Vivaldi Report

Pekko Lipsanen, Martin Melhus

December 16, 2013

Introduction

This work studies accuracy of Vivaldi algorithm. We have implemented the algorithm based on work done by Dabek et al. ("Vivaldi: A Decentralized Network Coordinate System"). In this work, we show how different parameters affect the outcome.

Hypothesis

We run our implementation varying two parameters, number of iterations and number of random-selected neighbors. Because the algorithm is iterative, we expected that greater number of iterations will give us better results (e.g. lower error). Also, if we calculate the values with more neighbors, we expected that algorithm would converge faster.

However, the input data is not perfect. Distances between two nodes might not be symmetric, e.g. distance from A to B is not the same as from B to A. Also, triangle inequality is violated - sometimes it might be faster to go from A to B via C than using the direct path. Because of these biases in input data, we were quite sure that we wouldn't achieve perfect results and some amount of error will be left. That said, two important questions remain open:

1. How fast will the error stabilize?
2. How small of an error we are able achieve?

Implementation

The Vivaldi algorithm doesn't clearly state all the details. The most important open questions are about the initial state and first movements: when nobody knows anything about their position, how we could calculate the first positions and movements?

We resolved the problem by setting everyone's initial coordinates at zero. After that, initial movement direction is randomized, but amount of movement is calculated as the algorithm indicates.

We chose the variable c_c to be equal to c_e.

Calculation of Relative Error

Every node has a measured round-trip time to a neighbor denoted as $rtt_{measured}$. Also, it has a predicted round-trip time $rtt_{predicted}$ which is the euclidian distance between the two nodes, derived from the resulting graph of running the Vivaldi algorithm. The relative error e between a node i and its neighbor k is computed by calculating the following.

$$e_{i_k} = |\frac{rtt_{predicted} - rtt_{measured}}{rtt_{measured}}| \tag{1}$$

The average relative error in a node $\bar{e_i}$ with K neighbors is calculated by the following formula. We use this average as our measure of the relative error in a node.

$$\bar{e_i} = \frac{1}{K}\sum_{k} e_{i_k} \tag{2}$$

Having this we can calculate the relative error E in the graph consisting of n nodes by the following formula

$$E = \frac{1}{n}\sum_{i} \bar{e_i} \tag{3}$$

Running simulation

We chose to create a new file simulation.py to be able to collect data from several runs at once. We chose to do this to reduce the difference in environment in different runs. Doing this we were also able to run the same configuration multiple times, thus adding confidence that our results are describing the general case and not some obscure special case. The number of times each configuration is simulated is denoted in the code as runs_per_config, and can be found in the simulation.py file.

To run the simulation with multiple configurations at once one can use the following command. (The table of results printed to stdout is representing the first configuration, being values for i and K)

```
python simulation.py VivaldiMatrix.txt
```

Results

All our results were obtained with the following configuration:

```
runs_per_conf = 3
random.seed(1234)
```

We calculated the average relative error for each node after all the iterations, and some metrics on this value can be found in table 1.

The results we obtained when varying the number of neighbors and iterations can also be seen in the graphs in appendix A.

K	i	mean	median	var	max	min	50th percentile	90th percentile	99th percentile
3	20	297	243	37655	1058	44	244	638	1058
3	200	296	234	39674	1071	41	237	615	1071
3	1000	296	246	41759	1104	38	249	612	1104
10	20	289	242	36191	1146	44	245	544	1146
10	200	285	245	34783	1050	40	245	569	1050
10	1000	297	238	41146	1091	39	238	643	1091
20	20	282	226	32808	991	39	228	566	991
20	200	293	230	39454	1068	39	231	610	1068
20	1000	293	240	38752	1145	39	241	578	1145

Table 1: Relative errors (%) from simulations

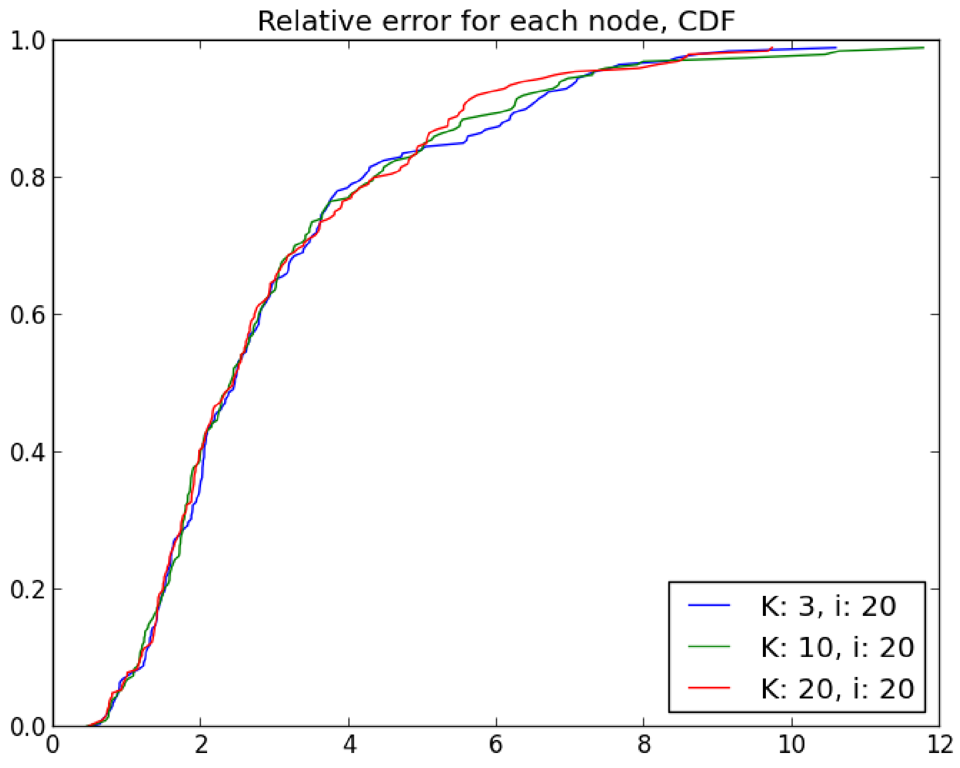
Discussion

We can see from table that parameters have very little impact on actual results. This means that the algorithm converges towards stable state very fast, and this can be also seen from graphs in Appendix A. Average movement keeps relatively stable after only a few iterations, and also absolute error keeps stable after only few dozen iterations.

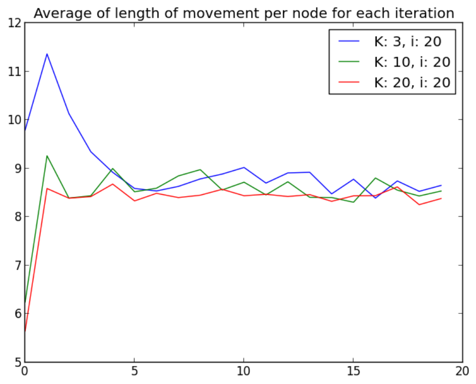
However, errors at stable stage are still quite large, median error being more than 200 percent. Vivaldi algorithm assumes that input data has basic geometric properties: distance from A to B is the same as distance from B to A, and triangle inequality $|x + y| \leq |x| + |y|$ holds. However, with internet delays this is not always the case, and therefore Vivaldi algorithm will not achieve perfect results.

In a conclusion, the algorithm converges fast, but actual results are not so good.

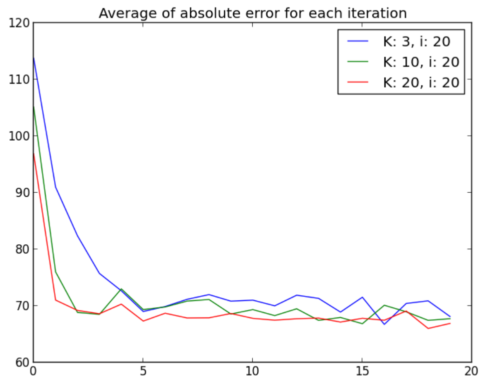
Appendix A: Graphical representation of results



(a) CDF

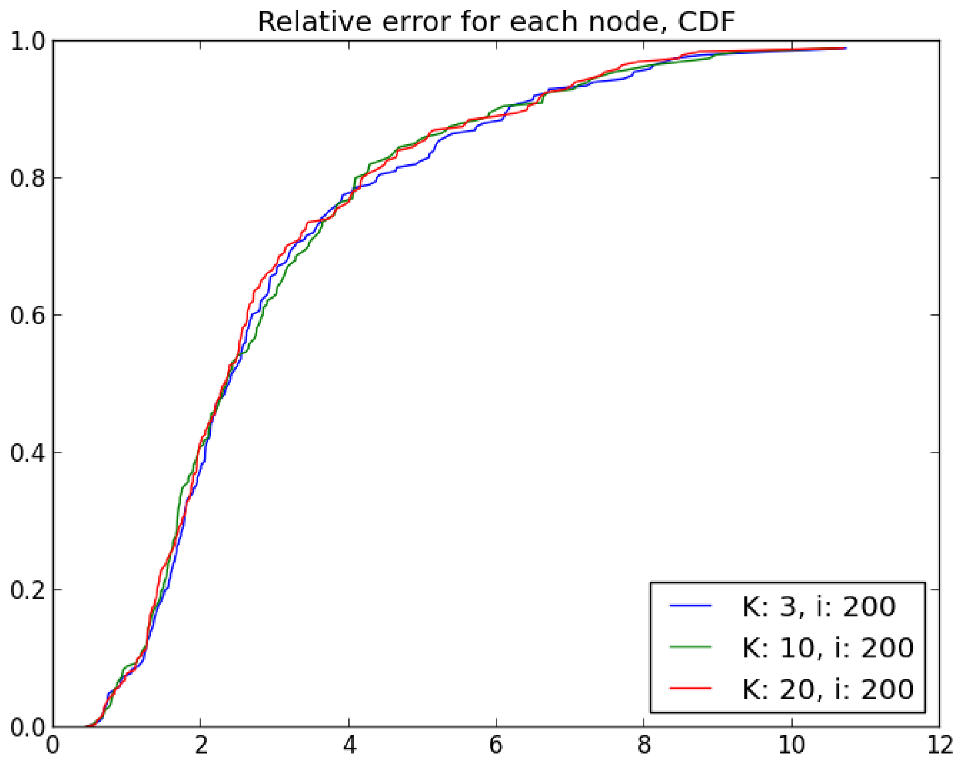


(b) Average move length (ms) per iteration

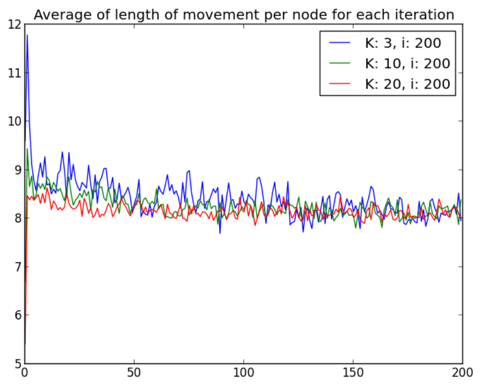


(c) Absolute average error (ms) per iteration

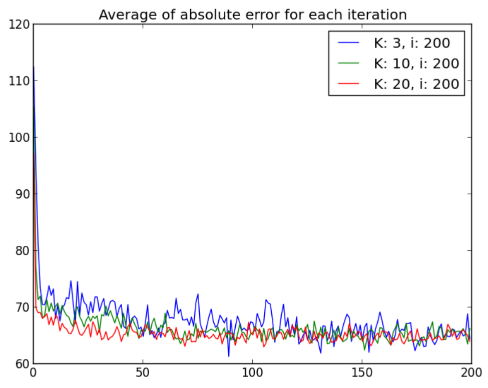
Figure 1: Simulations with 20 iterations



(a) CDF

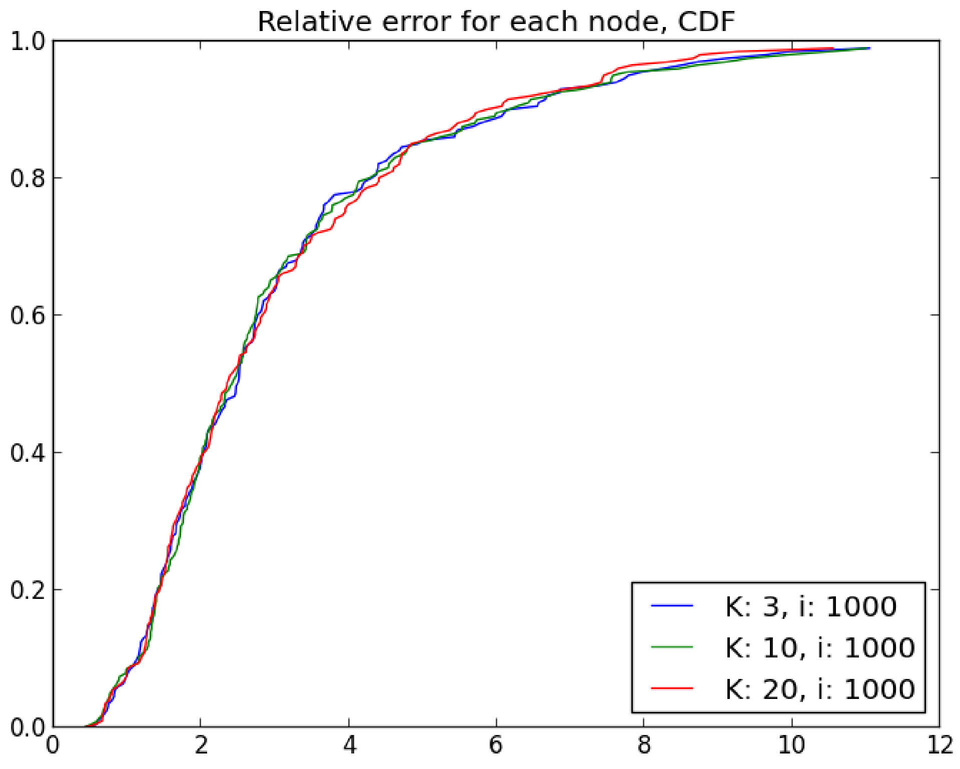


(b) Average move length (ms) per iteration

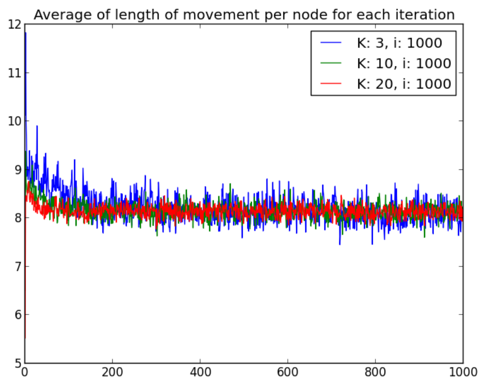


(c) Absolute average error (ms) per iteration

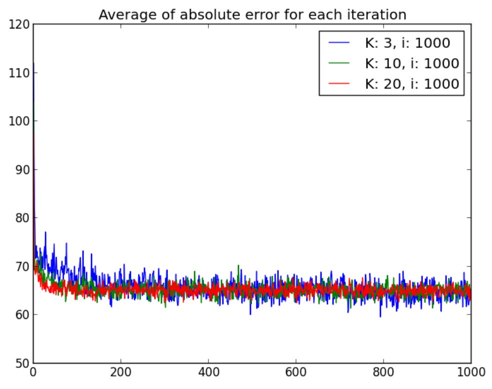
Figure 2: Simulations with 20 iterations



(a) CDF



(b) Average move length (ms) per iteration



(c) Absolute average error (ms) per iteration

Figure 3: Simulations with 20 iterations