Vivaldi simulation

Federico Giraud
Mattia Mantello

Abstract
This document provides an analysis of the Vivaldi algorithm simulation. First of all, a brief discussion of the input matrix is presented. Then main characteristics of the implementation are explained and finally the results are given.

federico.giraud@eurecom.fr
mattia.mantello@eurecom.fr

Contents

1. Input matrix

The input matrix used in this test is a 200x200 matrix containing the round trip time measured between 200 nodes. To simulate the behaviour of algorithms in real world situations, the data is typically collected from a real subset of internet nodes. As such, the round trip time between nodes can vary because of nodes and links saturation, malfunctioning ecc.

1.1 Triangle inequality

One of the main requirements of a metric is triangle inequality:

$$d(x,z) \leq d(x,y) + d(y,z) \quad \forall x,y,z \in N \tag{1}$$

where N is the set on nodes.

The main limitation of Vivaldi coordinates (related to prediction precision) is that nodes are represented as points in a multidimentional space, and the round trip time prediction is done only with node distances. As such, triangle inequality violations play an important role in the algorithm predicting precision, as it is not possible to build point coordinates that violate metric conditions.

In order to roughly estimate the amount of prediction error we should expect, a small analysis can be done. We considered all possible 3-tuples $(x,y,z) \subset N$. For each tuple we calculate the amount of triangle inequality error (2).

$$e = max(0, d(x,z) - (d(x,y) + d(y,z))) \tag{2}$$

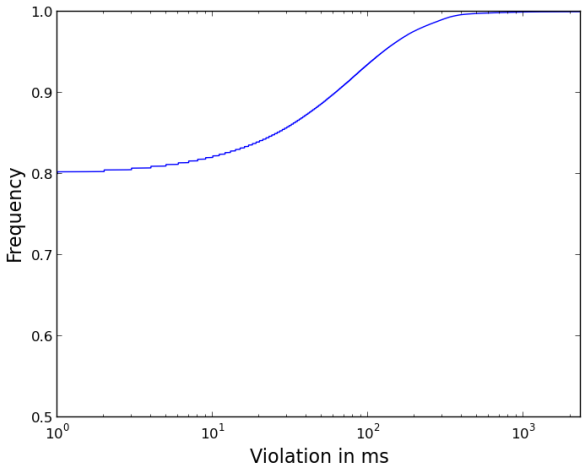

Figure 1. Triangle inequality violations

As we can see in Figure 1, the triangle inequality is violated for 20% of tuples. The average error in this 20% is 95.8 ms and the maximum is 2338 ms. For example, distances of nodes in the tuple $(81, 170, 165)$ are:

- $d(81,165) = 2387ms$
- $d(81,170) = 33ms$
- $d(165,170) = 16ms$

According to those measurements, a packet travelling from node 81 to node 165 will take 2387 ms to come back, while the same packet will take only 33 + 16 = 49 ms if the packet travels also through node 170. Even thought this measures could be correct (for example the link between nodes 81 and 165 could be congested), they do not follow the triangular inequality and can deteriorate the algorithm performance.

Table 1. Simulation results

	Node movements				Relative error						
	min	*avg*	*max*	*var*	*min*	*avg*	*max*	*var*	*0.5*	*0.90*	*0.99*
Neighbours = 3											
Iterations = 20	1.99	7.47	13.46	11.19	0.45	4.87	17.94	14.18	3.62	10.11	17.77
Iterations = 200	0.28	2.82	33.15	20.04	0.77	5.98	30.57	23.57	4.35	12.83	24.89
Iterations = 1000	0.02	1.09	29.19	5.67	0.75	5.98	30.90	22.33	4.77	12.60	20.97
Iterations = 2000	0.05	0.69	27.31	2.04	0.66	4.80	19.42	13.61	3.54	10.62	19.41
Neighbours = 10											
Iterations = 20	4.34	13.10	54.40	211.93	0.56	4.10	15.43	8.13	3.39	7.95	13.87
Iterations = 200	0.84	3.06	41.04	12.13	0.62	3.94	15.15	8.22	3.07	7.98	13.75
Iterations = 1000	0.13	1.94	52.77	4.66	0.44	3.85	12.45	6.69	3.22	7.98	12.12
Iterations = 2000	0.17	3.86	26.71	3.87	0.42	3.37	12.69	5.09	2.63	6.52	11.72
Neighbours = 20											
Iterations = 20	0.54	6.90	70.96	225.61	0.51	3.48	12.43	5.50	2.67	7.26	10.46
Iterations = 200	1.58	5.15	49.27	12.32	0.50	3.73	14.38	6.92	2.85	7.60	13.75
Iterations = 1000	0.36	3.33	65.89	7.06	0.44	3.54	14.91	6.50	2.69	6.95	13.19
Iterations = 2000	0.18	3.71	41.76	3.68	0.39	3.39	12.14	5.40	2.70	7.17	10.68

1.2 Symmetry

Another requirement of a metric is symmetry:

$$d(x,y) = d(y,x) \quad \forall x,y \in N \tag{3}$$

In our input matrix, the 1.36% of all possible couples $(x,y) \in N$ doesn't satisfy this condition. The average of $|d(x,y) - d(y,x)|$ in those node was of 65 ms.

1.3 Input matrix considerations

Violations of triangle inequality will play an important role in the algorithm prevision capability, because a big number of nodes is affected, with large rtt differences. Symmetry violations are less important because they affect only a small amount of nodes.

2. Algorithm description

2.1 Parameters

We implemented the standard Vivaldi algorithm with adaptive timestamp based on node error prediction. The parameters are:

- $\delta = 0.25$
- $c_c = c_e = 0.25$
- $dimesions = 3$

2.2 Node initialization

The first simulations were executed with node starting positions in zero and randomly chosen directions for the unit-length vectors, as suggested in the document [1]. Anyway doing in this way, the relative error at first iterations is always close to one (because the predicted rtt is close to 0).

To avoid this fake low error at the beginning, we decided to start nodes coordinates with random values, chosen in the interval $[0,300]$, that is an approximately sufficient size area in order to contain the majority of nodes of our input matrix.

2.3 Neighbours

For the majority of P2P systems the neighbour set is defined at the beginning and usually it doesn't change much during time. So, for this simulation we decided to randomly select node neighbours at the beginning and keep the same set for the whole algorithm execution time.

3. Results

3.1 Node movements

We measured the difference between each node position before and after each iteration. This distance represents how much the node was moved in one iteration in order to better represent nodes real round trip time. In figure 2 we can see the amount of node movements (the average of all nodes) during different iterations, for 3 different simulations (all starting with 10 neighbours). As we can see, at the beginning nodes movement are much bigger, from 15 to 45, they decrease in about 20 iterations, and then continue oscillating in next iterations. Minimum, maximum, average and variance of node movements are given in table 1.

As we expected from the input matrix analysis, nodes continue moving trying to fit RTTs, but triangle inequality violations prevent them from stabilizing. In table 1 node movements are represented, related to number of neighbours and iterations. As expected, the average movement distance tends to decrease as we increase the number of iterations, even thought nodes never stop moving.

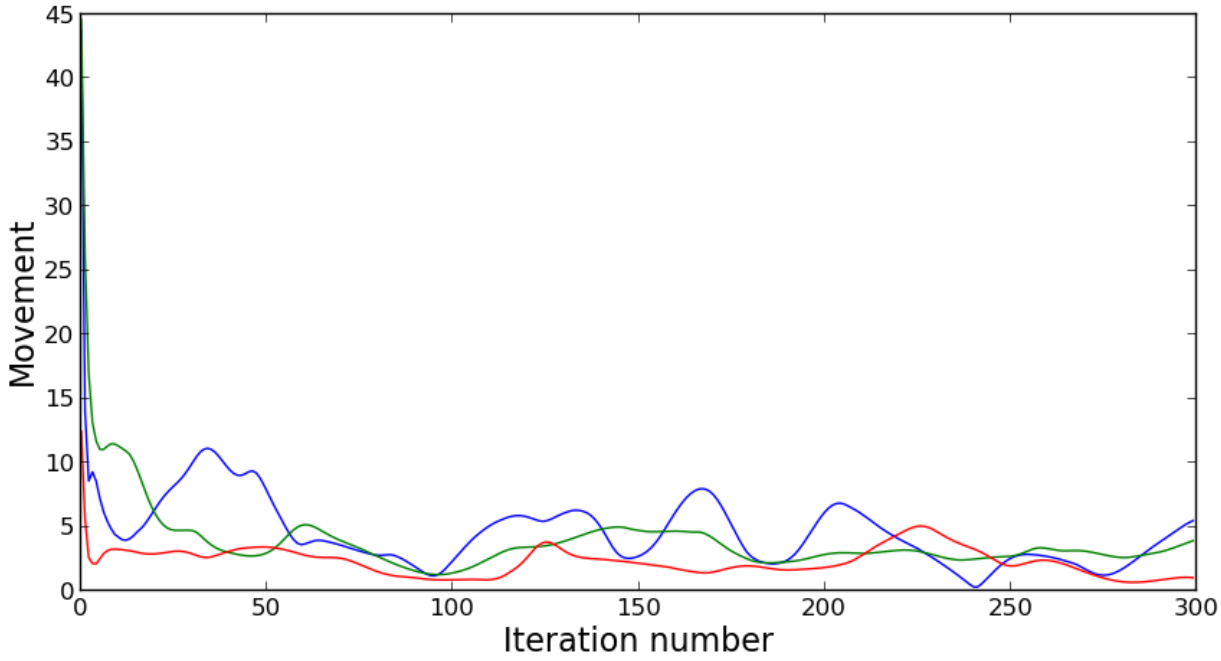

Figure 2. Node movements

3.2 Prediction error

For every node we calculated the average relative error on each link between two nodes. So, for each node $i \in N$, the error e_i is expressed in the formula (4):

$$e_i = avg_j(|\|x_i - x_j\| - rtt_{ij}|/rtt_{ij}) \quad \forall j \in Neigh_i \qquad (4)$$

Values of e_i are shown in table 1 and CDFs are represented from Figure 4 to Figure 12 (percentile of relative prediction errors are shown in the table). As we can see, the average relative error stabilizes around 3.4 in 10 and 20 neighbours simulations. Usually the average relative error has a local minimum after a small number of iterations. This behaviour is better shown in Figure 3, that represents the average relative error of the 20 neighbours simulation related to the iteration number. As we can see, the error goes from 7.8 after the first iteration to 3.11 just after 3 iterations, and then increases and stabilizes after more iterations. The results vary a lot from simulation to simulation: in some cases this local minimum is also the global one.

In general, the algorithm converges, even if it converges to a high error values.

3.3 Input matrix test

We tested the algorithm with another 200x200 input matrix, created from distances of random positioned nodes, with a small amount of random noise in each distance measurement. For that matrix, 1% of node links violate the triangle inequality. In this case the algorithm performed much better, even with a low neighbour number, with a relative error of 0.05 after few iterations.

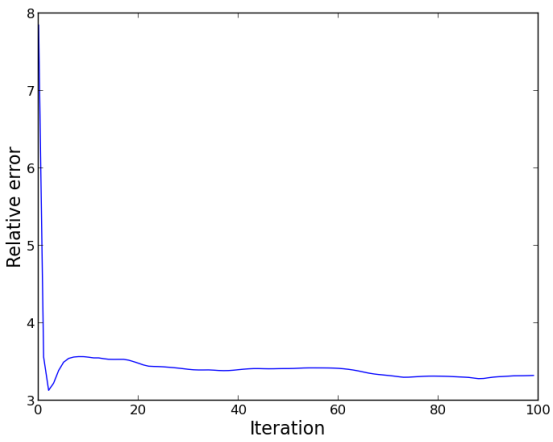

Figure 3. Prediction errors for iteration

4. Conclusion

The algorithm performance is highly related to the input data: if the triangle inequality is violated in a large number of nodes, the algorithm will result in big approximation errors and will require a big number of iterations to converge (as shown in table 1, the error is still decreasing after 1000 iterations).

If the input measurements have low triangle inequality violations, the algorithm produces good approximations even after a small number of iterations.

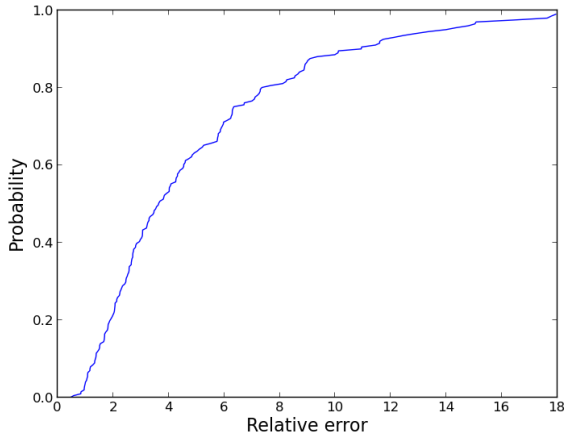

Figure 4. CDF of $e_i : n = 3, i = 20$

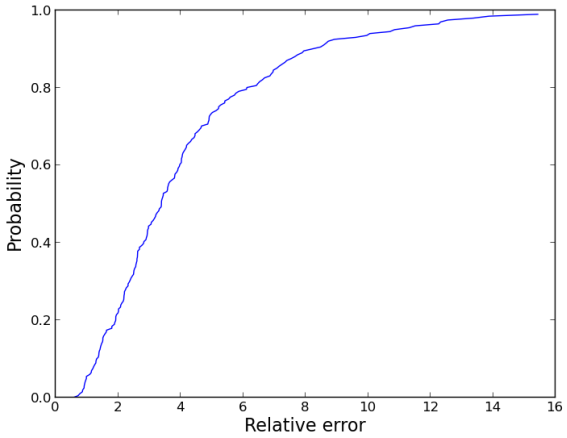

Figure 7. CDF of $e_i : n = 10, i = 20$

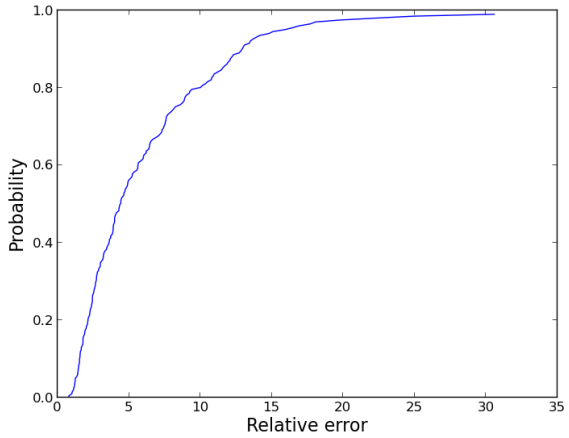

Figure 5. CDF of $e_i : n = 3, i = 200$

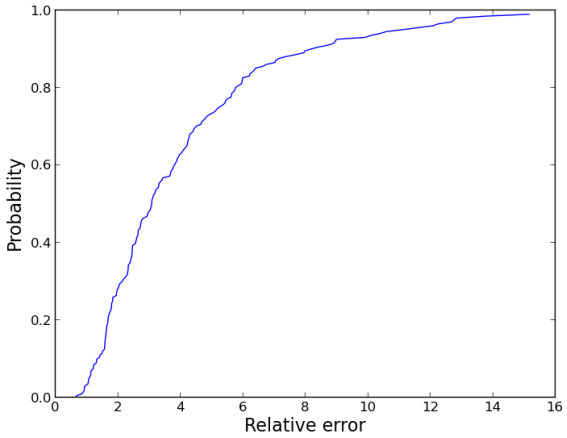

Figure 8. CDF of $e_i : n = 10, i = 200$

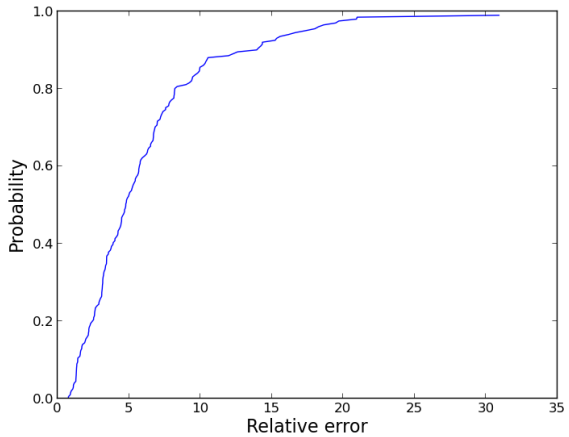

Figure 6. CDF of $e_i : n = 3, i = 1000$

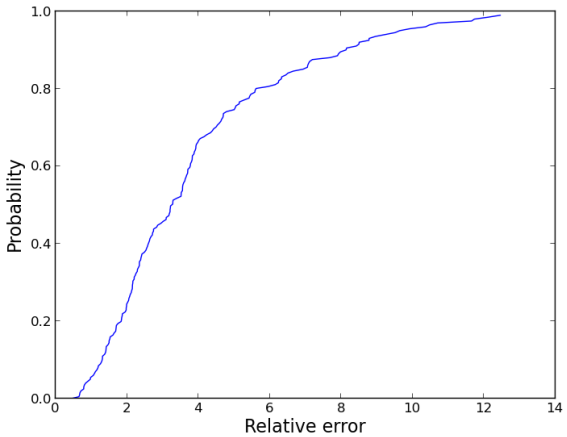

Figure 9. CDF of $e_i : n = 10, i = 1000$

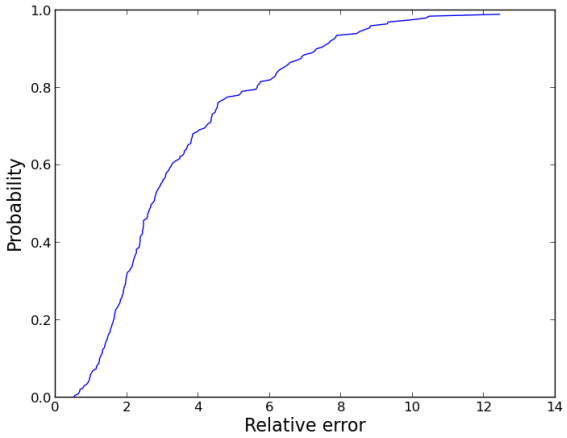

Figure 10. CDF of $e_i : n = 20, i = 20$

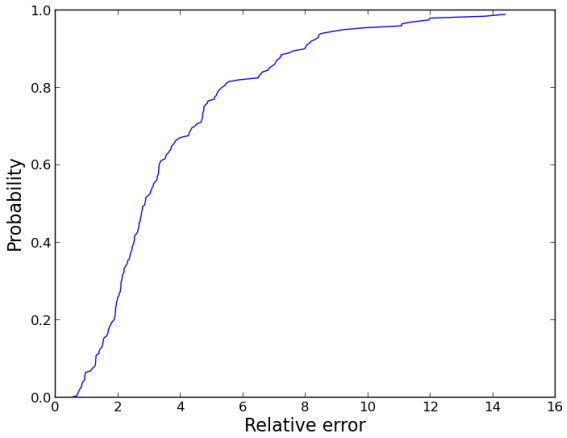

Figure 11. CDF of $e_i : n = 20, i = 200$

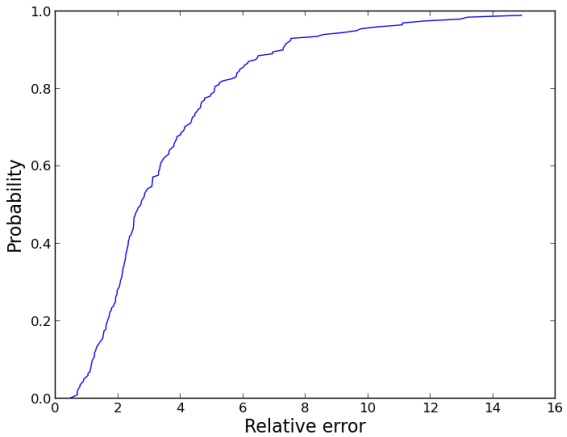

Figure 12. CDF of $e_i : n = 20, i = 1000$

References

[1] Frank Dabek, Russ Cox, Frans Kaashoek, Robert Morris, *Vivaldi: A Decentralized Network Coordinate System*. MIT CSAIL Cambridge, MA