Report I.

Xiaohui Li, Luhuan Wu

May 29, 2017

1. Introduction:

The task is to

1. Generate a simple directed configuration model given in-degree and out-degree distribution. Specifically, we also test:
    - the convergence of the algorithm to generate the bi-degree sequence that have the same sum, which is introduced as *Algorithm 2.1*.
    - the performances of the three different algorithm to transform the mnultigraph to the simple graph, which are listed in *Algorithm 2.2*..
2. Test whether beweenness centrality of the nodes are equivalent to the pageranke of the nodes.

2. Algorithms:

The algorithms below except 2.2.3 are from the paper Directed Random Graph with Given Degree Distributions.

2.1 Algorithm to generate bi-degree sequence that satisfies:

$$\sum D^{+} = \sum D^{-}$$

2.2 The algorithms to transfrom the multigraph to the simple graph:

2.2.1 Repeated algorithm: keep generating random dcm untill obtaining a simple graph.

2.2.2 Erased algorithm: remove self-loops and parallel edges of the dcm

2.2.3 Revised repeated algorithm: we modified the repeated algorithm so that it stops adding pairing the edges that result in self-loops or parallel edges. But we are not sure whether this approach is reasonable.

3.Procedures:

3.1 Pre-Anlaysis and calculations

First, we need to generate the bi-sequences that have the following powerlaw distribution:

$$P(D^+ = d^+, D^- = d^-) = P(Poisson(W^+) = d^+, Poisson(W^-) = d^-)$$
$$\text{where } (W^+, W^-) \text{ are jointly regularly varying.}$$
$$P(W^+ > x) = (\frac{x}{b})^{-\alpha}, \qquad x > b$$
$$P(W^- > x) = (\frac{x}{c})^{-\beta}, \qquad x > c$$
$$(\alpha, \beta > 1)$$
$$W^+ = a(W^-)^d$$

Second, we can calculate the relationship between the parameters from the joint and margin distribution:

$$\begin{aligned} P(W^+ > x) &= P(a(W^-)^d > x) \\ &= P(W^- > (\frac{x}{a})^{\frac{1}{d}}) \\ &= x^{-\frac{\beta}{d}}(ac)^{\beta} \\ &= x^{-\alpha}b^{\alpha} \end{aligned}$$
$$\Rightarrow d = \frac{\beta}{\alpha}, \qquad c = (\frac{b}{a})^{\frac{\alpha}{\beta}} \tag{1}$$

Furthermore, according to the conditions of Algorithm 2.1 to converge, the expectation of F and G should be equal. Conditional on W+ and W-, it suffices to let the expecation of W+ and W- to be equal. Thus, we have

$$p(W^+) = \frac{d}{dt}(1 - (\frac{x}{b})^{-\alpha}) = \frac{a}{b}(\frac{x}{b})^{-\alpha-1}$$
$$E(W^+) = \int_b^{\infty} xp(W^+)dx = \frac{\alpha b^{\alpha}}{\alpha - 1}b^{1-\alpha}$$
$$E(W^+) = E(W^-) \Rightarrow c = \frac{\alpha}{\beta}(\frac{\beta - 1}{\alpha - 1})b \tag{2}$$

By (1) & (2), we can derive

$$\begin{aligned} & d = \frac{\beta}{\alpha} \\ \text{If } \alpha \neq \beta, \qquad & b = (\frac{\alpha(\beta-1)}{(\alpha-1)\beta})^{\frac{\beta}{\alpha-\beta}} a^{\frac{\alpha}{\alpha-\beta}} \\ & c = (\frac{\alpha(\beta-1)}{(\alpha-1)\beta})^{\frac{\alpha}{\alpha-\beta}} a^{\frac{\alpha}{\alpha-\beta}} \end{aligned}$$
$$\text{If } \alpha = \beta, \text{ then } a = 1 \text{ and we could arbitralily choose } b = c. \tag{3}$$

So we only need to choose a, alpha and beta to initialize the model.

3.2 Progamming

We use python as our programming language and our programs develop upon the support of package *networkx* and its method *directed_configuration_model* specifically. We create 11 python files (visit code repo)

- PowerLawDistribution.py

  To generate the specific bi-sequences that have the specific powerlaw distribution.

- ValidDegree.py
  Input any bi-degree sequence, use algorithm 2.1 to make the sum of in-degree sequence equal to the sum of out-degree sequence.

- SDGErased.py.

  Input any bi-degree sequence, use algorithms 2.2.2, the erased algorithm, which removes the self-roops and parallel edges to get simple directed graph. This module also provides methods to plot the degree distribution and the graph, etc.

- SDGRepeated.py

  Input any bi-degree sequence, use algorithm 2.2.1, the repeated algorithm, which repeats generating the DCM untill obtaining a simple directed graph. This module also provides methods to plot.

  In actual implementation, we keep generating directed comfiguration model by calling method `directed_configuraion_model_revised` in DCMRevised.py until `flag == True`.

- DCMRevised.py

  We modified the source code of *directed_configuration_model* by chaning only the part of random pariing the 'stubs', which stops as long as there exist self-loops or parallel edges. The idea is to stop generating the graph once it is added a self-loop or a parallel edge.

  The random pairing in the original source code is:

  ```
  while in_stublist and out_stublist:
          source = out_stublist.pop()
          target = in_stublist.pop()
          G.add_edge(source,target)
  ```

  We moditied this part to be:

  ```
  while in_stublist and out_stublist:
          source = out_stublist.pop()
          target = in_stublist.pop()
          if source == target or G.edges().__contains__((source, target)):
              flag = False
              print(source, target)
              break
          G.add_edge(source,target)
  ```

- DCMRevised2.py

  Again, we modified the random pairing part in the source code as mentioned above. The idea is to seek for new pairing once the graph is added a self-loop of a parallel edge.

```
    while in_stublist and out_stublist:
        source = out_stublist.pop()
        target = in_stublist.pop()
        if source == target or G.edges().__contains__((source, target)):
            print((source, target))
            continue
        G.add_edge(source,target)
```

- DCMGenrator.py

  Input the algorithm type, using that algorithm to generate the Directed Configuration Model.

- BetweenessCentralityVSPageRank.py

  Input an DCM model, calculate its nodes' betweenness centrality and page rank. Compare the two results.

- FurtherTestDifferentPower.py

  To test the bi-degree sequnce convergence using *Algorithm 2.1*.

- test1.py & test2.py

  Feed the parameters a, b, alpha, beta to the file and generate the desired results including the directed graph model and relevent plots.

4. Results and Analysis

4.1 Test for Bi-degree Sequence Convergence

From the SLLN, we know that given the distribution F and G have the same mean value, as n goes to infinity, the sum of in-degree and out-degree should converge to zero, also the same-sum algorithm requires that the delta_n be small enough to "negligible" :

$$|\Delta n| \leq n^{1-\kappa+\delta_0}$$

We necessarily need the difference between in-degree sequence and out-degree sequence to decrease fast, i.e. the relative difference ratio less than 1:

$$\frac{|\Delta n|}{n} < 1$$

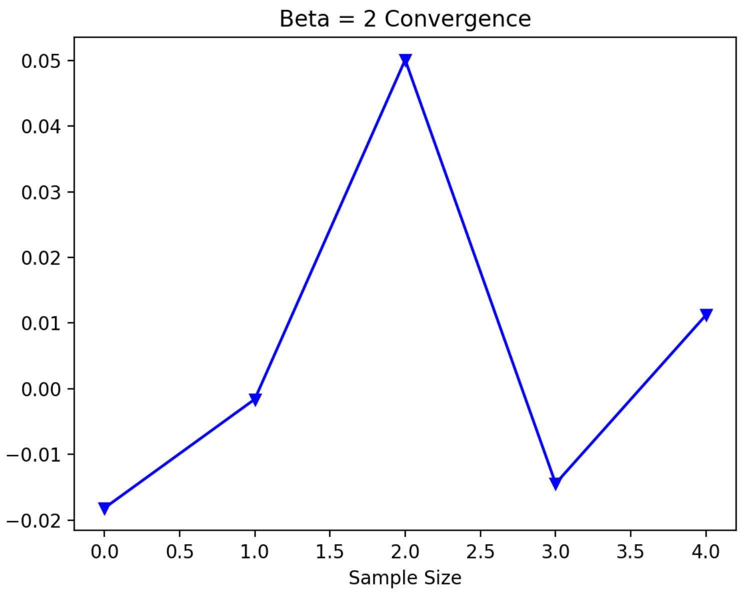
Given default parameter

$$a = 1; \alpha = 2$$

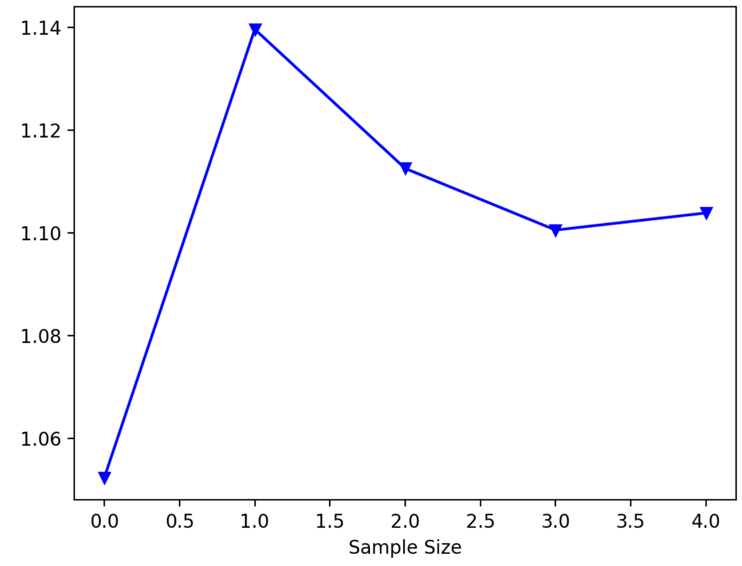
We adjust beta values to respective check the relative difference ratio, specifically we use the average delta_n from repeated sample as measurement . We set beta value: [2, 2.5, 3] and sample size n: [1000, 2000, ..., 5000]

The results show as below:

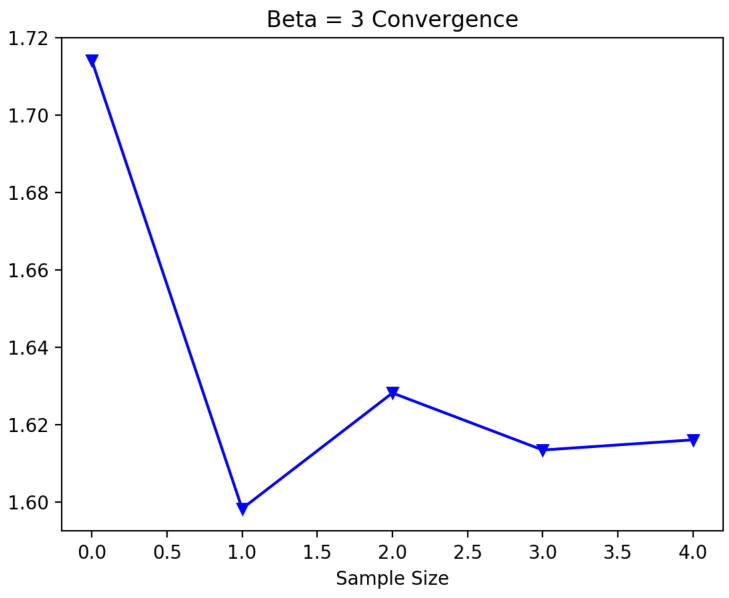
$$Case1 \qquad \beta = 2 \qquad\qquad (\alpha = \beta)$$



$$Case2 \qquad \beta = 2.5$$



$$Case3 \qquad \beta = 3$$



We could find that in the sample size from 1000 to 5000 scaling, there is not significant decrease.

In special case when alpha equals to beta, the delta_n shows good property, this denotes that the same-sum alogrithm performs well. While in other beta only 2.5 with sample size 1000 shows good sign, the rest are beyond 1, and this infers that the repeating sample to get delta_n may not work out in those cases.

4.2 Generated Bi-degree Sequence Simple DCM

Basically, we test 2 cases of parameter setting:

- Special Case(alpha = beta, which suggests F = G) Homogeneous Bi-degree:

$$a = 1, \alpha = 2, \beta = 2, b = 2$$

  By (3) we know that in this case, a always equal to 1.

- (alpha != beta, which suggests F != G) Inhomogeneous Bi-degree

4.2.1 Perfomances of 3 algorithms

We feed 2 sets of parameters which are:

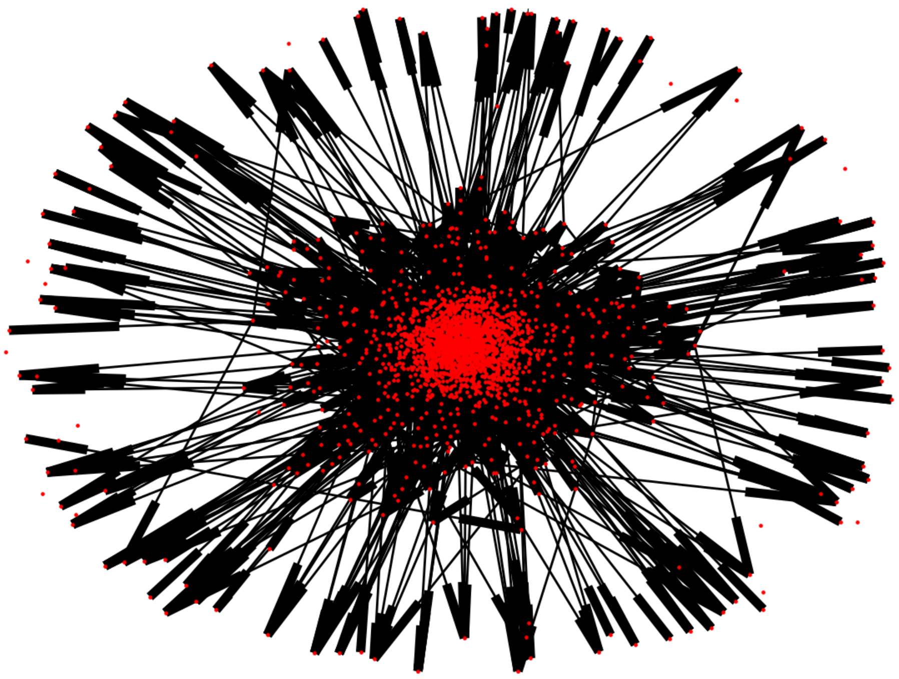
$$case1 : a = 1, \alpha = 2, \beta = 2, b = 2, n = 2000$$
$$case2 : a = 1, \alpha = 2, \beta = 2.5, n = 2000$$

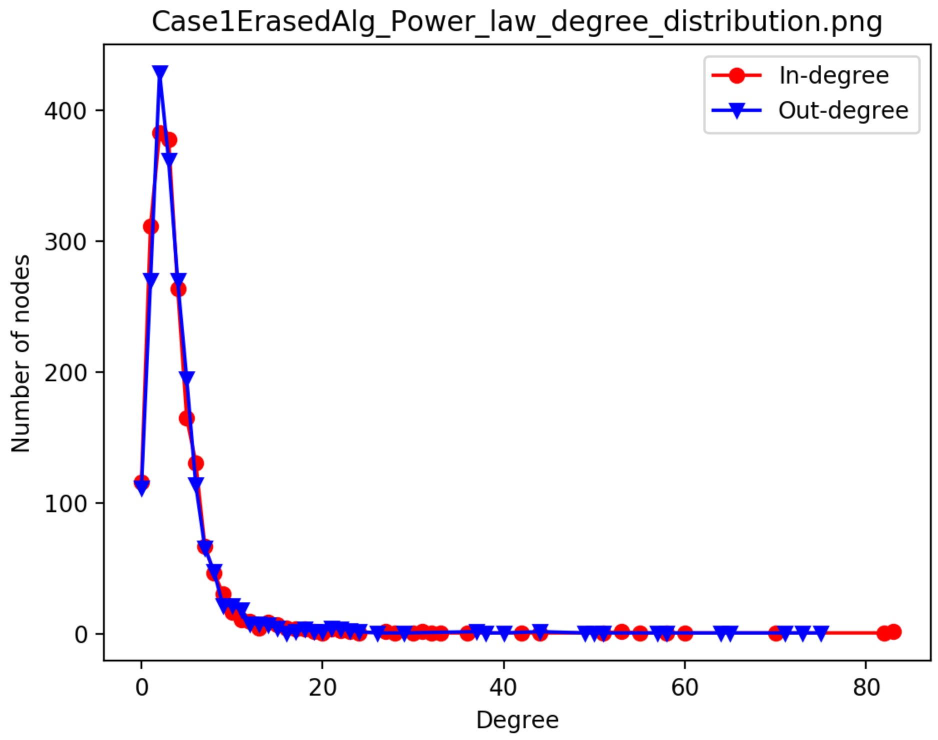
to Algorithms 2.2.1, 2.2.2, 2.2.3. We found that Algorithm 2.2.1, the Repeated Algorithm does not converge after running for an hour, whie the other two converges in less then 2 minuntes. The detailed running time is listed below:

Algorithm	case1 runtime(s)	case2 runtime(s)
2.2.2 Erased	36.275549	90.259211
2.2.3 Revised Repeated	58.632179	88.031426

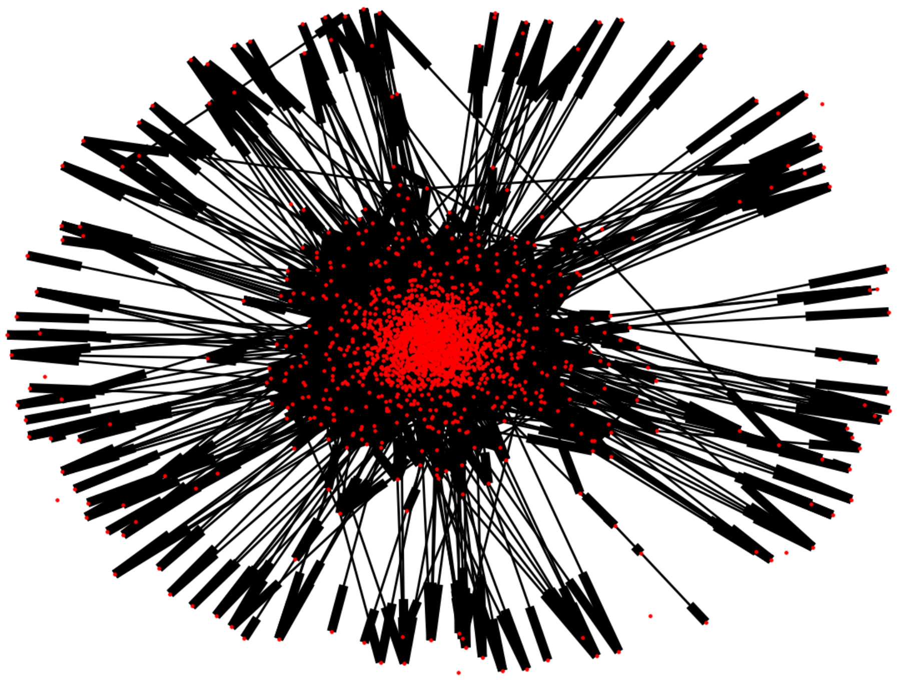
We also compare the degree distribution and the graph resulted by these two algotithms. For simplicity, we show the plots of case1.

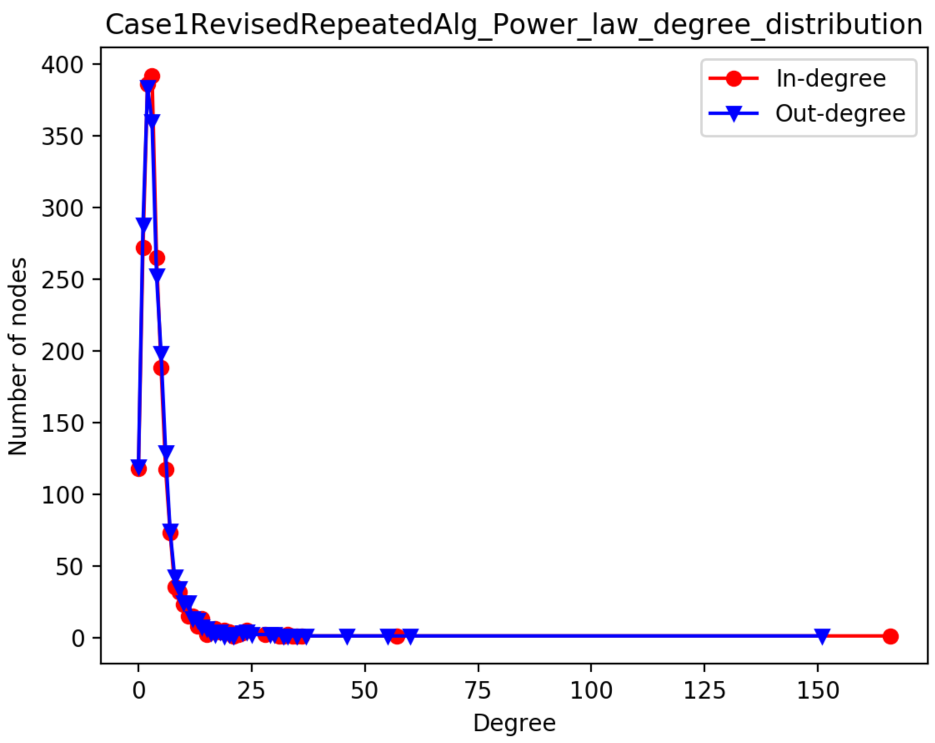
Erased Algorithm:





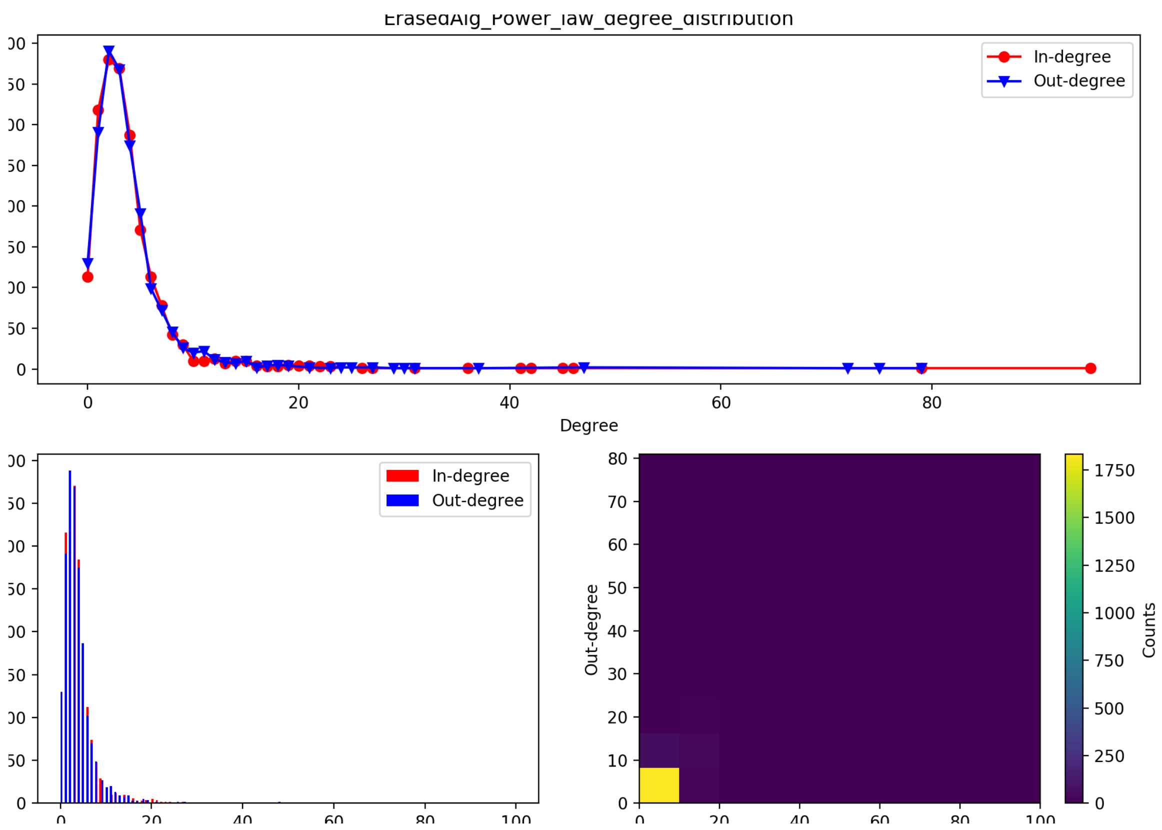
Revised Repeated Algorithm:





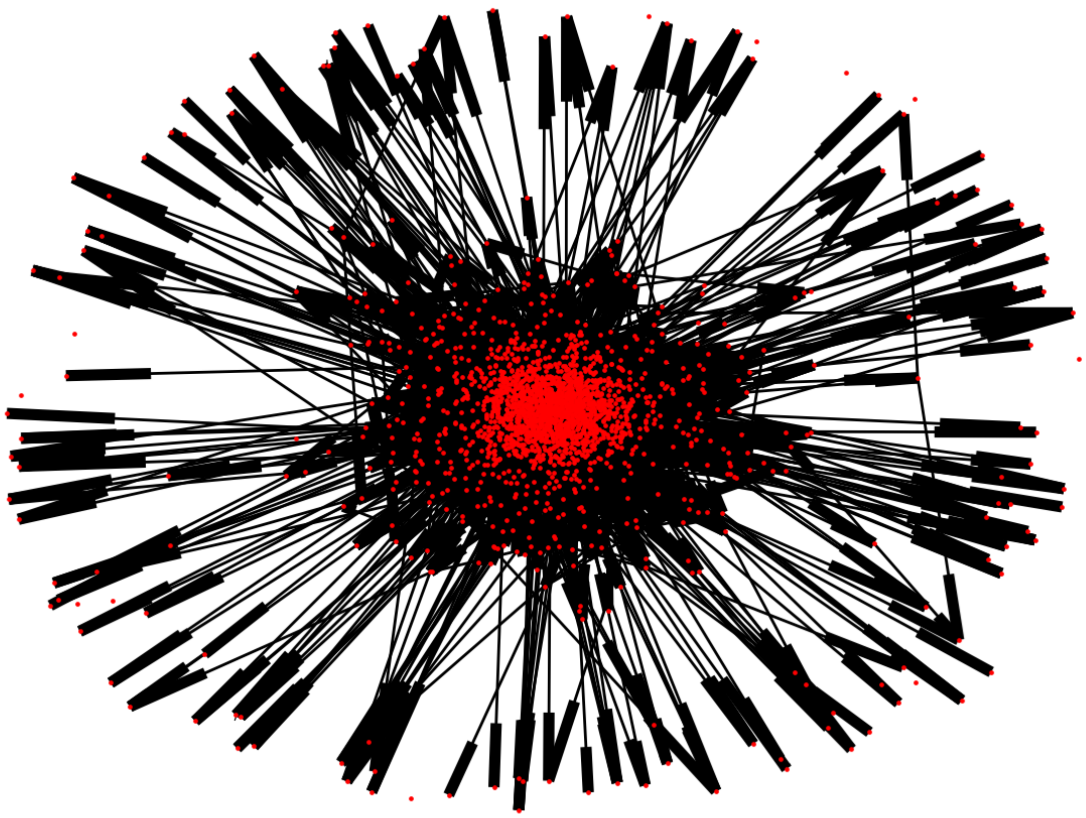
4.2.2 Further analysis baed on Erased Algorithm generated model

- Special Case (alpha = beta) Homogenuous Bi-degree
    - Degree Distribution

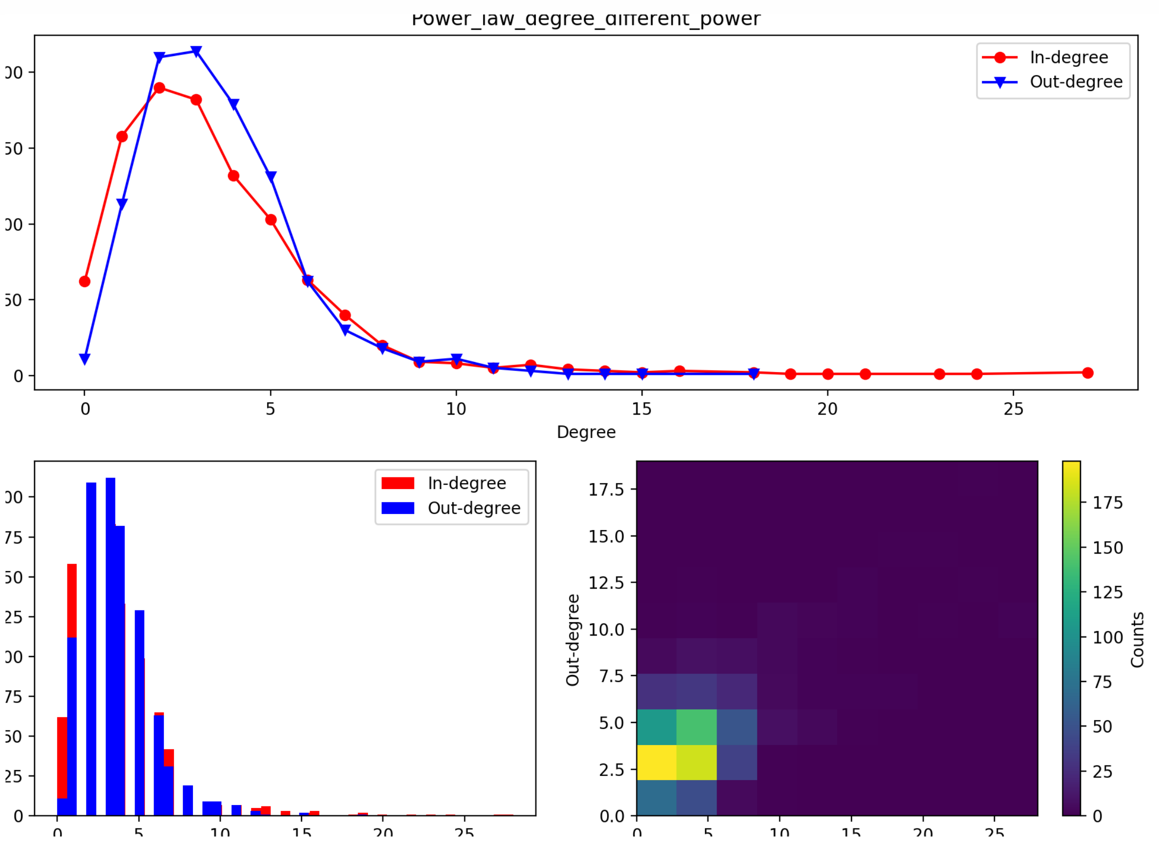


The modified degree sequence of the graph looks similar to the raw bi-degree sequence, most nodes concentrate in the 0-10 degrees range, the peak-larger part of degree shows power law distribution pattern.

- Graph

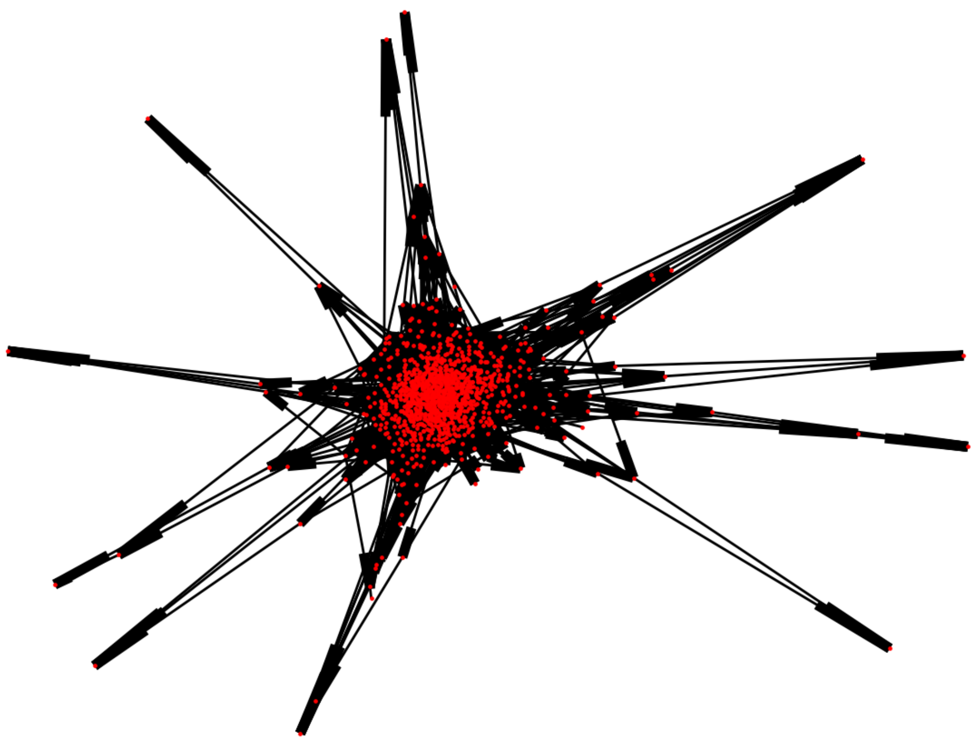


- Inhomogenuous Bi-degree sequence (F not equal to G)
    - Power inhomogenuous (beta = 2.5)
        - Degree distribution



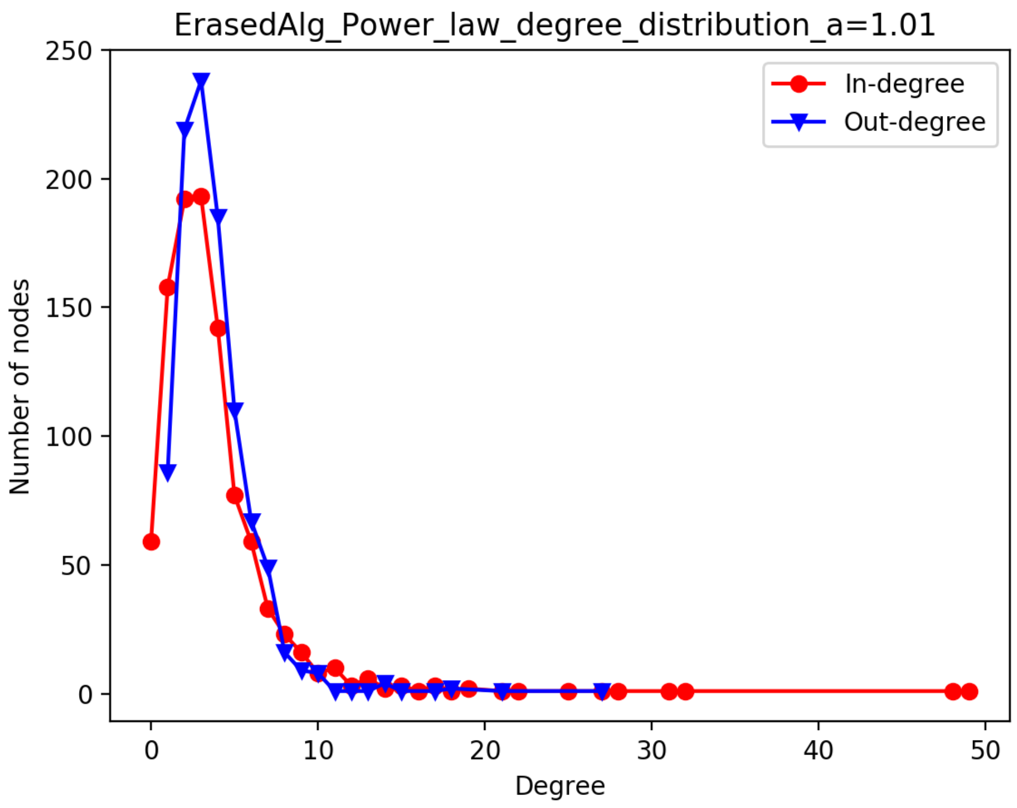
The out-degree seems to be more concentrated on the lower degree, with higher peak and larger proportion of lower degree.

- Graph

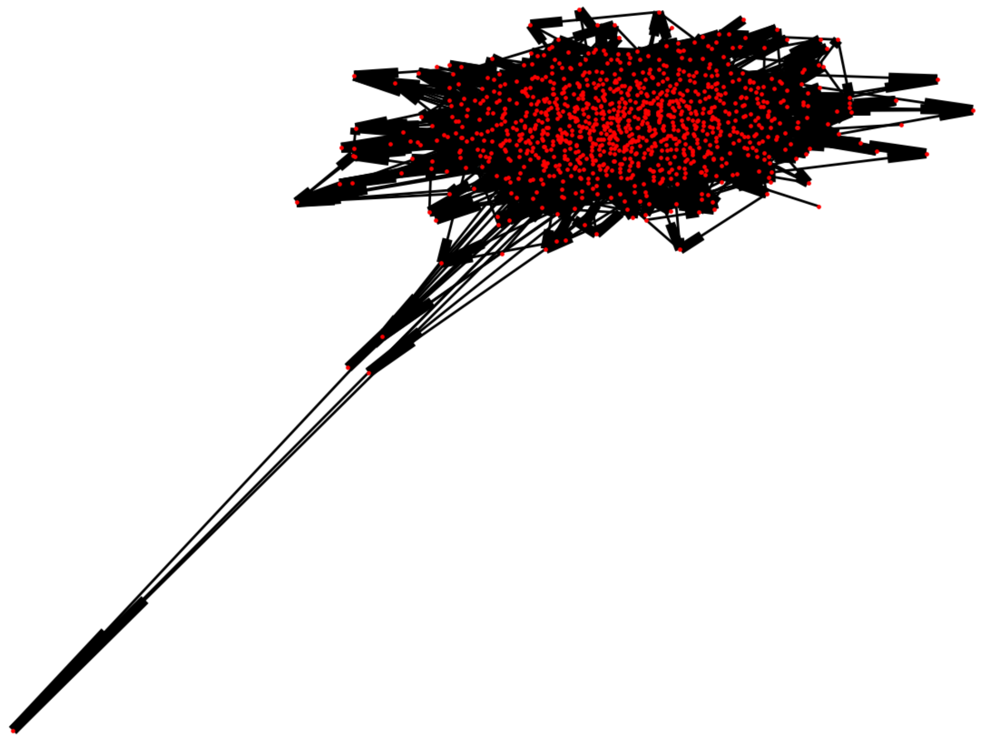


The graph shows that the nodes with large out-degrees (incident to many nodes) significantly decrease, corresponds to the distribution finding. Also as beta increases, less isolated nodes.

- Linear and power inhomogenuous (a = 1.01, beta = 2.5)
    - Degree distribution



    - Graph



In linear and power change, there is enhancing out-degree concentration effect.

4.3 Betweenness Centrality and Page Rank

Using the DCM model generated by *Erased Algorithm*, we calculate the BC(Beweenness Centrality) and PR(Page Rank) of the nodes. After re-sorting the results, we could check the top 20 nodes that have largest BC and PR.

- Homogeneous bi-degree sequence

$$a = 1, \alpha = 2, \beta = 2, b = 2$$

```
pr_sorted[0:20]
Out[26]:
[(208, 0.00863407370598 88),
(382, 0.007157200405385412),
 (935, 0.005870257218606968),
 (9, 0.005117351118500958),
 (1658, 0.004851104025769731),
 (1739, 0.004740581029246568),
 (618, 0.004305323547821 4946),
 (980, 0.004297698853385 3155),
 (1225, 0.0039596504224661076),
```

(**601**, 0.0037401860991799238),
(**1105**, 0.0035124005803981837),
(**650**, 0.0035076988508568377),
(**395**, 0.003373848022790835),
(**1915**, 0.003291773518914025),
(**923**, 0.003138025603751571),
(**560**, 0.0028197902507183454),
(1977, 0.0027952263952016193),
(**1765**, 0.0027095309598794766),
(**722**, 0.0026814121290670966),
(167, 0.002613429083497588)]
bc_sorted[0:20]
Out[27]:
[(**208**, 0.12990672212658236),
(**382**, 0.09868202154415745),
(**1658**, 0.08058379346908832),
(**1739**, 0.06173723727280615),
(**935**, 0.060057982456294914),
(**9**, 0.05563721345394432),
(**1225**, 0.05379769312806656),
(**1915**, 0.04197415367772159),
(**395**, 0.038429808094791934),
(**618**, 0.03561271076451409),
(**1105**, 0.03059670644141617),
(**650**, 0.029747167256267505),
(**980**, 0.029026243346060514),
(**923**, 0.028382878866056123),
(1562, 0.027406190056525365),
(14, 0.026135684059342468),
(**601**, 0.02115408426031194),
(**722**, 0.019739128163438535),
(**1765**, 0.019305914679615458),
(**560**, 0.01849621882482529)]

- Inhomogeneous bi-degree sequence

$$a = 1, \alpha = 2, \beta = 2, .5b = 2$$

bc_sorted[0:20]

[(**1378**, 0.049994599348149704), (**1902**, 0.03920905973565602), (1648, 0.032313359183737685), (1838, 0.02808163728935751), (1348, 0.02636433779475632), (**1934**, 0.02621224779146816), (525, 0.02544710392233814), (1060, 0.025389670230956098), (**243**, 0.025269427037284457), (**1473**, 0.023058889271041744), (**1049**, 0.021645407760879642), (**1596**, 0.021492981486972616), (**1327**, 0.018933014161163227), (1339, 0.01842563629670244), (**450**, 0.018103229322014882), (**432**, 0.017864061737926422), (387, 0.01728755000556956), (**629**, 0.017136761743310126), (1830, 0.016520755739850963), (416, 0.016339285098128358)]

pr_sorted[0:20]

[(**1902**, 0.004495872066075664), (**450**, 0.004312409227570033), (**1378**, 0.003991597795637992), (**1049**, 0.0033455779596443767), (43, 0.0027636758944675223), (**1596**, 0.002758148800156944), (911, 0.002723852212017048), (**1473**, 0.0026964050037848676), (875, 0.0026269183490760213), (**1327**, 0.002619333545794879), (**243**, 0.002585339029660149), (**629**, 0.0024397889748383563), (1939, 0.00241753571612743), (1102, 0.0023535052798460466), (407, 0.00223678266438104), (**1934**, 0.0022252599294097872), (53, 0.002221833727986279), (936, 0.0022201620200149466), (1521, 0.0022141954525901238), (**432**, 0.0021462762332718067)]

In the output above, we could find that the top 20 nodes that have larges BC and PR are highly overlapped.

5. Conclusions

1. The bi-degree sequence obtained by *Algorithm 2.1* converges to the original bi-degree sequence.
2. *Algorithm 2.2.1, the Repeated Algorithm* fails to converge after one hour's running, while *Algorithm 2.2.2 the Erased Algorithm* and *Algorithm 2.2.3 the Revised Repeated Algorithm* converges in less than 2 minutes, under specific set-up (n = 2000).
3. In the set-up above, the nodes of high Betweenbess Centrality tend to overlap the nodes of high Page Rank.