PAFit - An R package for joint estimation of the attachment function and node fitness in temporal complex networks

Thong Pham, Paul Sheridan and Hidetoshi Shimodaira

March 28, 2015

This tutorial demonstrates the use of the `R` package `PAFit` to estimate the attachment function A_k and node fitness f_i in a temporal complex network. The growth process of the network is assumed to follow a modified version of the Bianconi-Barabási model [1]. In this model, a node v_i with degree k and fitness f_i receives a new edge with probability proportional to the product of A_k and f_i. `PAFit` provides a statistical method for joint estimation of the attachment function A_k and node fitness f_i [2]. It does not make any assumptions on either the functional form of A_k or f_i. `PAFit` implements Minorize-Maximization(MM) algorithms [3, 4] to estimate A_k and f_i jointly. Method for computing confidence intervals of the estimated values is also provided. Binning and regularization are implemented, too. `PAFit` are written mainly in `C++` by using the package `Rcpp` [5, 6]. It employs OpenMP for parallel processing. `PAFit` also implements a quasi-Newton acceleration method [7] for speeding-up MM algorithms. These considerations allow high performance even in large datasets.

1 A quick example of how to use PAFit

Here we use `PAFit` to analyse the UC Irvine online student community message sending-receiving network [8]. This publicly available dataset is included in the `PAFit` package under the name `UCIrvine.data`. The format of the data is a matrix where each row contains information of one edge in the form of `(from_node, to_node, time_stamp)`. `from_node` and `to_node` are the ids of the source node and destination node, respectively. `time_stamp` is the arrival time of the node.

We can use such a data as input of the function `GetStatistics`. This function summarizes all important statistics needed in estimation of A_k and f_i. It is assumed that both ids are `integer` starting from 0. `time_stamp` can be either `numeric` or `string`. The only assumption is that a smaller `time_stamp` represents an earlier arrival time. The following script gives us summary statistics of `UCIrvine.data`:

```
library("PAFit")
stats  <- GetStatistics(UCIrvine.data, deg_threshold = 5)
```

The option `deg_threshold = 5` indicates that we only estimate fitnesses of nodes whose number of new edges acquired is not less than 5. The fitnesses of nodes whose number of new edges acquired is 0 are fixed at 0. The fitnesses of all remaining nodes are fixed at 1. This simplification is reasonable, since we are usually only interested in fitnesses of hubs, i.e. nodes that have acquired a considerable number of new edges. This simplification also helps stabilizing the estimation, as well as making estimation in very large datasets possible by reducing the number of parameters.

From now on we can use the object `stats` in subsequent estimation of A_k and f_i without re-running the `GetStatistics` function, unless we want to change the way the dataset is summarized, for example by changing `deg_threshold`.

1.1 Estimation of the attachment function

First we estimate the attachment function A_k in isolation, i.e. fixing $f_i = 1$ for all i, by specifying the option `only_PA = TRUE` in the following script:

```
result <- PAFit(stats, only_PA = TRUE)
```

It is important to note that the aforementioned option `deg_threshold = 5` does not affect the estimation result in the current case of estimating A_k in isolation. We can plot the result as follows.

```
plot(result, stats, plot = "A")
```

The option `plot = "A"` indicates that we want to plot only A_k. The results is shown in Fig. 1a. We can access the estimated attachment function $\hat{A}_k$ via `result$k` and `result$A`. Note that `PAFit` can also estimate the confidence intervals of $\hat{A}_k$, as can be seen from Fig. 1a. One can access the upper ends and the lower ends of these confidence intervals via `result$upper_A` and `result$lower_A`. These confidence intervals are calculated as two standard deviations from the estimated $\hat{A}_k$. The variances of $\hat{A}_k$ (square of standard deviations) are stored in `result$var_A`. If desired, the variances of $\log \hat{A}_k$ can also be explored via `result$var_logA`.

Sometimes one might want to estimate the attachment exponent α assuming the log-linear model $A_k = k^\alpha$, given the estimated $\hat{A}_k$. `PAFit` automatically does this. We can access the estimated $\hat{\alpha}$ via `result$alpha`. In this case, the estimated $\hat{A}_k$ is sub-linear with $\hat{\alpha} = 0.73$.

1.2 Joint estimation of the attachment function and node fitness

Next we estimate jointly the attachment function and node fitness in this dataset by removing the option `only_PA = TRUE` in the previous script:

```
result2 <- PAFit(stats, q = 2) #q >= 2: use quasi-Newton acceleration
```

Note the option `q = 2`. `PAFit` implements a quasi-Newton acceleration scheme [7] that will be activated if $q \geq 2$. Quasi-Newton accelerations are rarely needed in the case of estimating A_k in isolation, since in this case the MM algorithm is already fast enough. In the current setting of joint estimation of A_k and f_i, the implemented quasi-Newton acceleration can often speed up the convergence at little cost. q is the number of previous iterations using in calculating the quasi-Newton direction [7]. Good values of q vary problem-to-problem. Here we choose $q = 2$.

We can plot the estimated attachment function and node fitnesses as follows.

```
plot(result2, stats, plot = "A")
# User needs to open a new plotting device here
plot(result2, stats, plot = "f")
```

Results are shown in Figs. 1b and 1c. As in the previous example, the estimated attachment function $\hat{A}_k$, together with its confidence intervals and variances, are stored in `result2$A`, `result2$upper_A`, `result2$lower_A` and `result2$var_A`. The estimated node fitnesses and their variances can be accessed via `result2$f` and `result2$var_f`. The upper ends and lower ends of the confidence intervals of `result2$f` are `result2$upper_f` and `result2$lower_f`, respectively.

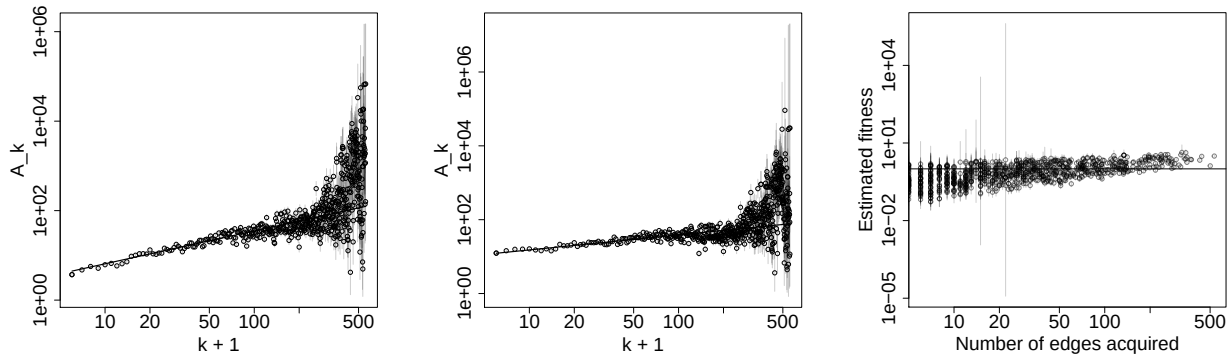
By inspecting `result2$alpha`, it is interesting to find that the estimated attachment exponent $\hat{\alpha}$ in this case ($\hat{\alpha} = 0.41$) is lower than in the previous example.

2 Binning

Binning is the process of dividing the range of the degree k into bins and then group together the statistics of k in each bin. It is a very important pre-processing step in order to obtain stable estimation of the attachment function A_k. `PAFit` employs logarithmic binning. Binning is performed when the statistics are summarized by the function `GetStatistics`. We specify `Binning = TRUE` and then specify the number of bins `G`.

```
#Number of bins is G = 100
stats2  <- GetStatistics(UCIrvine.data, deg_threshold = 5, Binning = TRUE, G = 100)
result3 <- PAFit(stats2,only_PA = TRUE)
plot(result3, stats2, plot = "A")
```

The estimated $\hat{A}_k$ when binning is shown in Fig. 2b. One can see that binning greatly stabilized the estimated $\hat{A}_k$. We recommend to always use binning. The number of bins G should be chosen small enough in order to affect the high degree region.

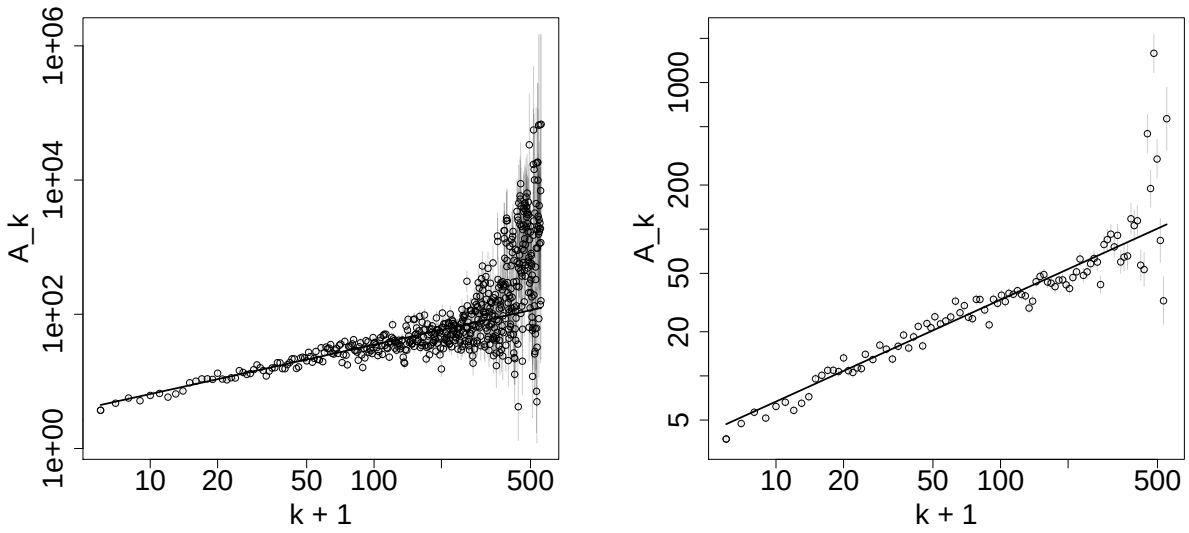


(a) $\hat{A}_k$ when estimating the attachment function in isolation ($\hat{\alpha} = 0.73$).

(b) $\hat{A}_k$ when estimating jointly the attachment function and node fitness ($\hat{\alpha} = 0.41$).

(c) $\hat{f}_i$ when estimating jointly the attachment function and node fitness.

Figure 1: Estimation results in the UCIrvine dataset.



(a) $\hat{A}_k$ when estimating the attachment function in isolation without binning ($\hat{\alpha} = 0.73$).

(b) $\hat{A}_k$ when estimating the attachment function in isolation with binning ($\hat{\alpha} = 0.69$).

Figure 2: Effect of binning

3 Regularization

Estimating jointly the attachment function A_k and node fitness f_i can be difficult, due to data sparsity and the non-concavity of the log-likelihood function. `PAFit` implements regularization for A_k and f_i in order to alleviate those problems. For the attachment function, `PAFit` use the following regularization term:

$$-\lambda \frac{1}{\sum_k w_k} \sum_{k=1}^{K-1} w_k (\log A_{k+1} + \log A_{k-1} - 2 \log A_k)^2 \ . \tag{1}$$

This penalty term penalizes the second order differentiation of $\log A_k$, and by doing so encourages linearity of $\log A_k$. The ratio of the strength of the regularization term (measured by λ) and the number of observed

data can be used as a heuristic, reasonable criterion for choosing λ. One can then specify λ indirectly through the parameter `ratio`. Regarding the weights w_k, the option `weight_PA_mode = 1` corresponds to the case $w_k = 1$ for all k, while specifying `weight_PA_mode = 0` corresponds to the case $w_k = \sum_t m_t(k)$, where $\sum_t m_k(t)$ is the total number of edges connected to a degree k node. We recommend the latter case.

Regarding node fitness f_i, `PAFit` put a regularization term that equals to the effect of placing a Gamma prior distribution on f_i. We can specify the shape and rate of this Gamma distribution through the parameters `shape` and `rate`.

To summarize, the following script performs estimation with `ratio = 0.1`, the weights $w_k = \sum_t m_t(k)$, `shape = 0.5` and `rate = 0.5`.

```
result4 <- PAFit(stats2, q = 2, ratio = 0.1, weight_PA_mode = 0,
                 shape = 0.5, rate = 0.5)
```

4 Simulated data

`PAFit` includes the function `GenerateNet` to generate networks from many important network models, including the Barabási-Albert model [9] and the Bianconni-Barabási model [1]. For example, the following script generates a network where $A_k = k$, $f_i \sim Gamma(1, 1)$, total number of nodes $N = 1000$, and number of new edges introduced at each time step is $m = 5$:

```
#mode = 1: A_k = k^alpha with alpha = 1
data1 <- GenerateNet(N = 1000, m = 5, alpha = 1, shape = 1, rate = 1,mode = 1)
```

The object `data1` is a list with components `data1$graph` and `data1$fitness`. `data1$graph` is a 3-column matrix where information about the edges is stored in each row. `data1$fitness` stores the true fitness value of each node. One then can use `data1$graph` as the input of `GetStatistics`.

One can also generate networks from the attachment function $A_k = \min(k, \text{sat\_at})^\alpha$ with $\alpha = 1$ and $\text{sat\_at} = 100$ as follows.

```
#mode = 2: A_k = min(k,sat_at)^alpha with alpha = 1, sat_at = 100
data2 <- GenerateNet(N = 1000, m = 5, mode = 2, alpha = 1, sat_at = 100, shape = 1, rate = 1)
```

Finally, the following script generates a network where the attachment function is $A_k = \alpha \log^\beta(k) + 1$ with $\alpha = 3$ and $\beta = 2$.

```
#mode = 3: A_k = A_k = alpha*log^beta(k) + 1 with alpha = 3, beta = 2
data3 <- GenerateNet(N = 1000, m = 5, mode = 3, alpha = 3,beta = 2, shape = 1, rate = 1)
```

In all examples of this section, the number of new edges at each step m has been fixed at `m = 5`. It might be more realistic to let m be a Poisson random variable, whose value of realization at each time-step varies. This can be archived by specifying `prob_m = TRUE`. In this case, if the option `increase` is `FALSE` then the mean of this Poisson distribution is fixed at `m`, otherwise the mean itself will grow with the current size of the network. In the latter case, if `log = TRUE`, the mean will grow logarithmically with the current size, otherwise it will grow linearly.

5 Conclusions

`PAFit` provides statistically sound algorithms for joint estimation of the attachment function A_k and node fitness f_i in a growing complex network. The package incorporates variance estimation, confidence interval calculation, binning and regularization. `PAFit` also provides plotting functions for easy visualization of the estimated result, and a function to generate simulated data based on various important network growth models. We also implement a quasi-Newton acceleration method [7]. Written mainly in `C++`, `PAFit` can perform well even in large datasets.

References

[1] Bianconni G, Barabási A. Competition and multiscaling in evolving networks. Europhys Lett. 2001;54:436.

[2] Pham T, Sheridan P, Shimodaira H. Nonparametric estimation of the preferential attachment function in complex networks: evidence of deviations from log linearity (submitted); 2015. .

[3] Hunter D, Lange K. Quantile regression via an MM algorithm. J Comput Graphical Stat. 2000;p. 60–77.

[4] Lange K. Numerical Analysis for Statisticians. New York: Springer-Verlag; 2014.

[5] Eddelbuettel D, François R. Rcpp: Seamless R and C++ Integration. Journal of Statistical Software. 2011;40(8):1–18. Available from: `http://www.jstatsoft.org/v40/i08/`.

[6] Eddelbuettel D. Seamless R and C++ Integration with Rcpp. New York: Springer; 2013. ISBN 978-1-4614-6867-7.

[7] Zhou H, Alexander D, Lange K. A quasi-Newton acceleration for high-dimensional optimization algorithms. Statistics and Computing. 2011;21:261–273.

[8] Opsahl T, Panzarasa P. Clustering in Weighted Networks. Social Networks. 2009 May;31:155–163.

[9] Albert R, Barabási A. Emergence of scaling in random networks. Science. 1999 October;286:509–512.