Network Analysis of Programming Languages

Kazuma Endo[a,1], **Bella Lam**[a,1,2], **and Britney Zhao**[a,1]

[a]Department of Mathematics, University of California, Los Angeles, CA 90024

This manuscript was compiled on December 17, 2022

We examined a list of programming languages and visualized a network where they influence one another. We then ran different tests and models to determine the most influential languages and how the network structure has evolved over time.

Programming Languages | Influences | Complex Networks | Directed Acyclic Graph

Programming languages are the notion system of writing computer programs. As software development engineers, we often find similarities between different programming languages. This makes us curious on how the languages are related to each other.

Introduction

The history of programming languages starts with a piece of code written on a piece of paper in 1843 by Ada Lovelace. Other rudimentary programming languages followed, such as a language that allowed for the creation of procedures (If, then, for statements) and a high-level language (shortcode) that needed programmers to manually turn all codes into 0 and 1s, for the computer to understand. The creation of compilers by Grace Hopper led to faster programming, as it resolved the shortcoming of needing to type binary to run code. The trends of how programming languages were used by the industry and the people drove other major shifts in the capability and usability of programming languages. For example, in the early stages, the need for scientific computing was eminent to give programmers access to computer innards where FORTRAN was designed by IBM. Then business computing started to take off. It was designed for the average businessman, and it was revolutionary in the sense that it had a very English-like grammar, making it quite easy to learn and adopt for many people. Other forms of coding paradigm were driven by designing a programming language for a specific use, such as artificial intelligence research and changes that came with the technological advancement of computer technology and the spread of personal computers.

Programming languages have been under development for years and will continue to be developed for many years to come. They got their start with a list of steps to wire a computer to perform a task. These steps eventually found their way into software and began to acquire newer and better features. The first major languages were characterized by the simple fact that they were intended for one purpose and one purpose only, while the languages of today are differentiated by the way they are programmed in, as they can be used for almost any purpose (1).

Research Question. Since the technologies and features which people want from programming have changed over time, and the languages adapt to those changes, we hypothesize that most languages run wide rather than long and that the network structure of how programming languages are influenced over time reflect periodic shifts in the industry. The research question that we want to answer is which programming languages are the most influential and how the structure of the network evolves over time.

Data Retrieval

As there is no data set on the internet that we can use for this research, we retrieved and compiled our own data from Wikipedia pages to get the corresponding data we needed for this project.

Web Scrapping. We first got the full list of programming languages from the Wikipedia page (2). To do that, we run a web scrapping program through Python and get each hyperlink from the table and their corresponding title which is the name of the programming language in our case. With that list of hyperlinks, we can run requests using Python and get each of their "Influenced By", "Influenced", "Year" field.

Cleaning Up. Wikipedia is a open-source Internet-based encyclopedia, which means everyone can modify the data on it. It caused inconsistency in the data we collected. The stage of data clean up consists of removal of duplicate nodes, removal of noisy nodes, addition of nodes. Since there is no easy way to do it, we ultimately had to manually look through the whole data set. Lastly, we have to preserve the ordering of the "Year" field. If we go through the list of programming languages, many of them have an empty "Year" field as it is not stated in their Wikipedia page. In order for our configuration model to work, we have to add valid dates to keep the order. For languages with a empty "Year" field and not influenced by or influence other languages, as known as nodes with degree 0, we set their year to 1. For languages with a empty "Year" field but has a valid "Influenced by" and "Influence" field, we set their year to 2003. For languages with a empty "Year" field but valid "Influenced by" field, we set their year to 2024. With this set, we go through each value in "Influenced By" and "Influenced", validate that "Influenced By" languages are

Significance Statement

Programming languages are notion system for writing computer programs. We want to find out which programming languages are the most influential and how the network structure of influence evolved over time. Our hypothesis is the network structure of how programming languages are influenced reflects periodic shifts in the industry.

Endo contributed on configuration models, Lam contributed on web scrapping, Zhao contributed on visualization. All three authors contributed on analysing and writing up paper.

The authors declare no competing interests.

[1]All three authors contributed equally to this work.

[2]To whom correspondence should be addressed. E-mail: lamchunying@ucla.edu

older and "Influenced" languages are newer in time, if not update. This then sums up the whole cleaning up process and all calculations below are done with the cleaned up data set. All the procedure above and the data set can be found in our GitHub repository (3).

Visualization and Architecture of the Network

Creating the Visualization. After web scraping from Wikipedia, we used this data to visualize the programming language influence network. For data manipulation, we used NumPy and pandas, and for visualization, we used the NetworkX and Matplotlib packages in Python.

Using the "Influenced" and "Influenced By" columns of the data set, we constructed the graph by creating a directed edge from a programming language to one that this programming language had influenced. For example, if language A influenced language B, then we added a directed edge from A to B. On the other hand, if language C had been influenced by language D, then we added a directed edge from D to C. Using this construction method, we used the data set to create the graph accordingly.

We then sized the nodes in proportion to their out-degree. Therefore, nodes that influenced more programming languages were larger on the network to visualize the importance of certain languages onto others. We also colored the nodes based on the decade they were created, spanning from 1940 up until 2024. In this specific visualization, orange nodes represent languages that emerged between 1940-1949, green nodes represent languages that emerged between 1950-1959, and so on. Figures 1 and 2 are visualizations of the network, one with every node visible and one being a zoomed-in section of the center of the network, which shows the largest nodes with the highest out-degree. A higher resolution version of our network can be found on our Github (3), which is named 'graphpng.png'.

Analysis of the Visualization. From the above images of the network, we can see that some of the larger nodes are well-known languages, such as C, Java, and Python. There are also other languages such as Smalltalk, ML, and Pascal that have high out-degree relative to the other languages.

We can also see that the more influential languages tend to be either red, purple, or brown. These colors correspond to the decades 1960-1969, 1970-1979, and 1980-1989. Based on this information, we can see that these thirty years created many programming languages that remain relevant to this day based on their influence on other languages.

Architecture of Network. After constructing this network, we have a directed acyclic graph. The acyclic nature of our programming language network arises from the fact that they are time ordered. A programming language can only cite others that already exist and this eliminates closed cycles since all paths in the network must lead backward in time and there are no forward paths available to close the cycle (4). The nodes represent the programming languages and the edges represent whether the two languages are influenced by one another. If there is an edge from node i to node j, it means that node i "Influenced" node j and that node j was "Influenced By" node i. Thus, this network of programming language influence is a directed acyclic graph.

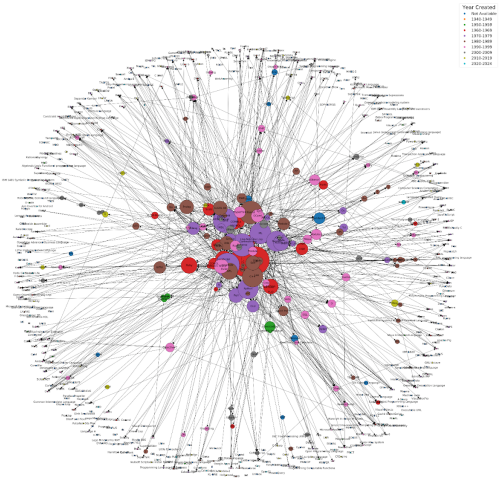

Fig. 1. Visualization of our network containing all nodes. Different colors depict which decade the programming language was created. Size of nodes is proportional to the out-degree of the node.

Data Calculations and Analysis

In order to determine the most influential language and to support our hypothesis, we conducted the following tests and calculations.

Centrality Measures. To analyze which programming languages were the most influential, we calculated various measures of centrality. We specifically decided to look into degree centrality, betweenness centrality and closeness centrality. Degree centrality will help us analyze which languages influenced the most languages, and betweenness centrality will look into which nodes heavily exist on the paths of other nodes, and closeness centrality shows which nodes can travel to other nodes quickly. All of these measures indicate which nodes tend to either have many connections or are near many connections, which can help us analyze which programming languages tend to be the most influential within the field.

Ranking	Programming Language	Score
1	C	0.08346
2	Java	0.07899
3	Lisp	0.07303
4	Smalltalk	0.06259
5	Python	0.06259

Table 1. Top-5 programming languages with highest degree centrality values .

Degree Centrality. As there are over 600 nodes in this network, we decided to look at degree centrality as a proportion to the entire network. Therefore, we normalized this calculation by dividing a node's degree by the total number of nodes in the

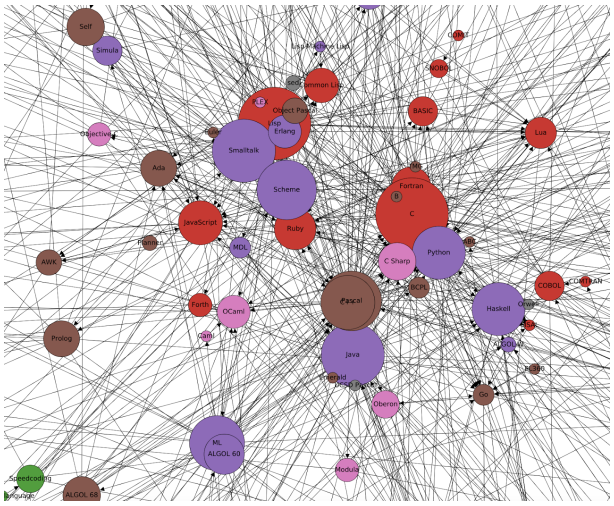

Fig. 2. Image of network with the largest nodes, highlighting the nodes with the highest out-degree.

network. In Table 1, the top five languages with the highest degree centrality are listed. From this, we can see that C has the highest degree centrality, and has influenced many large languages such as C#, C++, Java, and Python (5). Java is the second in degree centrality, and is one of the most popular programming languages out there, showing the influence of these notable languages. Thus, the degree of these nodes is consistent with their importance in the field of computer programming

Ranking	Programming Language	Score
1	C	0.00363
2	Java	0.00269
3	Python	0.00255
4	Haskell	0.00213
5	ALGOL 60	0.00202

Table 2. Top-5 programming languages with highest betweenness centrality values.

Betweenness Centrality. We also calculated the betweenness centrality, which measures the extent to which a node lies on paths between other nodes. This means a higher betweenness centrality means that more disruption within the network would occur if removed. The five languages with the highest betweenness centrality are listed in Table 2. There are similar languages with either degree and betweenness centrality, showing how these programming languages are highly influential. Languages like C and Java depict how their large influence on other languages means they would greatly impact the overall structure of the network.

Closeness Centrality. The last centrality measure we calculated was closeness centrality. This measurement calculates the mean distance from one node to other nodes. Therefore, a higher closeness centrality means that a node can reach other nodes quicker. In Table 3, we have the top 5 languages with the highest closeness centrality value. The languages with the highest closeness centrality all differ from the top five of degree and betweenness centrality. However, these nodes are

Ranking	Programming Language	Score
1	Rust	0.05320
2	Zig	0.05062
3	Crystal	0.05033
4	Go	0.05014
5	Swift	0.04962

Table 3. Top-5 programming languages with highest closeness centrality values.

connected to nodes with high degree centrality. For example, the language with the highest closeness centrality is Rust, and Rust is influenced by high-degree languages such as C#, C++, and Ruby (6). Similarly, Swift, with a closeness centrality of 0.0496 is influenced by popular languages such as Python and C# (7). Therefore, while these languages themselves may not be as notable as other programming languages, they are connected to highly important and influential languages.

Hubs and Authorities. Authorities are nodes that contain useful information on a topic of interest and hubs are nodes that tell us where the best authorities are to be found. In order to find the hubs and authorities for our particular network, we ran the HITS algorithm which is built in the NetworkX library(8). The HITS algorithm give each node i in a directed network two different centrality score, the authority centrality x_i and hub centrality y_i, which quantify nodes' prominence in the two roles.

$$x_i = \alpha \sum_j A_{ij} y_j \quad [1]$$

$$y_i = \beta \sum_j A_{ij} x_j \quad [2]$$

The hits() function in NetworkX library computes two numbers for a nodes. Authorities estimates the node value based on the incoming links and hubs estimated the node value based on outgoing links. It then returns HITS hubs and authorities values for nodes. The Top-5 programming languages with the highest hub and authority scores are listed in Table 4 and Table 5 respectively.

Ranking	Programming Language	Score
1	C	0.07338
2	Lisp	0.06021
3	Smalltalk	0.04980
4	C++	0.04982
5	Java	0.04851

Table 4. Top-5 programming languages with highest hub scores.

When applying the Hubs and Authorities theory in our paper, Authorities are programming languages that contains valuable information to influence other programming languages. Hubs are the programming languages that are relevant to finding the authorities. C is the programming language with the highest hub score. As C has a lot of outgoing links, C contains more information when comparing to other programming languages. In another words, C influenced a lot of different languages. Python is the programming language with the highest

Ranking	Programming Language	Score
1	Python	0.02103
2	Java	0.01775
3	Rust	0.01668
4	Ruby	0.01632
5	Scala	0.01631

Table 5. Top-5 programming languages with highest authority scores.

authority score. In another words, Python has the most incoming links. As Python introduced many new features to the market, such as list comprehensions, cycle-detecting garbage collection, reference counting, and Unicode support (9), it becomes the authority when influencing other programming languages.

Community Detection. To better understand the behavior of the systems a network represent, we wanted to investigate in the groups found in our network. Community detection is key when dealing with groups in a network. The Clauset-Newman-Moore Greedy Modularity Maximization is the algorithm we used to finding communities partition with the largest modularity. The Greedy modularity maximization begins with each node in its own community and repeatedly joins the pair of communities that lead to the largest modularity until no further increase in modularity is possible (10). NetworkX has a function called greedy_modularity_communities() that can calculate this. The function returned a total of 336 communities and are visualized in Table 6. The average size of the communities is 2.0. When scanning through our data set, there is a significant amount of programming languages that has no "Influenced By" or "Influenced" nodes which therefore created a lot small communities, 322 community with only 1 node, in exact.

Community Size	Count
85	1
60	1
55	1
52	1
27	1
18	1
16	2
6	2
3	1
2	3
1	322

Table 6. Community Size and their count

One might find it weird that there a lot of communities with size 1. If we take reference to our data set, there are actually a lot of nodes with no "Influenced By" and "Influenced" nodes which means it has no incoming or outgoing links. They are classified as single nodes which then creates communities with only itself.

In Figure 3, each color represent a community. We can also see from the graph that our communities are small and rarely overlapping.

Our biggest community has a total of 85 languages and contains the following languages: Script.NET, Python, Objective-C, Java, Perl, C etc. Even though there are both scripting languages, Perl, Python etc. and non-scripting languages, C, Java etc. in this community, here are some reasoning behind them being classified in the same community. Programming languages in this community are all general-purpose programming languages. They also share similar syntax and borrowed features from one another. Therefore it is reasonable to believe that if one have already know a programming language, it is easier for one to pick up other programming languages in the same community. If one wants to learn a completely new programming language, it would be best to avoid picking languages from the same community. This helps determines which programming languages to learn next.

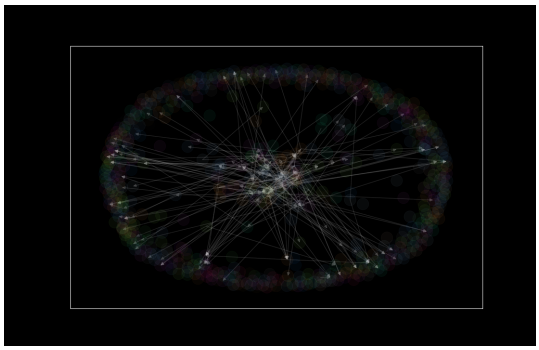

Fig. 3. Visualization of Communities in our network, where each color represents a different community

Methods and Models

Creation of Random Graphs using Configuration Model. The random graph models of directed acyclic networks that we used resembles a configuration model, where we take as input an ordered degree sequence consisting of the in degree k_i^{in} and out degree k_i^{out} for each node in the network. The edges in this network can only run from lower to higher indices (older to newer in terms of year the programming language was created) and this constraint enforces the acyclic nature of the network (4). To create random graphs, we randomly join each node's outgoing stubs to ingoing stubs chosen uniformly random from the set of all in-stubs that are available to be paired with (4). When all stubs have been matched, the network is complete and the algorithm ends. The algorithm was implemented efficiently since we maintained a list of currently unclaimed in-stubs for each node, where we choose one of the nodes randomly from this list to connect to. When it is chosen, each stub that was randomly chosen is erased from the list and the algorithm moves on to the next node.

Calculation of Flux. The creation of random graphs involves looking at the stubs going out from each node and pairing the in-going stubs with the out-going stubs until all stubs are matched respectively. The flux is calculated as the number of in-going stubs below node i that are available to attach to outgoing stubs at i and above, calculated as the following, from (4):

$$\mu_i = \sum_{j=1}^{i-1} k_j^{in} - \sum_{j=1}^{i-1} k_j^{out} \quad [3]$$

Low values of flux indicate a "bottle neck" in a network and high values indicate regions in which there are many edges. In other words, if a node i has low flux value, then there are not many connections between the programming language prior to and subsequent to that language.

Calculation of Structural Virality. Here, we aim to measure the characteristic of how programming languages are influenced by one another over time, either through languages that gain its popularity through a single, large broadcast or through multiple generations with any one individual directly responsible for only a fraction of the total adoption. The structural vitality definition that we use here is exclusively concerned with characterizing the structure of the observable adoption patterns that arise from some unobserved generative process, and the goal is to characterize a particular property of the network structure and to disambiguate between the broadcast and multigenerational branching schematics (11).

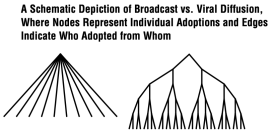

Fig. 4. Illustration of the two modes of distribution in a tree (broadcast on the left, viral on the right) from (11)

To find a metric that allows us to cover the goals that we have from the calculation of structural vitality, it is important for us to understand criteria that we want our calculated value to satisfy. These criteria are based on Figure 4, which shows the different types of distribution of a tree. There are three main things that we want our metric to have: "First, for a fixed total number of adoptions in a cascade, structural virality should increase with the branching factor of the structure: specifically, it should be minimized for the broadcast structure on the left of Figure 4 and should be relatively large for structures with a high branching factor, as on the right of Figure 4. Second, for a fixed branching factor, structural virality should increase with the number of generations (i.e. depth) of the cascade; that is, all else equal, larger branching structures should be more structurally viral than smaller ones. Finally, and in contrast with multigenerational branching structures, larger broadcasts should not be any more structurally viral than smaller broadcasts; hence we require that, for the extreme case of a pure broadcast, structural virality be approximately independent of size" (11). There are few natural choices such as looking at the depth or average depth of nodes, however for a case which involves large broadcast with just one, long, multigenerational branch or one long multigenerational branch with large broadcast at the end has misleading calculated values. To address the shortcoming of the two metrics, we turn our head to the Wiener Index which provides a "continuous measure of structural virality, with higher values indicating that adopters are, on average, farther apart in the cascade, and thus suggesting an intuitively viral diffusion event" (11).

Wiener index of a directed graph is defined as the sum of all distances of each ordered pair of nodes and they follow the equation below.

$$W(D) = \Sigma_{(u,v)\in V(D)\times V(D)} d_D(u,v) = \Sigma_{u\in V(D)} w_D(u) \quad [4]$$

Here, we assume that if there is no path between the two nodes, then the distance between them would be 0.

$$d_D(u,v) = 0 \quad [5]$$

Both equations are from (12). We needed this assumption since our network consists of nodes that have no path that connects them to some of the other nodes. The definition for the Wiener index for directed acyclic graphs given in (12) is strictly for a tree. However, it was the closest measurement of the Wiener index we could find for directed graphs and we decided to use them since they satisfy the three criteria that we want the metric to have, of what we want higher values and lower values to represent. Let $T(a)$ is the set of vertices x where there exists a directed path from x to a, and $S(a)$ is the set of all vertices x where there exists a directed path from a to x. Then, let $t(a) = |T(a)|$ and $s(a) = |S(a)|$. Then, we have the following equation for our calculation of the Wiener index for directed trees (12):

$$W(T) = \sum_{ab\in A(T)} t(a)s(b) \quad [6]$$

Results

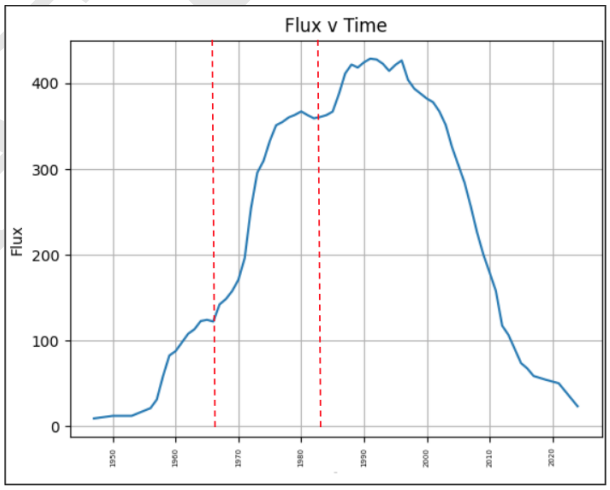

Fig. 5. Flux μ_i for the network of programming languages plotted as a function of year of language created. The dotted lines highlight the dips in the flux and correspond roughly to the two widely acknowledged shifts in the use of programming languages by the people in the industry.

The dotted lines in Figure 5 highlight the dips in the flux, and they correspond to temporal divisions between how programming languages are used and what people want from the programming languages, a paradigm shift in the programming languages. These dips represent how programming languages prior to the dotted year is little cited by the languages subsequent to the dotted year, in other words, these dotted lines allow us to make a reasonable guess that there were changes in approach to programming languages that made older languages obsolete (4). We also observe that there is a rapid increase in flux after each of the dotted line, suggesting that once there is a paradigm shift in the programming language, they make programming languages more accessible and perhaps easier to use or gives programmers more control over the language, giving rise to more languages that are specialized

in some way. The first dotted line corresponds to a paradigm shift in the mid 1960s, where programming languages LISP and ALGOL spread for their revolutionary developments in the field of programming. LISP is the second-oldest high-level programming language that is still commonly used today, as it started many core ideas in computer science such as tree data structures, recursion, and conditionals (13). ALGOL was another language that was developed early on in this field, and therefore became the standard for imperative coding languages by introducing concepts like code blocks and having formal grammar within the language itself (14).

The second dotted line corresponds to the paradigm shift in the 1980s and it corresponds to the spread of C language. C is another incredibly influential programming language. After its development in the 1970s, it spread rapidly during the 80s and became one of the most widely used programming languages at the time. Due to its ability to appropriately use targeted CPUs of machines, the language is still used in operating systems and device drivers today (5).

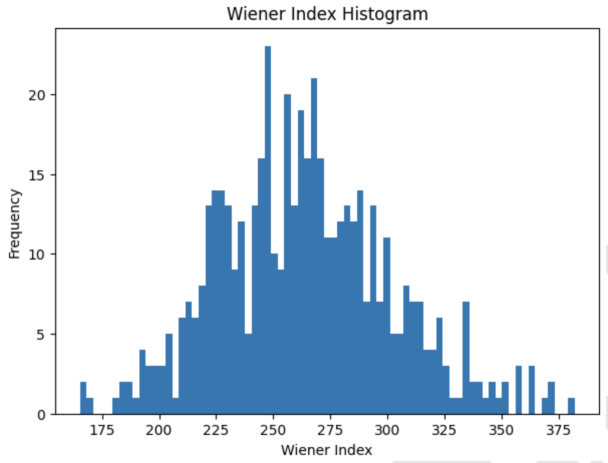

Fig. 6. Illustration of Wiener Index for the 500 ensemble of random graphs vs. Frequency

Result of Wiener Index. We calculated the Wiener index for the 500 ensemble of random graphs that we created using the configuration model and for the actual network of our interest. We found that the Wiener index for our actual network is 189.4, and the result of the Wiener index for the 500 ensemble of random graphs is illustrated in the Figure 6. It is trivially true that the Wiener index for our actual network is an outlier when referencing Figure 6. This shows that our network is more wide than long. When a newer and better programming language rolls out to the market, other developers would take influence from them instead of the older and outdated programming languages. This creates a wide network. Moreover, our network cannot be long. Programming languages do not stick for a long time as there is always a demand for new features and there is a paradigm shift as proven above.

Conclusions and Discussion

From the above analysis and conclusions, we can conclude that the network fails to refute that programming languages reflect periodic shifts within the industry. When calculating the flux against the year, we can see two distinct dips that correspond to two periods of programming history that created revolutionary languages like C and LISP. Additionally, we can see based on our visualization of the network itself that the nodes with high out-degree correspond with the 1960s, 70s, and 80s. Thus, from our network analysis, we fail to refute that programming languages reflect periodic shifts within the computing industry.

Some future work that could expand on this project include adding more data about each programming language to analyze other factors that could impact the influence and importance of a language. We only focused on the year a programming language was created. However, various qualities of a language could also impact which languages they influence and why they specifically are influenced by others. For example, programming paradigms could dictate why certain languages create certain communities, such as object-oriented or functional programming. Another extension of this project could be looking into multiple versions of languages to see if certain versions are more influential than others. Since programming languages are constantly evolving and receiving updates, it could be insightful to look into how the versions of these languages adapt accordingly.

Limitations. Some limitations of this project include the various drawbacks of using Wikipedia as our data source. As Wikipedia is a resource that anyone on the internet can revise at any time. Therefore, this dataset and network could become outdated as more information is added to the list of programming languages on Wikipedia. In addition to the potential for constant edits on Wikipedia pages, there can also be inaccuracies in the data itself. Wikipedia's formatting is not consistent across all pages about programming languages, so after scraping the data from Wikipedia, we had to go through the data and manually fix any inaccuracies. Thus, a large limitation in this project is the use of Wikipedia.

Another limitation is our analysis of the Wiener index. The derived equation and calculation is used specifically for directed trees. Our graph is not a tree, but a directed acyclic graph. Thus, since a directed tree is a type of directed acyclic graph, we decided to use this metric to estimate the Wiener index. However, our graph does not satisfy every necessary condition to make this calculation completely accurate. Therefore, deriving a broader equation for the Wiener index would make this result more applicable to this network of programming languages' influence.

Code Repository

For all of our code that we created and developed for this project, please reference this link: https://github.com/bellalam01/168project

ACKNOWLEDGMENTS. The authors would like to acknowledge Professor Mason Porter for his support and guidance throughout this project and also TA, Grace Li, for explaining key concepts of networking.

1. A Ferguson, A history of computer programming languages. (2004).
2. Wikipedia, List of programming languages — Wikipedia, the free encyclopedia (https://en.wikipedia.org/wiki/List_of_programming_languages) (2022) [Online; accessed 16-December-2022].
3. Endo, Lam, Zhao, 168-project — github (https://github.com/bellalam01/168project) (2022) [Online; accessed 16-December-2022].
4. B Karrer, MEJ Newman, Random graph models for directed acyclic networks. *Phys. Rev. E* **80** (2009).

5. Wikipedia, C (programming language) — Wikipedia, the free encyclopedia (http://en.wikipedia.org/w/index.php?title=C%20(programming%20language)&oldid=1127424876) (2022) [Online; accessed 16-December-2022].
6. Wikipedia, Rust (programming language) — Wikipedia, the free encyclopedia (http://en.wikipedia.org/w/index.php?title=Rust%20(programming%20language)&oldid=1127855553) (2022) [Online; accessed 16-December-2022].
7. Wikipedia, Swift (programming language) — Wikipedia, the free encyclopedia (http://en.wikipedia.org/w/index.php?title=Swift%20(programming%20language)&oldid=1123562941) (2022) [Online; accessed 16-December-2022].
8. NetworkX, hits — networkx (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.link_analysis.hits_alg.hits.html) (2022) [Online; accessed 16-December-2022].
9. Wikipedia, Python (programming language) — Wikipedia, the free encyclopedia (https://en.wikipedia.org/wiki/Python_(programming_language)) (2022) [Online; accessed 16-December-2022].
10. NetworkX, greedy_modularity_communities — networkx (https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.community.modularity_max.greedy_modularity_communities.html) (2022) [Online; accessed 16-December-2022].
11. S Goel, A Anderson, J Hofman, DJ Watts, The structural virality of online diffusion. *Manag. Sci.* **62**, 180–196 (2016).
12. M Knor, R Škrekovski, A Tepeh, Some remarks on wiener index of oriented graphs. *Appl. Math. Comput.* **273**, 631–636 (2016).
13. Wikipedia, Lisp (programming language) — Wikipedia, the free encyclopedia (http://en.wikipedia.org/w/index.php?title=Lisp%20(programming%20language)&oldid=1127862716) (2022) [Online; accessed 17-December-2022].
14. Wikipedia, ALGOL — Wikipedia, the free encyclopedia (http://en.wikipedia.org/w/index.php?title=ALGOL&oldid=1117739141) (2022) [Online; accessed 17-December-2022].