Lab 6 - CSN: Network dynamics

Pierre-Antoine Porte
porte.pierreantoine@gmail.com

Rodrigo Arias Mallo
rodarima@gmail.com

December 15, 2017

1 Introduction

In this session, we had to model 3 different models following the dynamical principles of the Barabasi-Albert model. Those principles are: vertex growth and preferential attachment. The different models we needed to implements were the following:

- Barabasi-Albert with the dynamical principles
- Barabasi-Albert with random attachment instead of preferential attachment (only one dynamical principle).
- Barabasi-Albert with vertex growth suppressed (only one dynamical principle).

Those models were simulated and the data kept in files so we could analyze mathematical properties of those models.

In this report we will show our results regarding those models and their analysis. We will discuss those results and explain them. Then we will explain how we implemented the model simulations and what we used to analyze it in the Methods section.

2 Results

For table 1, the columns represents the models where respectively:

- A is Barabasi-Albert
- B is with random attachment instead of preferential
- C is without vertex growth

	A	B	C
1	96.625	95.676	0.000
2	52.683	0.000	7.687
3	62.506	112.933	108.152
4	32.408	104.567	108.202
5	0.000	102.636	110.078

Table 1: AIC measures for the degree distribution.

	A	B	C
0	69 534.390	59 969.865	10 786.775
1	45 188.426	48 383.218	38 204.833
2	0.000	4088.919	33.872
3	57 486.358	23 686.609	65 632.205
4	57 130.622	10.271	49 999.142
0+	45 767.062	22 835.391	4019.623
1+	27 730.356	25 665.773	26 144.944
2+	19 115.101	10 442.574	0.000
3+	∞	∞	∞
4+	12 117.889	0.000	9608.871

Table 2: AIC measures for the vertex degree over time for $t_i = 1$.

	A	B	C
0	56 219.491	56 948.832	3294.824
1	33 958.147	44 276.525	35 679.447
2	5638.288	4262.024	77.463
3	80 808.086	75 603.323	65 467.831
4	41 881.941	5870.707	48 564.528
0+	33 417.747	22 409.635	1463.430
1+	17 922.167	15 043.736	23 894.852
2+	11 109.884	9443.397	0.000
3+	∞	∞	∞
4+	0.000	0.000	4786.908

Table 3: AIC measures for the vertex degree over time for $t_i = 10$.

	A	B	C
0	46 992.180	56 158.051	6775.516
1	26 745.892	40 993.239	34 445.263
2	4084.008	11 373.071	1083.024
3	70 608.550	76 491.996	62 248.713
4	30 090.842	24 884.091	46 399.114
0+	22 011.767	27 947.947	343.071
1+	10 879.865	20 065.047	20 084.117
2+	3419.044	14 228.805	0.000
3+	∞	∞	∞
4+	0.000	0.000	1983.962

Table 4: AIC measures for the vertex degree over time for $t_i = 100$.

	A	B	C
0	24 212.569	24 852.123	16 241.758
1	8639.541	4706.668	35 197.595
2	4130.729	4019.593	465.222
3	58 036.196	55 068.283	69 970.214
4	31 443.880	27 424.662	49 730.948
0+	0.021	10 189.872	2862.488
1+	5672.115	3420.122	13 800.604
2+	0.000	1158.596	424.130
3+	∞	∞	∞
4+	159.748	0.000	0.000

Table 5: AIC measures for the vertex degree over time for $t_i = 1000$.

Dataset	A	B	C
1λ	1.593	1.593	10.000
$2q$	0.500	0.500	0.100
4γ	2.189	2.078	2.000
5γ	2.000	2.000	2.000
$5k_{\max}$	20.000	20.000	20.000

Table 6: Parameters for degree distribution models fitting.

Dataset	A	B	C
$0a$	0.005	0.002	0.001
$1a$	0.434	0.134	0.110
$2a$	1.616	3.348	0.000
$2b$	0.350	0.133	1.166
$3a$	14.000	8.637	0.000
$3c$	0.000	0.000	−0.527
$4a$	3.792	1.222	0.837
$4d_1$	−0.849	1.117	−0.871
$0+a$	0.003	0.000	0.002
$0+d$	14.000	8.496	−0.735
$1+a$	0.370	0.072	0.178
$1+d$	5.000	5.000	−5.000
$2+a$	0.500	0.413	0.000
$2+b$	0.466	0.300	1.155
$2+d$	5.000	5.000	−0.058
$3+a$	0.289	0.289	0.289
$3+c$	0.853	0.853	0.853
$3+d$	−10.000	−10.000	−10.000
$4+a$	12.971	1.230	48.860
$4+d_1$	952.161	1.413	27 177.319
$4+d_2$	−80.698	−0.060	−500.000

Table 7: Parameters for the vertex degree over time for $t_i = 1$.

Dataset	A	B	C
$0a$	0.003	0.001	0.001
$1a$	0.237	0.093	0.087
$2a$	1.036	1.795	0.001
$2b$	0.330	0.159	1.070
$3a$	−0.330	−0.331	−0.352
$3c$	−1.363	−1.363	−0.566
$4a$	2.079	0.820	0.693
$4d_1$	−9.843	−9.408	−9.875
$0+a$	0.001	0.000	0.001
$0+d$	10.366	5.424	−0.220
$1+a$	0.172	0.036	0.139
$1+d$	4.885	4.303	−3.958
$2+a$	0.500	0.314	0.000
$2+b$	0.398	0.300	1.088
$2+d$	2.325	2.893	0.076
$3+a$	0.289	0.289	0.289
$3+c$	0.853	0.853	0.853
$3+d$	−10.000	−10.000	−10.000
$4+a$	5.606	0.973	47.430
$4+d_1$	364.909	0.000	37 522.690
$4+d_2$	−30.642	−1.282	−500.000

Table 8: Parameters for the vertex degree over time for $t_i = 10$.

Dataset	A	B	C
$0a$	0.001	0.001	0.001
$1a$	0.083	0.064	0.074
$2a$	0.422	0.735	0.000
$2b$	0.313	0.220	1.122
$3a$	−0.500	−0.500	−0.500
$3c$	−0.500	−0.500	−0.500
$4a$	0.718	0.563	0.568
$4d_1$	−15.000	−15.000	−15.000
$0+a$	0.000	0.000	0.001
$0+d$	3.674	3.410	−0.463
$1+a$	0.059	0.033	0.128
$1+d$	1.864	2.366	−4.056
$2+a$	0.498	0.289	0.001
$2+b$	0.300	0.300	1.047
$2+d$	−0.328	1.054	−0.304
$3+a$	0.289	0.289	0.289
$3+c$	0.853	0.853	0.853
$3+d$	−10.000	−10.000	−10.000
$4+a$	1.772	0.938	46.936
$4+d_1$	209.777	0.000	41 725.087
$4+d_2$	−8.984	−3.152	−500.000

Table 9: Parameters for the vertex degree over time for $t_i = 100$.

Dataset	A	B	C
$0a$	0.000	0.000	0.001
$1a$	0.036	0.034	0.120
$2a$	0.016	0.025	0.005
$2b$	0.596	0.537	0.864
$3a$	-0.500	-0.500	-0.500
$3c$	-0.500	-0.500	-0.500
$4a$	0.306	0.300	1.013
$4d_1$	-15.000	-15.000	-15.000
$0+a$	0.000	0.000	0.001
$0+d$	0.925	1.003	0.928
$1+a$	0.041	0.037	0.183
$1+d$	-0.360	-0.224	-4.921
$2+a$	0.000	0.334	0.004
$2+b$	1.001	0.300	0.880
$2+d$	0.926	-1.835	0.132
$3+a$	0.289	0.289	0.289
$3+c$	0.853	0.853	0.853
$3+d$	-10.000	-10.000	-10.000
$4+a$	16.509	1.460	48.389
$4+d_1$	50 000.000	743.241	31 025.620
$4+d_2$	-177.765	-10.149	-500.000

Table 10: Parameters for the vertex degree over time for $t_i = 1000$.

3 Discussion

3.1 Simulation of the models

As explained in section Methods, we used m_0 and n_0 always with the same values (which can be however different for each model). We never compared the models with different m_0 or n_0 for the same model. It could have indicate us if the model behave differently given different graph in input. We could have went further and do this but we preferred to focus on the analysis of our different models with the input specified in the methods section.

3.2 Barabasi-Albert

3.3 Vertex degree over time

We checked if the power-law dependency with 1/2 exponent gives the best fit to all the time series as required by the statement. This power-law dependency is not the best fit. This power-law is model 1, which has an AIC higher than model 2 (see tables 2 to 5. In fact the best model is model 2, defined as $a * t^b$, which is the exponential growth. It makes sense because with one free parameter we can better optimize this model to fit the data, by adapting b to somewhat close to 1/2 as expected in model 1.

However if we look at table 7 to 10, we see that $b \approx 0.5$ only for $t_i = 1000$ (table 5).

3.4 Degree distribution

3.5 Barabasi-Albert without preferential attachment

3.6 Vertex degree over time

3.7 Degree distribution

3.8 Barabasi-Albert without vertex growth

3.8.1 Vertex degree over time

As expected by statement, this model's scaling vertex degree over time should fit a linear scale. By computing the AIC (see tables 2 to 5) we have seen that the linear model was good. Also, we are really confident by saying it's linear when we look at the plot generated for model3, for every $t_i s$. However we find that model0, 0+, 2 and 2+ are the best. Sometimes it's 0(+), sometimes it's 2(+). Model 2 is represented as at^b, as b is close to 1 as we can see in tables 2 to 5, which explain this good fit for model 2 and 2+.

3.8.2 Degree distribution

As stated by the statement, the degree distribution for this model should be closer to a binomial distribution. Indeed it looks like it, but we found out it looked even more of a displaced poisson with on a different scale. If we took $\lambda = 2$, then we would have a fastly increasing and decreasing poisson which is what we want. However, this Poisson has a mean of $\lambda = 2$. Therefore, it does not fit our data which has a mean of approximately 10. So we displaced the poisson and adjusted the scale. The lasting problem was that due do this scale and displacement, our model never produces data ≈ 0 whereas the generated model data had a lot of values ≈ 0.

In the end we chose to use a poisson distribution as we did in lab2. It's not a really good fit but it's still the best fit we have. You can see this plot in figure

We also made sure that the distribution giving the best fit in not a power-law, it's looking more of a Gausian one. However since it's not symetric it was hard to model the data using a normal law, for example.

4 Methods

For the default *Barabasi-Albert* and model with random attachment the initial graph was an empty graph with only one vertex.

For the model without vertex growth, we used an unconnected graph with t_max vertices. Because we have no vertex growth, the vertices are not increasing and $n_0 = n_tmax$.

For the three models we used $m_0 = 0$ to used "*clean*" and "*empty*" graphs.

We measured the growth of the vertex degree over time and the degree distribution for each model. The vertex degree was measured over the time for $t_i = 1, 10, 100, 1000, 10000$ successively.

We used python for generating the models, to store the results and to analyze the data. For each BA model M a folder in `data/model`M`/` contains all the results produced from this model. Inside, the degree sequence is stored in the file `dseq.txt`, the degree distribution in `dd.txt` and for each T in the arrival time, we produced `dt_`t_i`.txt` tracing the degree of the vertex arriving at time t.

4.1 Generating model with preferential attachment

While trying to define how to make the preferential attachment for model 3 we faced a problem. We were choosing from the edges to be linked to our vertex in an array of probability p, with the degree of the node over the sum of the degrees of all nodes as:

```
    p[i] = vertex.degree()/sum(graph.all_degrees())
```

However, with this, if you had m0 = 5 and only 4 vertices with a degree ¿ 0, then you could not choose your 5 vertices, and it did not work.

What we needed to do is counting stubs instead of degree, with a virtual stub for each unconnected vertex. Therefore our array of probability p was computed as:

```
if vertex.degree == 0:
    p[i] = 1 / sum(graph.all_degrees()) + sum(
                                       number_of_nodes_with_degree_0
                                       )
else:
    p[i] = vertex.degree() / (sum(graph.all_degrees()) + sum(
                                       number_of_nodes_with_degree_0
                                       ))
```

In the end the degree was represented with the number of stubs.