Network Analysis for Programming Languages and Their Fields

Taksh Girdhar[27012699] and Kussh Satija[80384878]

University of British Columbia Okanagan, Kelowna BC V1V 1V7, Canada

Abstract. The overwhelming abundance of programming languages, and frameworks in computer science overwhelms students and raises concerns about which language to focus on in the process of academic and professional growth. This study looks at the trends in programming languages for the role of a computer science job by analyzing job descriptions for insights into their versatility, co-occurrence patterns, and beginner-friendly options supporting specialization. Using a bipartite network model of programming languages and computer science fields, it applies normalized degree and betweenness centrality with Leiden community detection to find key relationships and identify trends. These results indicate the high versatility of languages like Python, Java, and Julia- foundational tools in a wide number of fields-while other languages, like Ruby, serve as transitional languages from the foundational learning to high-level specializations. Community detection pinpointed the clustering of some programming languages, such as TensorFlow in more data-driven fields, C++ with cybersecurity, and embedded systems. From here, the students receive a data-driven guide for skill prioritization to decrease learning path uncertainty and help their future career prospects.

Keywords: Programming Languages, Frameworks, Network Analysis, Centrality Metrics, Leiden Community Detection, Programming Language Versatility, Beginner-Friendly Programming Languages, Programming Language Co-Occurrence, Data-Driven Learning Pathways, CS Field Specializations, Bipartite Graphs, Transitional Programming Languages

1 Introduction

As the computer science industry continues to expand and change, students often find themselves overwhelmed by the sheer variety of programming languages, and frameworks available (Akdur,2022). Determining which skills to acquire to stay relevant in a specific field or maintaining flexibility to specialize in the future is an overwhelming challenge that causes inefficiency and uncertainty about aligning efforts with industry demands and career goals (Rabai et al., 2015). The following is the study's problem statement. "The overwhelming abundance of resources in computer science presents a challenge for computer science students, making it difficult to identify and prioritize what skills and knowledge to learn to build a strong foundation in the field and ultimately, leading to uncertainty about where to focus their efforts."

This research investigates programming language trends in Computer Science (CS) roles by analyzing job descriptions to uncover insights into their versatility, co-occurrence patterns, and beginner-friendly languages that serve as gateways to advanced specializations. Through these analyses, the following research questions aim to equip students with data driven recommendations by answering the following research questions for efficient and effective learning pathways, benefiting both their academic journey and professional career development.

1. What programming languages are most central in their versatility across different fields?
2. Which programming languages are optimal for beginners while facilitating further specialization in various computer science fields?
3. Which programming languages tend to co-occur in the same job descriptions?

2 Related Work (Literature Review)

The reviewed research papers provide several valuable perspectives into network analysis methodologies that complement the goals of this study in helping students to navigate the complex landscape of programming languages in CS. Research into course-prerequisite networks illustrates how network science can model and optimize learning pathways and serves as an inspiration for mapping programming languages and their relations to academic and career development (Stavrinides et al. , 2023). The skill-proximity network analysis underlines the utility of the examination of co-occurrence and compatibility principles, which is in tune with the understanding of programming language versatility and their interlinkages in job descriptions (Waters et al., 2022). The study on trends in CS shows how network analysis can uncover relationships and dominant CS fields, providing criteria for identifying beginner-friendly programming languages that can act as gateways to specialization (Kalhor, 2023). Finally, the research on the use of programming languages in academia and industry underscores the importance of aligning language adoption with educational and professional goals, offering insight into how students can be best guided toward skill acquisition that bridges learning and implementation (Rabai et al. ,2015). Taken together, these works provide the network-based approach used in this study to support students in prioritizing skills and creating adaptable learning pathways in CS.

Our research paper attempts to fill critical gaps identified in prior work across several domains, focusing on the use of programming languages in academia and its alignment with the broader work trends. Course-prerequisite networks have focused on curriculum visualization and identification of key courses, but it has not recognized the foundational role of programming languages in structuring academic and professional growth. This study builds on this by incorporating programming languages as integral components within professional settings. Similarly, Skills approximate paper examines skill proximities but did not investigate the role of programming languages as core technical skills transferable across occupations. This paper, therefore, brings to light the potential of programming languages as a transitioning skill set. Moreover, analysis of research trends in CS stressed interdisciplinarity but failed to investigate the contribution of programming languages to fostering connections between fields. The paper discusses the relationship between programming languages and interdisciplinary growth in both academic and professional fields. Lastly, programming language use in US academia and industry identified a disconnect between academia and industry but did not analyze the underlying trends that drive language choices in academic and professional settings. This study addresses that gap by providing a detailed examination of programming language trends, thereby offering insights into their adoption patterns across different academic and professional contexts.

3 Methodology

In this section, we describe the process of gathering and preparation of the data followed by a definition of the network built on the data and the metrics used in our analysis.

3.1 Data Collection

The data for this project was sourced from two primary datasets found on Kaggle (Regmi, 2019; Georges, 2019). We combined two datasets containing a total of 12,000+ unique job postings from distinct IT-related job roles. These roles reflect the real-world job requirements and were extracted in February 2019. Each entry in the dataset pairs a job title with a detailed job description. We used a supplementary dataset of skillsets that included a curated list of the top 20 programming languages and frameworks across specific fields. Together, the Kaggle datasets and Skillset serve as a reference for matching job descriptions with relevant fields and programming languages.

3.2 Data Preprocessing

During the data preprocessing phase, using Jupyter the two datasets were merged and missing values were addressed, resulting in the creation of a raw data csv file. Then, a Python-based job description and skills parser, developed with the spaCy natural language processing library was used instead of brute-force searching for keywords. The parser leverages a skill matcher built using the PhraseMatcher utility on an en_core_web_sm pre-trained spaCy model and processes raw data to extract structured information about fields, programming languages and frameworks from job descriptions. The combination of the Kaggle dataset and the Skillset Excel Sheet enables the parser to identify and quantify relevant skills, to create Data.csv. Lastly, the data was aggregated by counting the occurrences of each CS field and language. Each row represents a job posting and includes columns for extracted fields, programming language/framework, and their respective occurrence counts. This dataset highlights the technical skills demanded for various IT-related roles by capturing mentions of programming languages, and frameworks within the provided job descriptions.

3.3 Network Structure

The Bipartite Graph, $G = (U, V, E)$ where $U = \{L_1, L_2, \dots, L_n\}$ represents the set of programming languages and frameworks and L_i is a specific language (e.g., Python, Java, C++). $V = \{F_1, F_2, \dots, F_n\}$ represents the set of fields, where F_j is a specific field (e.g., Data Science, Web Development, Systems Programming). Let $E \subseteq U \times V$ be the set of edges, where each edge $(L_i, F_j) \in E$ represents the fact that programming language L_i is required in field F_j. Note that for simplicity the network will be referenced as the Programming Languages and CS Fields Network even though it also contains frameworks.

3.4 Normalized Degree Centrality

Normalized degree centrality is a measure of the importance or influence of a node in a graph based on the number of connections (edges) it has, scaled to account for the size of the graph. The formula for calculating betweenness centrality is as follows,

$$\mathrm{NDC(i)} = \frac{\mathrm{deg(i)}}{N-1}$$

where deg (i) is the degree of node i, and N is the total number of nodes in the graph (Newman, 2018). Pertaining to our study, the normalized degree centrality provides valuable insight into the relative importance and connectivity of both fields and programming languages/frameworks within our graph. This measure highlights the versatility of certain programming languages and the demand for specific fields, offering valuable insights into the skills most relevant to current job market trends (i.e which programming languages are widely used across multiple fields).

3.5 Betweenness Centrality

Betweenness Centrality measures the frequency with which a given node lies on one of the shortest paths between all other pairs of nodes in the network. The formula for calculating betweenness centrality is as follows,

$$B_i = \sum_{st} n_{st}^i$$

where n_{st}^i is 1 if the node i lies on the shortest path from s to t and 0 if no such path exists (Newman, 2018). We utilize betweenness centrality because it can be useful to determine which programming languages are transitional, facilitating the transfer of skills and knowledge to more specialized fields.

3.6 Leiden Coefficient

The Leiden coefficient measures how well a given node belongs to its own community compared to other communities. It improves on the modularity coefficient by considering not only the connection density within communities but also ensuring that the communities are well-separated from one another. The formula for calculating the Leiden coefficient as follows,

$$L = \frac{1}{2m} \sum_{ij} (A_{ij} - \frac{k_i k_j}{2m}) \delta(c_i, c_j).$$

where A_{ij} is the adjacency matrix element that represents the edge between nodes i and j, k_i and k_j are the degrees of nodes i and j, m is the total number of edges in the graph, c_i and c_j are the community assignments of nodes i and j, and $\delta(c_i, c_j) = 1\ or\ 0$ if nodes i and j belong to the same community or not (Newman, 2018). The Leiden algorithm was chosen over the Louvain algorithm because a major shortcoming of the Louvain algorithm is producing “arbitrarily badly connected communities or even disconnected” whereas the Leiden algorithm is able to “yield communities that are guaranteed to be connected” and given the possible implications of the results, the authors deemed the Leiden Community Detection Algorithm to be more accurate for this study (Traag et al., 2019). The resulting communities will provide insight into how different CS fields are related through their specific programming languages/framework.

4 Results and Discussion

Refer to Appendix 1 for access to code for data preprocessing, calculations and results.

4.1 Programming Languages and CS Fields Network Graph and Structure

In this part, we present the CS Fields and Programming Languages Network in a bipartite layout to understand employer expectations of CS students' skillsets.

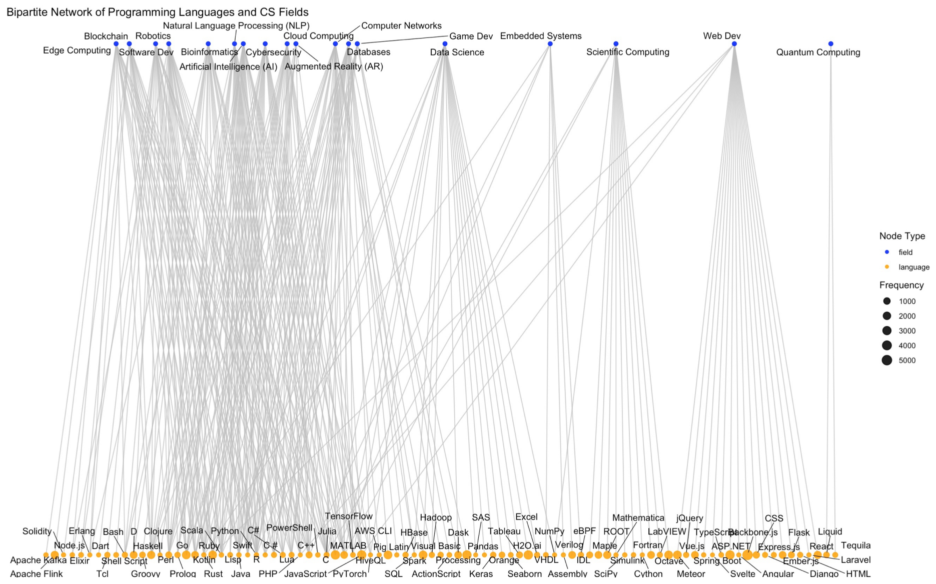

Fig. 1. The Bipartite Network of Programming Languages and CS Fields. The types of nodes are field (CS field) represented by a blue color and the language (language or framework) represented by an orange color. The size of a node corresponds to the frequency of appearance in the dataset of job descriptions

The network consists of a total of 109 nodes of two types (CS fields = 18 nodes, programming languages and frameworks = 91 nodes) and 280 edges. It is not a connected network and with a density of 0.0476.

4.2 Normalized Degree Centrality

Firstly, we calculate the centrality metric known as normalized degree centrality to assess the connectivity of the network and determine central languages in CS fields.

The network visualization in Figure 2 depicts a comprehensive overview of the interconnections of the nodes using the size and color of each node. Since these attributes are based on the normalized degree centrality, the visualization illustrates the level of connectivity of programming languages and frameworks in an interdisciplinary setting.

Consequently, this indicates their relevance in the broader ecosystem of computer science. As supported by Table 1, languages such as Python, Java, Julia and C++ stand out as the most central in the visualization, marked by larger and darker nodes. These languages are highly interconnected, suggesting that these languages have widespread use across multiple CS fields. In contrast, languages with smaller nodes and lighter colors, such as the languages and frameworks of Quantum Computing including Tequila and Liquid are nodes with minimal connections, as supported by the visualization. This indicates languages and frameworks with a low normalized degree centrality are less commonly used or have more specialized applications.

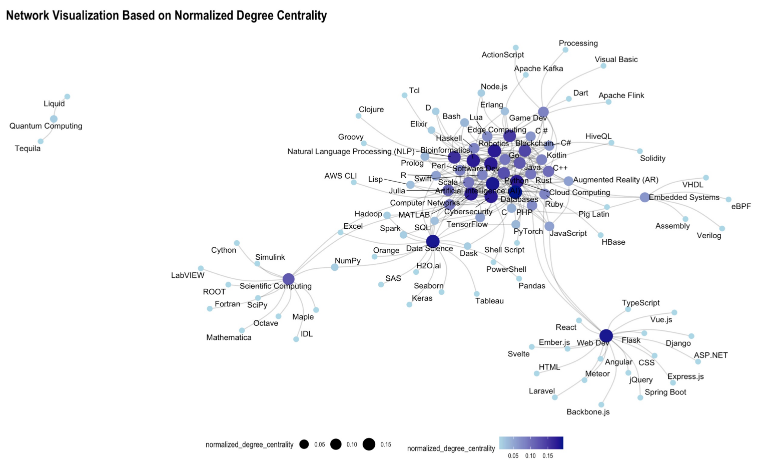

Fig. 2. Network visualization of Programming Languages and CS Fields where the nodes which represent the programming languages and CS fields indicate the normalized degree centrality based on color and size

Programming Language	Normalized Degree Centrality
Python	0.1296
Java	0.1204
Julia	0.1111
C++	0.1019
Go	0.0926
MATLAB	0.0926
Rust	0.0926
Scala	0.0926
Haskell	0.0833
Perl	0.8333

Table 1. Top 10 Programming Languages and Frameworks from normalized degree centrality

Finally, the normalized degree centrality provides clarity to our research question "What programming languages are most central in their versatility across different fields? " by highlighting which languages are most central to multiple CS fields. We can answer this using Table 1, i.e. identifying the languages with the highest centrality scores, such as Python and Java which are positioned at the core of the network visualization. These languages and frameworks are highly interconnected with a variety of other technologies indicating they are foundational and have diverse usage across multiple CS fields.

4.3 Betweenness Centrality

Next, we calculate the betweenness centrality to assess the role of programming languages as intermediaries within the network, helping to understand their connectivity and influence in an interdisciplinary context.

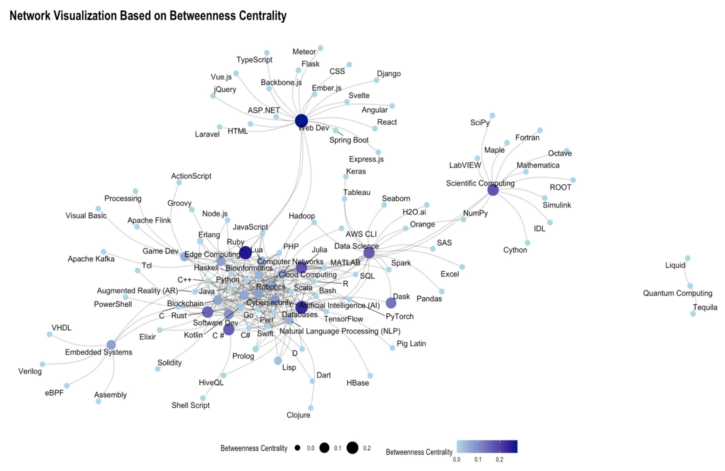

Fig. 3. Network visualization of the Programming Languages and CS Fields where the nodes represent the betweenness centrality of each programming language/CS field by the size and color of the node

At its core, the betweenness centrality metric answers the research question "Which programming languages are optimal for beginners while facilitating further specialization in various computer science fields?" and by examining the network visualization in Figure 3 which illustrates the betweenness centrality of programming languages/frameworks and CS fields. In this visualization, the positioning of nodes representing languages and frameworks emphasizes their role as bridges that connect different CS fields. As highlighted in Table 2 below, languages with a high betweenness centrality score such as Ruby, Julia, C# are highly central in the visualization indicating they act as key intermediaries that link multiple CS fields. They provide a foundational understanding of programming concepts that offer a pathway to more advanced specializations within CS fields. In contrast, languages such as ActionScript and Liquid display low betweenness centrality in the visualization, suggesting that they are less connected and serve more specialized or isolated roles. These languages are better suited for higher-level learning, typically after mastering foundational programming skills. Therefore, analysis of the betweenness centrality metric provides insights into how programming languages with high betweenness centrality scores can act as building blocks for beginners to further explore diverse areas in computer science.

Programming Language	Betweenness Centrality
Ruby	0.2648
Julia	0.1751
C#	0.1611
Rust	0.1551
Dask	0.1294
Lua	0.0507
Lisp	0.0298
Kotlin	0.0171
Erlang	0.0035
Prolog	0.0033

Table 2. Top 10 Programming Languages and Frameworks based on betweenness centrality

4.4 Leiden Community Detection

Moving on, we present the communities of the Programming Languages and CS Fields Network calculated using the Leiden Community Detection algorithm. Firstly, we calculate an optimal resolution parameter for community detection as presented Appendix 2. , followed by a visualization of the communities and their statistics.

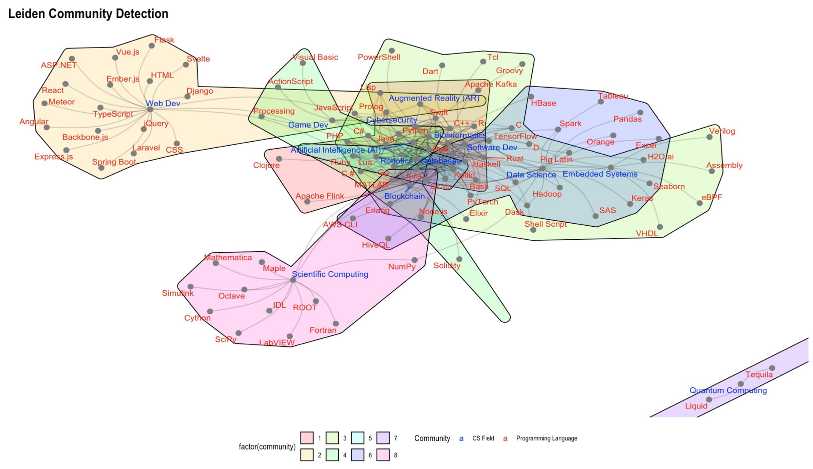

Fig. 4. The Programming Languages and CS Fields network, with communities detected via the Leiden Algorithm.

Figure 5 illustrates the Programming Languages and CS Fields Network, divided into communities identified using the Leiden Community Detection algorithm. The optimal resolution parameter, derived from the scatter plot in Appendix 1,was set at 1.475 resulting in 8 communities as outlined in Table 3 below. The largest community (community 3) includes Bioinformatics, Cybersecurity, Embedded Systems, and Software Development, with C++ as the most central programming language. Community 7, the densest, has Quantum Computing and Liquid as the central nodes, while community 6, focused on Databases, features TensorFlow as the most central programming language. This analysis addresses the research question, "Which programming languages tend to co-occur in the same job descriptions?" By applying the Leiden algorithm, we identified communities that highlight relationships between languages and fields. For example, C++ frequently co-occurs with Bioinformatics, Cybersecurity, Embedded Systems, and Software Development, indicating its common presence in job descriptions for these areas. Similarly, TensorFlow in community 6 suggests a strong connection between Data Science, Computer Networks, and Databases. These findings shed light on the languages most likely to appear together in job postings, helping CS students identify the skillsets needed for various career options and providing clarity for students uncertain about their future specializations.

Community	Size	Density	Most Central Programming Language Node	Most Central CS Field Node	CS Field Nodes Present
1	11	0.2727	Java	Artificial Intelligence	Artificial Intelligence Edge Computing Natural Language Processing
2	20	0.1000	JavaScript	Web Dev	Augmented Reality Web Dev
3	23	0.1462	C++	Bioinformatics	Bioinformatics Cybersecurity Embedded Systems, Software Dev
4	11	0.2545	Python	Robotics	Blockchain, Game Dev, Robotics
5	6	0.3333	Go	Cloud Computing	Cloud Computing
6	21	0.1238	TensorFlow	Databases	Computer Networks Data Science Databases
7	3	0.6667	Liquid	Quantum Computing	Quantum Computing
8	14	0.1429	Julia	Scientific Computing	Scientific Computing

Table 3. Statistics of Leiden communities within the Programming Languages and CS Fields Network

4.5 Discussion

In this section, we interpret the significance of our findings and explore their implications for the field of computer science. The purpose of our analysis was to explore the structure and relationships within the programming languages and CS fields network and we aimed to provide research-backed assistance for CS students on their learning journey based on our results.

Regarding our first research question, the languages identified in Table 1 are highly central in the network, indicating their widespread use across multiple CS fields. These findings align with Vailshery's (2024) survey, which reports Python and Java as the most widely used languages among 60,000 developers. The TIOBE index (Jansen, 2022) also supports our results, ranking Python as the most popular, followed by C++ and Java, further validating our findings. However, while Vailshery's research shows JavaScript as the most used language, our study indicates Python holds this position. Moreover, some results of our study did not appear or did not have any significance in any existing research highlighting how of our findings can bring newfound impact. The key difference lies in the data collection methods employed in our study, which diverge from those in existing research. Both Vailshery’s survey and the TIOBE index primarily analyze data based on current skillset profiles which reflect the learning journeys of developers who were students in earlier periods, with access to different resources than those available to today's CS students. In contrast, our study focuses on the current demands of employers, providing a more accurate representation of the skill sets required in the job market today. Interestingly, despite their lower presence in previous studies, languages like Julia, Scala, Haskell, and Perl may still prove valuable in specific contexts. For example, Julia is a language that is "designed to possess the speed of C, the general programming ability of Python, and the statistical power of R" (Crabtree, 2022). The Julia has straightforward syntax and takes elements of existing data programming languages making it ideal for beginners such as CS students. In this context, Julia offers an easy learning curve for CS students allowing them to understand fundamental concepts while allowing access to advanced specializations through Julia libraries such as GameZero for game development and packages such as Dash.jl which provides access to the Dash ecosystem thus allowing users to create web application in Julia itself (Crabtree, 2022). Therefore, we can conclude that our findings which identify the languages most central in their versatility across multiple CS fields, despite not aligning with existing research, are highly beneficial for CS students. These results can help broaden their knowledge base, making it more adaptable and facilitating further specialization in various CS fields. By highlighting the languages in Table 1, we offer clear guidance for CS students on where to focus their efforts for a strong foundation, as these languages are central to the CS ecosystem and interconnected across fields like software development and data science. A deep level understanding of these languages equips students with versatile skills essential for many specializations, helping them navigate the abundance of learning resources and pursue diverse career paths.

Regarding our second research question, the results of our study uncovered how certain languages can act as key intermediaries in between CS fields. Languages and frameworks such as Ruby, Julia and C++ , as shown in Table 2, were identified as having high betweenness centrality, indicating that they facilitate connections across a variety of CS fields. Although we did not find any existing research or data explicitly indicating which languages aid in the transition to specialized CS fields, we identified an article by Hewner et al. (2011) that examined the process of how CS majors choose a specialization. Their analysis of interviews conducted with student advisors, undergraduate students and graduate students determined four driving forces to this decision. First, students use their enjoyment of individual classes as indicator of how well they fit in a specialization. Second, many students do not conduct adequate research, leading them to purse a specialization based on misconceptions. Third, students tend to rely on curriculum to protect them from making poor educational choices. Fourth, students typically lack a personal vision for their future career with a computer science degree. Not only does this article directly relate to the crux of this study's problem statement, it also highlights why the findings of this study, in particular the betweenness centrality, are relevant and highly beneficial to CS students. For instance, Ruby is a flexible, object-oriented programming (OOP) language with a simple syntax that is similar to natural language allowing users to finish complex tasks with shorter code and as a result, boosts teamwork and productivity (Danao, 2023). Ruby has widespread usage including but not limited to; Software Engineering, Bioinformatics, Robotics (Seaton, 2022). Most notably, Ruby's framework Ruby on Rails for web development is used to power a variety of popular website and application platforms such as Airbnb and Github (Danao, 2023).

This raises the question, "Why not utilize Ruby for learning and implementation in CS specializations from the beginning?". Despite, the growing community and framework support for Ruby, it is not an industry standard for most CS specializations. However, its simplistic features and syntax can allow for the learning of fundamental concepts and then progressing to more complex implementation later. To illustrate this point, consider the example of data processing and management in Ruby. While Ruby has the capability to perform data processing and management tasks, it does not nearly have enough libraries or support as the industry standard of data science and machine learning, Python (Danao, 2023). This analysis demonstrates why Ruby is an ideal transitional programming language, enabling beginners to grasp fundamental concepts such as OOP and data processing/management techniques through its simple and clear syntax. Once students pursue a specific CS specialization, they can then transition to understanding more complex implementations required by the industry, such as those in Python. Furthermore, the straightforwardness of Ruby can allow CS students to learn fundamental concepts, through implementation, of multiple CS specializations effortlessly and more quickly, diversifying their knowledge base. Overall, the concept of learning transitional programming languages to facilitate further specialization in CS fields, while also addressing the issues outlined by Hewner et al. (2011), empowers CS students to take control of their learning paths towards specialization. By providing an effective and comprehensive approach to learning, this goal can be realized through the programming languages listed in Table 2.

Regarding the third question, the findings of this study reveal the formation of communities among various CS fields highlighting the relationships between specific programming languages and these fields. As illustrated in Table 3, languages and frameworks such as C++, Python, and TensorFlow emerge as the most central nodes of their type, each associated with multiple CS fields within their respective communities that also have its own most central CS field node. Our findings align with the conference paper Information Technology Roles and Their Most-Used Programming Languages (Dada et al., 2022), which identified and ranked the 'most-used' programming languages for 23 predefined IT-related job roles based on surveys conducted among StackOverflow forum members. Dada et al.'s study identified C++ as the most used language across various IT roles, from Mobile Developers to System Administrators, supporting our finding that C++ is the most central programming language in Community 3, which includes the highest number of CS field nodes. Additionally, while Dada et al. identified Python as the most used language for data science and academic research, our analysis using the Leiden detection algorithm highlights Python as central to the community containing Blockchain, Game Development, and Robotics. This difference arises from Dada et al.'s focus on percentage frequency data from the survey, whereas our study emphasizes the interrelationships between CS fields through shared programming languages and frameworks. Python's extensive libraries make it ideal for handling complex tasks in these CS fields (Mishra, 2024). Similarly, our findings show that TensorFlow is the most central framework for the fields of Data Science, Computer

Networks, and Databases. TensorFlow's ability to handle large-scale computations using multidimensional data flow graphs (Banoula, 2024) makes it well-suited for these CS fields. Unlike Dada et al.'s identification of languages that are industry standards, our study reveals programming languages and frameworks that are most interconnected within their respective CS field communities. This distinction highlights their potential for teaching foundational concepts that would be transferable across multiple specializations, making them ideal for students. Furthermore, our analysis provides valuable insights into future career prospects across diverse CS fields.

5 Conclusion

This study aimed to explore the structure and interrelationships within a network of programming languages and CS fields. It focused on identifying central languages and frameworks that are versatile across multiple CS fields, validating the concept of learning transitional programming languages that facilitate specialization, and understanding how these languages co-occur in job descriptions. By providing data-driven insights, the aim was to aid CS students in making informed decisions about their learning paths.

The findings in Table 1 revealed that several programming languages and frameworks are highly central in the network, indicating their widespread use across various CS fields. Table 2 identified transitional languages that serve as stepping stones for beginners, helping them understand foundational concepts before specializing. The Leiden algorithm results in Table 3 also highlighted how certain languages connect multiple CS fields, opening career prospects for students and reducing uncertainty in their specialization choices. This study offers a guide for CS students to navigate the abundance of resources and focus on key skills. Our results are particularly relevant to CS students, offering insights into how they can develop a well-rounded skillset that remains relevant in the rapidly evolving CS landscape. However, many students may not have the sufficient knowledge of network analysis to interpret these findings accurately. To address this, we have provided an interactive HTML widget in the Appendix 3, allowing students at any stage of their learning journey to understand the relationships and results from this study. The aim is not to make decisions for students, rather to offer tools and opportunities to do so. The implications also extend beyond students to academia and industry professionals. Academia can use these findings to shape curriculum and course design, ensuring they align with current industry needs. The study also emphasizes the importance of continuous learning to enhance adaptability and competitiveness in a rapidly changing field making the results of this study beneficial for professionals as well.

There are a few limitations to this study. The dataset used job descriptions from Feb 2019, which may not fully reflect current industry trends. Additionally, the study's focus on centrality and community detection does not account for all factors that influence language popularity, such as regional or organizational preferences. As a result, newer languages may not have been adequately represented. Future research could broaden the scope by including more CS fields, languages, and frameworks, as well as incorporating qualitative data from surveys of IT professionals and industry-specific demands. Further analysis using additional metrics and community detection techniques could provide deeper insights into the trends and relationships within computer science.

Appendix 1.

A github repository with all code for data-preprocessing, calculations and results can be found at: https://github.com/Takshg/COSC421_Project/tree/main

Appendix 2.

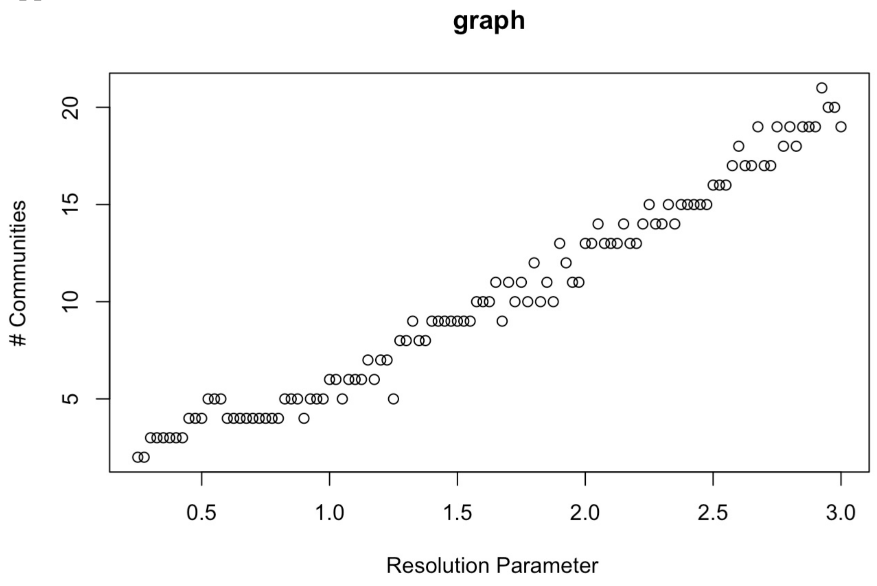

Fig. Scatter plot showing the relationship between the resolution parameter and the number of communities in the graph. The resolution parameter corresponding to the point with the smallest rate of change was then identified as the optimal value (1.475) for the Leiden community detection algorithm.

Appendix 3.

The interactive HTML widget version of the Programming Languages and CS fields Network can be found at:
https://github.com/Takshg/COSC421_Project/blob/main/Interactive_Network_EndProdduct.html

References

1. Akdur D (2022) Analysis of Software Engineering Skills Gap in the industry. ACM Transactions on Computing Education 23:1–28. doi: 10.1145/3567837
2. Banoula M (2024) What is TensorFlow? In: Simplilearn.com. https://www.simplilearn.com/tutorials/deep-learning-tutorial/what-is-tensorflow. Accessed 4 Dec 2024

3. Crabtree M (2022) What is julia used for? 10 applications of Julia Programming. In: DataCamp. https://www.datacamp.com/blog/what-is-julia-used-for. Accessed 1 Dec 2024
4. Dada, O.A., Aruleba, K., Yunusa, A.A., Sanusi, I.T., Obaido, G. (2022). Information Technology Roles and Their Most-Used Programming Languages. In: Abraham, A., et al. Innovations in Bio-Inspired Computing and Applications. IBICA 2021. Lecture Notes in Networks and Systems, vol 419. Springer, Cham. https://doi.org/10.1007/978-3-030-96299-9_75
5. Danao M (2023) What is Ruby? an intro to the dynamic programming language. In: Lighthouse Labs. https://www.lighthouselabs.ca/en/blog/what-is-ruby. Accessed 2 Dec 2024
6. Georges, M. S. (2019). *IT Job Post Descriptions* [Data set]. Kaggle. https://www.kaggle.com/datasets/mscgeorges/itjobpostdescriptions
7. Hewner M, Guzdial M (2011) How CS majors select a specialization. Proceedings of the seventh international workshop on Computing education research 11–18. doi: 10.1145/2016911.2016916
8. Jansen P (2022) TIOBE Index. In: TIOBE. https://www.tiobe.com/tiobe-index/. Accessed 1 Dec 2024.
9. Kalhor G (2023) (PDF) analysis of Research Trends in computer science. In: Research Gate. https://www.researchgate.net/publication/376721151_Analysis_of_Research_Trends_in_Computer_Science_A_Network_Approach. Accessed 6 Dec 2024
10. Newman MEJ (2018) Networks. Oxford University Press, Oxford
11. Mishra V (2024) Top 10 Python Applications in Real World. In: GeeksforGeeks. https://www.geeksforgeeks.org/python-applications-in-real-world/. Accessed 4 Dec 2024
12. Rabai LBA, Cohen B, Mili A (2015) Programming language use in US academia and industry. In: Research Gate. https://www.researchgate.net/publication/284886874_Programming_Language_Use_in_US_Academia_and_Industry. Accessed 6 Dec 2024
13. Regmi, K. (2019). Jobs and Job Description [Data set]. Kaggle. https://www.kaggle.com/datasets/kshitizregmi/jobs-and-job-description
14. Seaton C (2022)In: The Ruby Bibliography. https://rubybib.org/. Accessed 2 Dec 2024.
15. Stavrinides P, Zuev KM (2023) Course-prerequisite networks for analyzing and understanding academic curricula - applied network science. In: SpringerOpen. https://appliednetsci.springeropen.com/articles/10.1007/s41109-023-00543-w#Sec2. Accessed 5 Dec 2024
16. Traag VA, Waltman L, Van Eck NJ (2019) From Louvain to Leiden: Guaranteeing well-connected communities. In: Nature News. https://www.nature.com/articles/s41598-019-41695-z#Sec10. Accessed 5 Dec 2024
17. Vailshery LS (2024) Most used languages among software developers globally 2024. In: Statista. https://www.statista.com/statistics/793628/worldwide-developer-survey-most-used-languages/. Accessed 30 Nov 2024.

18. Waters K, Shutters ST (2022) Skills-approximate occupations: Using networks to guide jobs retraining - applied network science. In: SpringerOpen. https://appliednetsci.springeropen.com/articles/10.1007/s41109-022-00487-7#Sec7. Accessed 5 Dec 2024.

Contributions

1. Taksh Girdhar: Methodology, Results and Discussion, Conclusion
2. Kussh Satija: Abstract, Introduction, Literature Review, Methodology