Programming Language Evolution

Eoin Doherty

CSCI 5352: Network Analysis and Modeling
Fall 2019

Abstract

Programming languages incorporate features and functionality from older, influential languages. The set of influences between programming languages forms a directed treelike network, similar to a citation network. In 2015, Sergi Valverde and Richard Sole analyzed the programming language influence network using data pulled from Wikipedia in 2012. They found that programming language evolution is a punctuated equilibrium, programming languages can be split up into two main clades based on language paradigm, and the influence network can be modelled reasonably well by a modified GNC model[5]. This project replicated Valverde and Sole's paper with new data from 2019. The results were very similar with a few differences. The network was split into three main clades instead of two, and the GNC model had a much less accurate link distribution.

1 Introduction

Programming languages, much like spoken languages borrow structure and functionality from each other as time goes on. An older programming language like Algol may not be used widely anymore, but features it introduced are still present in modern programming languages. Investigating language influence can help programming language creators rediscover influential languages whose forgotten features may be useful today. From a broader perspective, knowing which modern language is the most influential is valuable to computer science educators who want to introduce their students to coding while also teaching them marketable skills. For those who already know how to code, learning a language that is influential but outside of the family of languages they already know can broaden their programming knowledge and help them approach problems differently. Lastly, from a historical standpoint, trends in the emergence of new programming languages can also reflect trends in computer science and the tech industry as a whole.

Programming language influences can be modeled as a tree-like citation network where each language is a node, and each edge from a language points to another language that influenced it. Though plenty of research has been conducted on academic citation networks, and some research has been conducted on the history of programming languages[4], very little work has been done on the programming language influence network. In addition to this being a very niche topic, there are no first-hand ”works cited” lists written by the creators of programming languages which makes citation analysis difficult. Some documentation of programming language influences exists on wikipedia where each language's page contains a list of languages that influenced it and a list of languages that it influenced. One paper titled ”Punctuated equilibrium in the large-scale evolution of programming languages,”[5] by Valverde and Sole analyzes this network. In this paper, Valverde and Sole treated the programming language influence network like an evolutionary network. They analyzed how programming languages have changed over time, extracted programming language clades, created a randomized model to mimic the language network, and analyzed the topology of the network[1]. The results of this paper were interesting, but it was published in 2015 and used data scraped from wikipedia in 2012. In the last seven years, new programming languages have been created, and new language versions have been created. This project replicated Valverde and Sole's research using new data scraped from Wikipedia in October 2019 to see if the trends observed in that paper still exist.

[1]Supplementary Material 1

2 Methods

2.1 Data Extraction and cleaning

Data was gathered from Wikipedia using a python script and the Wikipedia API. The script traversed links in a breadth first order starting with wikipedia's list of programming languages. For every language page, the language's name, creation date, paradigm(s), typing discipline, list of influenced languages and languages that influenced it (adjacency lists) were extracted from the page's info-box. Each page link in the adjacency list that had not been traversed or queued for traversal was added to the breadth first queue. Different links can point to the same page, and different languages can have the same name (ex: apple's swift, and the swift parallel scripting language), so languages were stored by Wikipedia page id. The adjacency lists of all languages were combined into one edge list for the entire network. 886 programming languages were extracted, but many of these were not pages for languages. Some of these nodes were programming tools, libraries, language versions or implementations, joke programming languages, or educational video games. These nodes were flagged by hand and their entries in the edge list were removed. The final network consisted of 795 language nodes and 1221 edges. Much of the analysis in the original paper depended on language creation date and temporal relationships, so nodes without a valid creation date were removed from the network. The final network contained 355 nodes and 946 edges. This data along with the code used in the project can be found on the project github repository.[1]

2.2 Network Analysis

The analysis conducted in this paper replicates the techniques used by Valverde and Sole in the original paper and its supplementary materials. This paper reproduced the analysis conducted in the original paper as closely as possible, though some parameters had to be tweaked due to the new data. Some more specific details that were not discussed in the original paper also had to be inferred which could be responsible for slightly different results.

To start, the in-degree of each node and the network's link distribution were analyzed over time. The in-degree over time shows how influential some languages are compared to others, and the link distribution plot shows how the network changes over time.

Programming language clades were also extracted from the network by analyzing the edges of the network and keeping each node's most significant outward pointing edge. Valverde and Sole defined the impact of a node j on its descendent i as

$$Y_{ij} = a_{ij} \sum_{k \neq i} \lambda s^a_{ik} + (1-\lambda) s^d_{ik}$$

where $\lambda \in \{0,1\}$ and k is a peer node of i, meaning $a_{kj} = 1$. Out-similarity (s^a_{ij}) and in-similarity (s^d_{ij}) are measures of neighbor overlap between two peers. They are defined as

$$s^a_{ij} = \frac{1}{k^{out}_j} \sum_l \frac{a_{il}a_{jl}}{k^{in}_l}; s^d_{ij} = \frac{1}{k^{in}_j} \sum_l \frac{a_{li}a_{lj}}{k^{out}_l}$$

s^a_{ij} gives the probability that a random walk starting at i reaches j through a common ancestor of i and j while s^d_{ij} gives the probability that a random walk starting at i reaches j through a common descendent of i and j. To speed up calculations, s^a_{ij} was calculated by iterating over the union of i and j's ancestor sets (adjacency lists), and s^d_{ij} was found using the union of i and j's descendent sets. If both similarities were undefined, Y_{ij} was set to 10^{-9}. λ was set to 0.5, the same value used in the original paper. To form clades, the edges with the maximum Y_{ij} for each j were copied over to the clade network. This keeps each node from the original network in the clade network, but each node only has one outward pointing edge.

Next, a randomized model was constructed to replicate the structure of the programming language influence network. Valverde and Sole used a modified network growth by copying (GNC)[3] model. At every time step, a new node was introduced. That new node linked to every old node with probability p, then it linked to each ancestor of its new neighbors with probability q. Using just p and q can lead to an explosion of edges, so a Boltzmann saturation term was used instead of the raw probabilities. p and q became

$$p(j) = \frac{p}{1 + exp(-\beta(k_j - k_M))} \text{ and } q(j) = \frac{q}{1 + exp(-\beta(k_j - k_M))}$$

Since there were two phases of the link distribution, the GNC simulation was done in two phases with different ps, qs, βs, and k_Ms. p_1 and q_1 were inferred from the data by simulating adding nodes over time from the earliest time in the dataset, 1948, to the end of the first phase at 1982. At each iteration, the the out-degree of the new node was divided by the number of nodes already in the network to estimate the p of that iteration; these values were accumulated and averaged to estimate p_1. To find the q of each iteration, the length of the union of the set of the new node's neighbors and the set of the new node's neighbors'

neighbors was divided by the number of neighbors's neighbors. These q estimates were averaged to find q_1. The same process was repeated for nodes from 1982 onward, but starting with the network of pre-1982 nodes as a seed. The performance of this model was evaluated by comparing degree distribution and link distribution to the data.

Tree imbalance was also examined to analyze the structure of the language network clades and the model. Tree imbalance was defined as the average distance from the root to all nodes in the network.

$$\langle d \rangle = \frac{1}{N} \sum_i d(r, i)$$

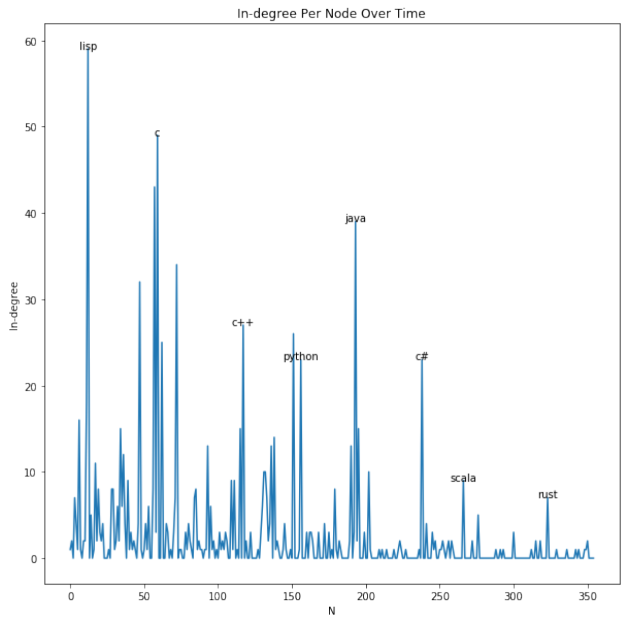
This was compared to the GNC model and a simple ERM model where two new nodes are added to the leaf of the tree at each timestep.

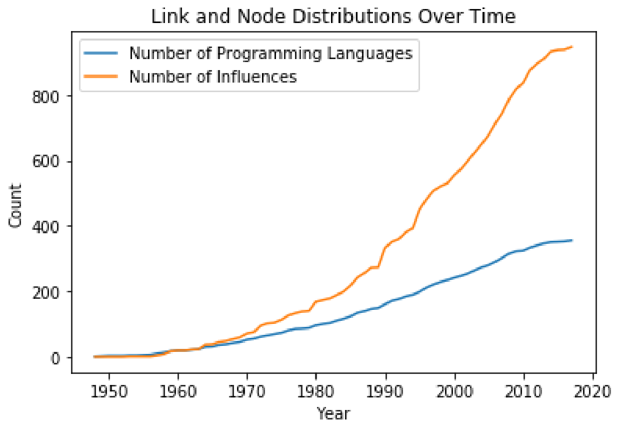
The rest of the analysis is from supplementary material 1 of the original paper. This supplement analyzed the topology of the network, showed programming language family trees, and analyzed how programming language paradigm influences clade structure. For the sake of brevity, I have not included pictures of programming language family trees.

The degree distribution of the network (sum of the in and out degrees of each node) was fitted to a power law using a least squares fit[2] to check if it is scale free. The average clustering coefficient was also found and compared to the expected clustering coefficient of a randomly generated graph to see if the network exhibits small-world behavior. A pearson correlation coefficient was also found to determine if the language network exhibits disassortative or assortative mixing. Lastly, the most frequent language paradigm was found for each of the largest clades to see if clades were biased toward a paradigm. This was not done in the same way in the original paper because programming paradigm was not present in the original dataset.

3 Results

For the most part, the results were consistent with the original paper. Analyzing the degree, node, and link distributions over time yielded the following plots:





Some nodes like Lisp or C have much higher in-degree than the other nodes in the network which shows that the network follows a punctuated equilibrium. A punctuated equilibrium is a model of evolution where influential changes happen over short periods of time. The original paper found the same behavior. Older nodes also tend to have higher in-degree implying there is some preferential attachment. The same two phases before and after 1982 are present in the link distribution which is consistent with the original paper.

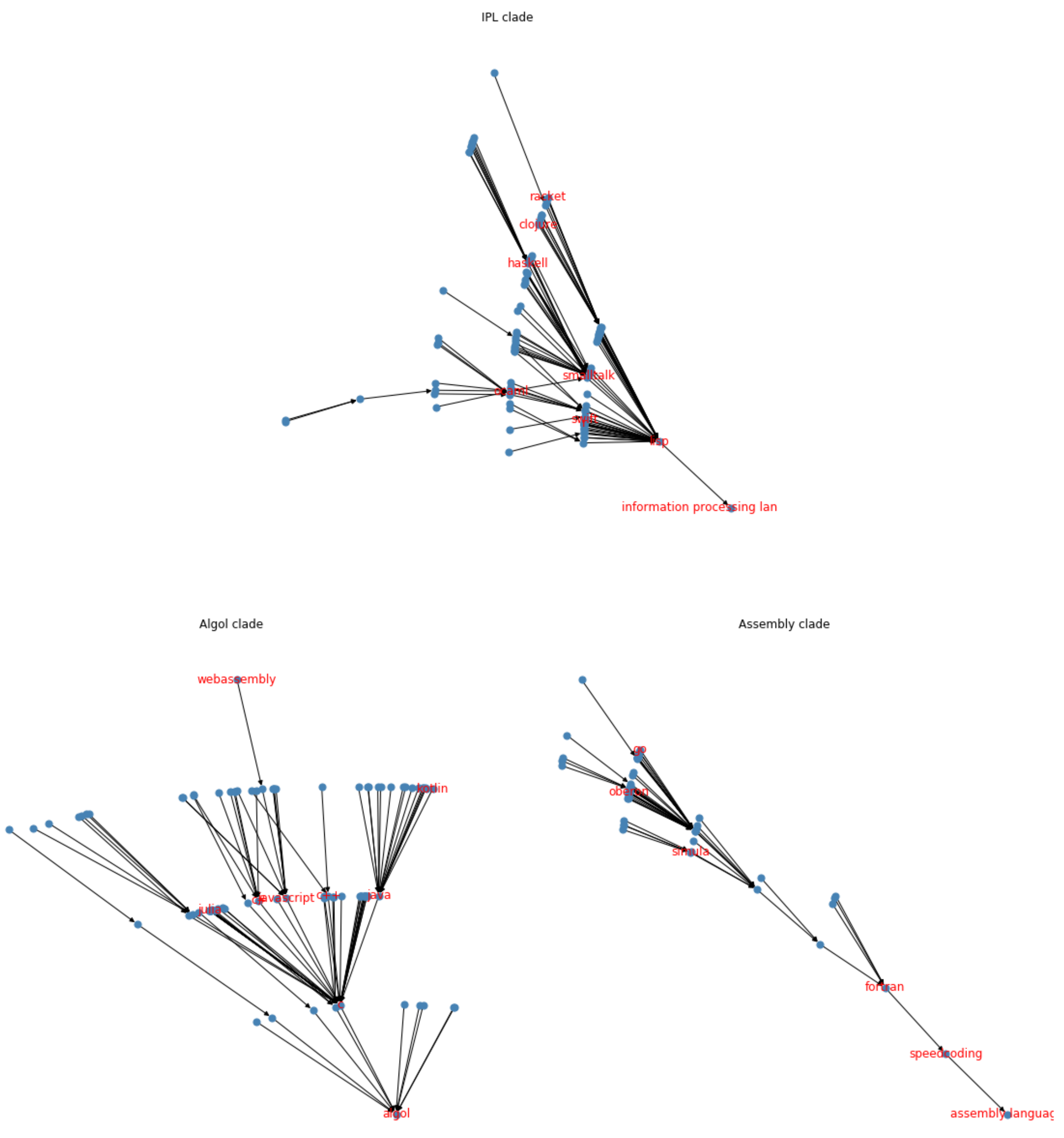


Figure 1: *The three largest clades extracted from the programming language network.*
The programming language clades extracted from the network differ from those extracted in the paper. The two largest clades found by Valverde and Sole were rooted at Lisp and

Fortran. With this dataset and the clade extraction methods detailed above, those two clades were split into three large clades: one rooted at IPL (though lisp is the de facto root since IPL only has one in-edge), one rooted at algol, and another rooted at assembly. It appears that the Fortran clade has been split up into two distinct clades (Algol and assembly).
Attempts to replicate the network behavior using a GNC model also differed from the original paper. Estimates of p values and q values yielded $p_1 \approx 0.05, q_1 \approx 0.11$ and $p_2 \approx 0.01, q_2 \approx 0.09$. Other parameters used were $\beta_1 = \beta_2 = 0.1$, $k_{M1} = 5$, and $k_{M2} = 10$. The GNC model was run 100 times, and for each node, the average, minimum, and maximum out-degree was recorded. When compared to the actual link distribution of the the language influence network, the fit was very loose meaning GNC using these parameters is not a very good fit for the link distribution of the actual network. This discrepancy could be due to how p and q were found for this project. In-degree over time which was used as a measurement of evolution over time follows a slight punctuated equilibrium similar to the original data, but it does exhibit much more preferential attachment.

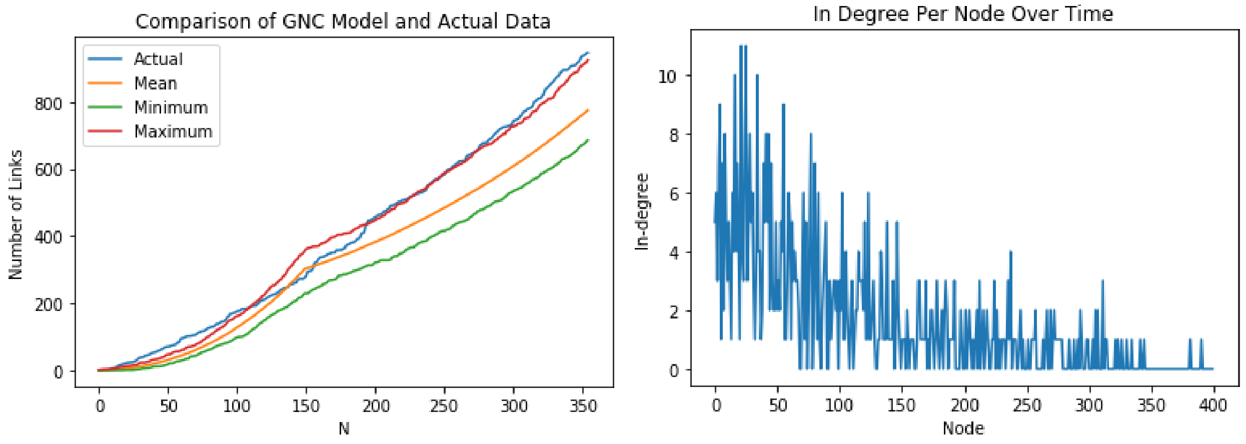


Figure 2: ***Left:*** *GNC link distribution compared with actual link distribution.* ***Right:*** *GNC in-degree over time.*

The distributions of the mean in-degrees and out-degrees of the GNC model iterations were also analyzed and compared to the data. The distributions from the model follow the actual distributions very closely, so the model is useful for replicating in and out-degree behavior.

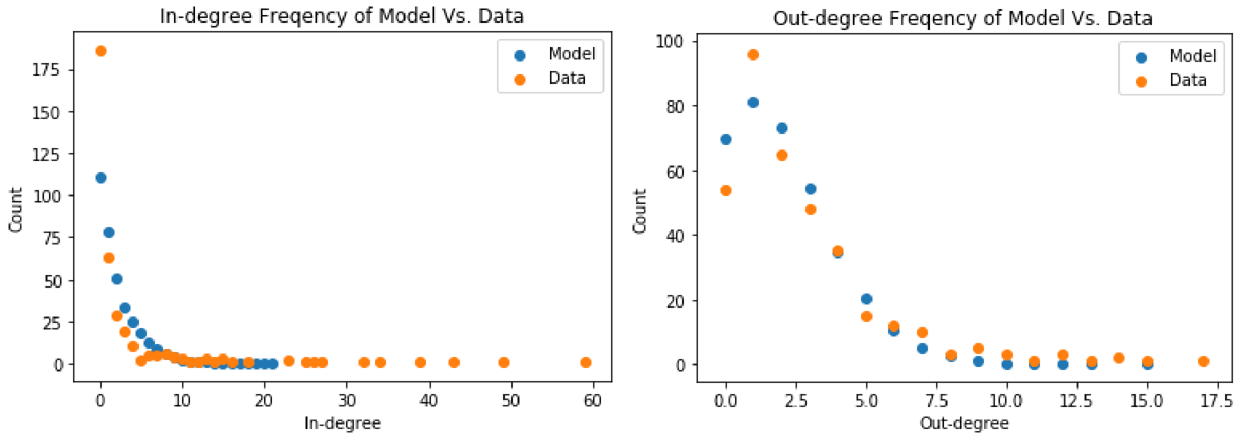


Figure 3: *Mean GNC in-degree and out-degree distributions versus the in and out-degree distributions of the data.*

The average depth of the language network using the oldest language (plankalkul) as the root was 3.36 which is much lower than the perfectly balanced value of $\log_2 314 \approx 8.29$. The average depth of the GNC model was about 0.93 which was significantly more unbalanced because attachment is so biased. The average depth of the ERM model was 8.1, very well balanced. Valverde and Sole did find that GNC was more imbalanced than the actual data, but they also found that the actual data had a higher average depth than the ERM model. The degree distribution of the network was fitted to a power law ($k^{-\gamma}$) with $\gamma \approx 1.26$. Since a power law fit the distribution, the network is scale free.
The average clustering coefficient of the network was 0.16 which is much larger than the expected $\frac{k_{avg}/N}{=}0.015$. The average shortest path length was about 3.01 which is close to the average shortest path of a randomly generated tree (≈ 3.15). This meets both of the requirements for a small-world graph used in the original supplementary materials and is consistent with Valverde and Sole's results.
The most frequent programming language paradigm was found for the three largest clades of the network. The IPL clade contained mostly functional languages while the Algol and assembly clades contained mostly imperative languages. This is consistent with the findings of the original paper where the Lisp clade was functional and the Fortran clade was imperative. In this case, the fortran clade was split into the Algol and assembly clades.

4 Discussion

The programming language influence network extracted from wikipedia has some interesting properties. It exhibits slight preferential attachment and is unbalanced, scale free, and small-

world. Programming language evolution follows a punctuated equilibrium, and programming language clades are generally grouped by programming paradigm. While there were some differences between this project's results and Valverde and Sole's results, for the most part, the same conclusions can still be drawn with the new data. My GNC model did not perform very well compared to Valverde and Sole's GNC however, and that is probably due to the different way I estimated p and q.
Given more time, I would have liked to analyze the rest of the data that I had collected. I had to throw out 440 nodes because they did not contain valid date information. Those nodes could have been useful for analyzing the structure of the network, but the original paper mostly focused on the temporal aspect of the network. The extra nodes also could have joined the two imperative clades together which would be closer to the results of the original paper. Ranking languages based on degree and/or total impact (Y) is some interesting future work as well.
I encountered a few problems that I had not anticipated at the start of the project. Wikipedia's list of programming languages was not complete, so I had to use a breadth first search to gather data instead of just iterating over the pages linked in the list. The link distribution results for the GNC model in this project were also very different from the original paper's results which found a very good fit. I tried plenty of different parameters and methods of finding p and q, but I could not replicate it. Other GNC models might be more successful.

5 References & links

References

[1] Programming language evolution. `https://github.com/EoinDoherty/PLEvolution`.

[2] Adam Ginsburg. plfit. `https://github.com/keflavich/plfit`, 2019.

[3] P. L. Krapivsky and S. Redner. Network growth by copying. *Phys. Rev. E*, 71:036118, Mar 2005.

[4] Jean E. Sammet. Programming languages: History and future. *Commun. ACM*, 15(7):601–610, July 1972.

[5] Sergi Valverde and Ricard V. Sole. Punctuated equilibrium in the large-scale evolution of programming languages. *Journal of The Royal Society Interface*, 12(107):20150249, 2015.