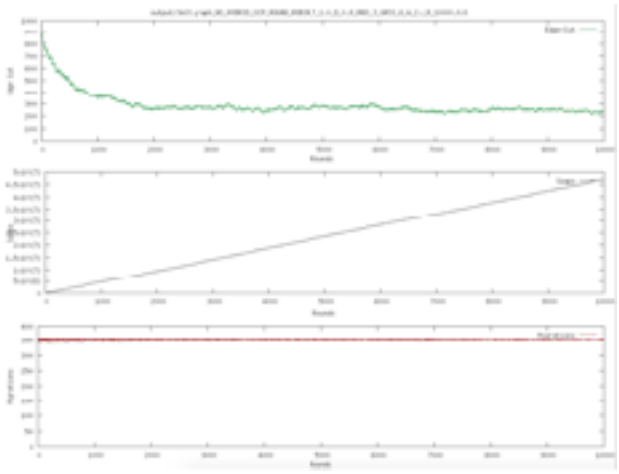
We can notice that the different simulated annealing mechanism is logarithmic: it converges faster at the beginning because the temperature is cooled down rapidly, but after a few rounds the cooling down is slower. Therefore, from an absolute perspective, the logarithmic simulated annealing takes more rounds to converge compared to the liner solution proposed by the Ja-Be-Ja algorithm.

The convergence depends on the number of rounds, the initial T (that cannot be higher than 1) and the delta that, for our experiments, is kept between 0.99 an 0.5.

graphs/3elt.graph

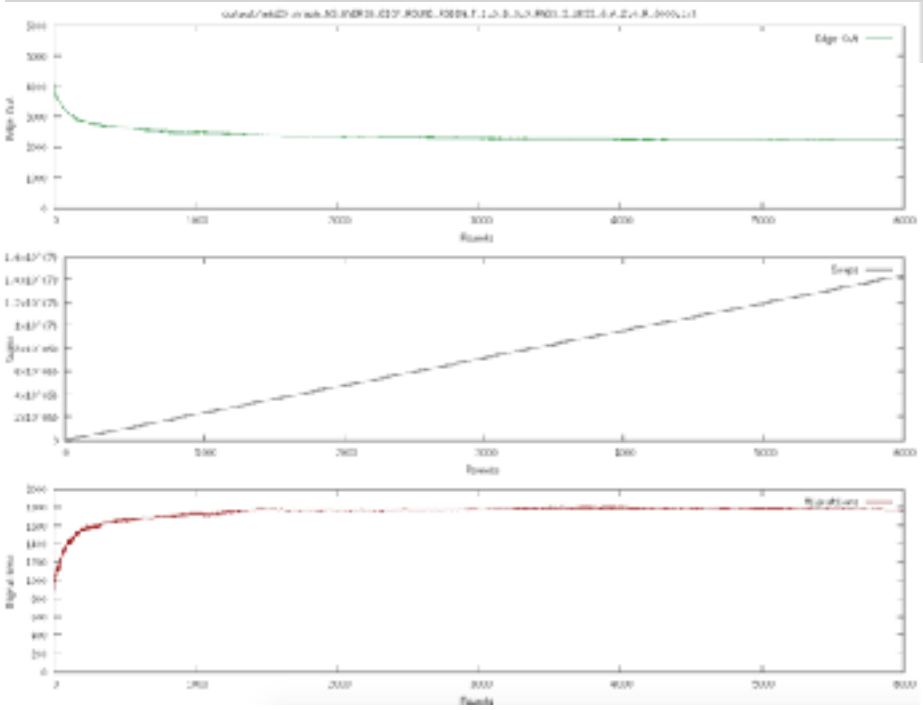
```
./run.sh -graph graphs/3elt.graph -rounds 2000
round: 1999, edge cut:1741, swaps: 32663, migrations: 3346

./run.sh -graph graphs/3elt.graph -logAccept -rounds 10000 -temp 1 -delta 0.9
round: 9999, edge cut:2674, swaps: 9437780, migrations: 3516
```



graphs/add20.graph

```
./run.sh -graph graphs/add20.graph -rounds 2000
round: 1999, edge cut:2122, swaps: 414173, migrations: 1757
```



Part 2.
By setting the temperature back to the start when the algorithm is converging causes the number of edge-cuts to decrease further. Therefore this small change improves the quality of the algorithm. The final number of edge-cuts depends on the number of rounds, the initial temperature and the frequency of temperature rise. For our experiments we raise the temperature back to initial value every double of the rounds.

We found out that the statement “For example if **T** is 2 and **delta** is 0.01 then after 200 rounds the temperature will cool down to 1 and no more bad-swaps will be accepted. Ja-Be-Ja will converge soon after that.” is not necessarily true. For example, the graph **3elt.graph** with delta = 0.03, T = 2 converges after 360 rounds, while the temperature equals to 1 is reached after 34 rounds. The same for **graphs/add20.graph.**

```
./run.sh -graph graphs/3elt.graph -rounds 2000
round: 1999, edge cut:1741, swaps: 32663, migrations: 3346

./run.sh -graph graphs/3elt.graph -logAccept -rounds 2000 -temp 1 -delta 0.9
round: 1999, edge cut:2674, swaps: 9437780, migrations: 3516

./run.sh -graph graphs/3elt.graph -extra -rounds 2000
round: 1999, edge cut:1562, swaps: 58131, migrations: 3343
```

Bonus

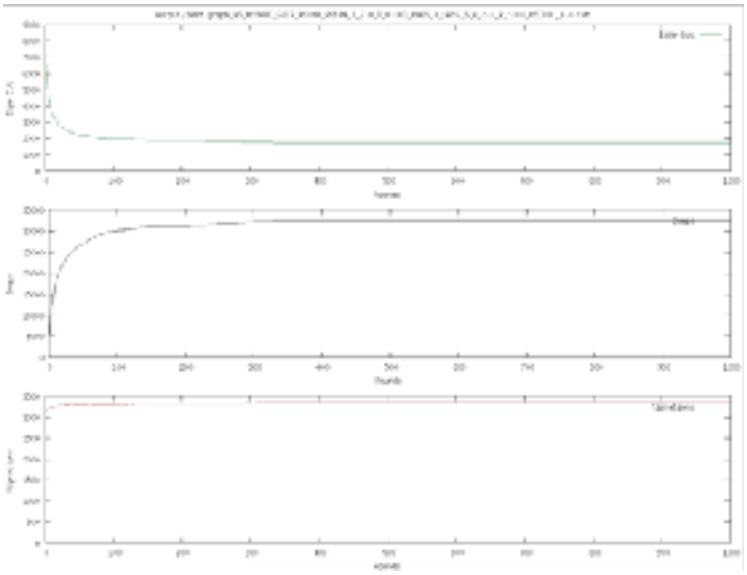
In the current implementation, with the 3elt.graph we converge to 1741 after 1000 rounds (but actually is able to stop there around the 350th round).

We decided to *shuffle the cards* once the balance is reached. Therefore, when there is a new swap and (T<= 1), we multiply T by a constant (around 1.1 -> 10% to 2 -> 100%). We decided to call this constant *recoil*.

This are the results for graph 3elt.graph and 1000 cycles. With recoil = 1.1.

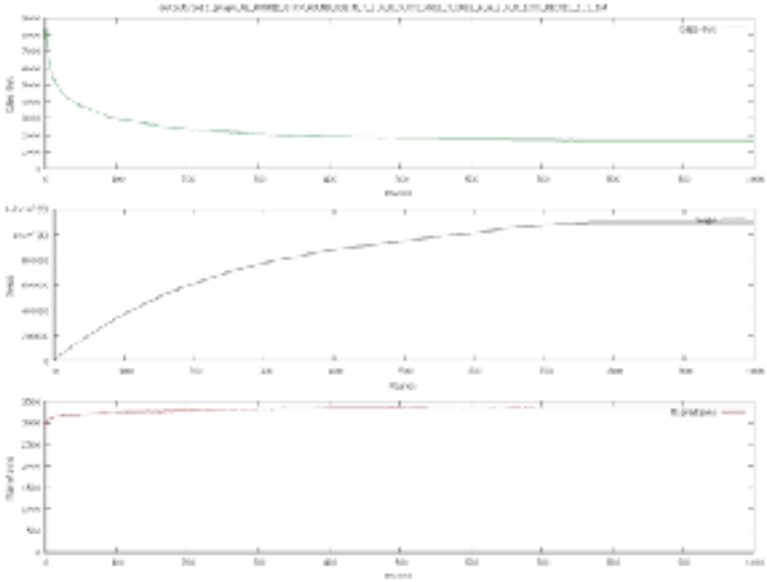
graphs/3elt.graph - 1000 rounds - recoil = 1 (null)

```
./run.sh -graph graphs/3elt.graph -rounds 1000
round: 999, edge cut:1741, swaps: 32663, migrations: 3346
```



graphs/3elt.graph - 1000 rounds - recoil = 1.1

```
./run.sh -graph graphs/3elt.graph -rounds 1000 -recoil 1.1
round: 999, edge cut:1675, swaps: 1096984, migrations: 3332
```



We can observe that the new solution converge to a more optimal solution even though, taking a longer time.
It is important to notice that with the previous implementation, incrementing the number of rounds'd not help. With this solution, instead, with an higher number of rounds we can reach a better solution.

graphs/3elt.graph - 10000 rounds - recoil = 2

```
./run.sh -graph graphs/3elt.graph -rounds 1000 -recoil 2
round: 9999, edge cut:1035, swaps: 37825334, migrations: 3455
```