วารสารแม่โจ้เทคโนโลยีสารสนเทศและนวัตกรรม ปีที่ 11 ฉบับที่ 2 พฤษภาคม – สิงหาคม 2568

Received: 28 เม.ย. 2568 **Revised:** 30 ก.ค. 2568 **Accepted:** 15 ส.ค. 2568

กราฟความรู้เชิงเหตุและผลสำหรับการวิเคราะห์และแสดงวิวัฒนาการของภาษาโปรแกรมจากข้อมูลอินโฟบ็อกซ์ในวิกิพีเดีย

A Causal Knowledge Graph for Analyzing and Visualizing Programming Language Evolution from Wikipedia Infobox Data

ปรียาภรณ์ อินต๊ะโมง[1], ปรวัฒน์ วิสูตรศักดิ์[1], ชโลธร ชูทอง[2] และ จิรวรรณ เจริญสุข[2*]

[1]ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ, คณะวิทยาศาตร์ประยุกต์, มหาวิทยาลัยเทคโนโลยีพระจอมเกล้าพระนครเหนือ, กรุงเทพมหานคร

[2]ภาควิชาวิทยาการคอมพิวเตอร์และสารสนเทศ, คณะวิทยาศาสตร์ ศรีราชา, มหาวิทยาลัยเกษตรศาสตร์ วิทยาเขตศรีราชา, ชลบุรี

Preeyaphorn Intamong[1], Porawat Visutsak[1], Chalothon Chootong[2], and Jirawan Charoensuk[2*]

[1]Department of Computer and Information Science, Faculty of Applied Science, KMUTNB, Bangkok, Thailand

[2]Department of Computer Science and Information Technology, Faculty of Science at Sriracha, Kasetsart University, Sriracha Campus, Chonburi, Thailand

*Corresponding author: jirawan.charo@ku.th

Abstract

Understanding the evolutionary relationships between programming languages is crucial for software development, education, and research. However, current research lacks comprehensive tools for analyzing causal relationships and influence patterns across the diverse programming landscape. This study proposes a method for creating a causal knowledge graph that illustrates the evolution of programming languages by utilizing automated data collection and graph modeling to identify different levels of influence. The system processes structured English and Thai Wikipedia infobox data, analyzing and transforming it into a graph format using Neo4j. Graph visualization is performed using the

Cypher Query Language. The knowledge graph presents C-influenced programming languages across three evolutionary stages, supporting both English and Thai. Key features include displaying influence paths and illustrating languages with similar structures. The results demonstrate that graph-based representations effectively capture the evolution of programming languages, enabling learners, educators, and researchers to explore inter-language relationships.

Keywords: *Causal Knowledge Graph; Programming Language Evolution; Wikipedia Infobox Data; Neo4j Graph*

บทคัดย่อ

การความทำความเข้าใจในความสัมพันธ์เชิงวิวัฒนาการระหว่างภาษาโปรแกรมเป็นสิ่งสำคัญต่อการพัฒนาซอฟต์แวร์ การศึกษา และงานวิจัย อย่างไรก็ตาม งานวิจัยในปัจจุบันยังขาดเครื่องมือที่ครอบคลุมสำหรับการวิเคราะห์ความสัมพันธ์เชิงสาเหตุและรูปแบบของอิทธิพลในภูมิทัศน์ของภาษาโปรแกรมที่มีความหลากหลาย งานวิจัยนี้เสนอวิธีการสร้างกราฟความรู้เชิงสาเหตุที่แสดงให้เห็นถึงวิวัฒนาการของภาษาโปรแกรม โดยใช้การเก็บข้อมูลอัตโนมัติและการจำลองข้อมูลในรูปแบบกราฟ เพื่อค้นหาระดับของอิทธิพลในแต่ละชั้น ระบบนี้ประมวลผลข้อมูลอินโฟบ็อกซ์มีโครงสร้างจากวิกิพีเดียภาษาอังกฤษและภาษาไทย ซึ่งนำมาวิเคราะห์และแปลงเป็นกราฟด้วยฐานข้อมูลนีโอโฟร์เจ พร้อมการแสดงผลกราฟผ่านโดยใช้ภาษาคิวรีไซเฟอร์ กราฟความรู้ที่ได้แสดงความสัมพันธ์ของภาษาที่ได้รับอิทธิพลจากภาษาซีในสามระดับของการพัฒนา และรองรับทั้งภาษาอังกฤษและภาษาไทย จุดเด่นของระบบนี้ได้แก่ การแสดงเส้นทางของอิทธิพลระหว่างภาษา และการจัดกลุ่มภาษาที่มีโครงสร้างใกล้เคียงกัน ผลการศึกษาแสดงให้เห็นว่าการใช้กราฟช่วยสะท้อนวิวัฒนาการของภาษาโปรแกรมได้อย่างมีประสิทธิภาพ และเป็นเครื่องมือที่มีประโยชน์สำหรับผู้เรียน ครูผู้สอน และนักวิจัยในการสำรวจความสัมพันธ์ระหว่างภาษาโปรแกรมต่าง ๆ

คำสำคัญ: *กราฟความรู้เชิงเหตุและผล; วิวัฒนาการของภาษาโปรแกรม; ข้อมูลอินโฟบ็อกซ์จากวิกิพีเดีย; กราฟนีโอโฟร์เจ*

1. Introduction

Understanding the evolution of programming languages is crucial for studying software development and technology. It enables researchers and students to effectively select appropriate languages to learn (Meyerovich & Rabkin, 2013). Valverde and Solé (2015) studied the evolution of programming languages based on adoption patterns and explored the relationship between evolutionary theory and programming language development. Systematic literature reviews have also examined the applications of knowledge graphs in software engineering contexts (Chen et al., 2020; Wang et al., 2017). However, a gap remains in creating comprehensive visual representations of programming language relationships using structured data.

This research aims to develop a knowledge graph that illustrates the relationships among various programming languages. Wikipedia serves as a reliable source under free documentation licenses, allowing for the distribution and modification of content. We propose a methodology that involves collecting data on C-based programming languages by extracting information from Wikipedia infoboxes. Data extraction is performed on both English and Thai Wikipedia, focusing on the "influenced by" and "influenced" fields. Additionally, a dictionary is used to ensure data consistency.

The knowledge graph is constructed using Neo4j, a graph database system that efficiently manages interconnected data in the form of nodes and relationships. The Cypher Query Language is used to manipulate and query graph data. This method demonstrates the efficiency of the graph-based approach in achieving completeness and eliminating redundancies. The resulting system offers a valuable tool for examining programming language relationships and creating knowledge graphs from publicly available data sources.

2. Literature reviews

2.1 Knowledge Graph Construction and Modeling

Knowledge graphs are graph-based data structures that represent knowledge units through nodes and relationships, where nodes represent entities and relationships indicate connections via semantic edges (Ji et al., 2021). This approach enables systems to understand contextual overviews while supporting automated decision-making and analytical capabilities. Knowledge graphs transform raw data into meaningful knowledge, making them fundamental to artificial intelligence and semantic web technologies (Cheng, 2022). Their construction involves entity recognition, relationship identification, and semantic integration from heterogeneous data sources.

Cheng (2022) demonstrated the effectiveness of knowledge graphs in educational curricula by describing knowledge points and their interconnections, allowing students to visualize content overviews more clearly. Qu et al. (2024) analyzed 48 publications from 2011 to 2023, revealing a rise in educational knowledge graph research incorporating quantitative evaluation methods. Bi et al. (2024) introduced CodeKGC, showing that code-based language models exhibit superior structured reasoning in knowledge graph construction for technical domains compared to natural language models.

Domain-specific knowledge graph construction presents unique challenges, particularly in technical fields where standard principles may not be directly applicable. Since programming languages contain specialized terminology not found in general dictionaries, domain-specific bilingual dictionaries are essential for both teaching and automatic language processing applications.

2.2 Causal Knowledge Graphs and Query Language Integration

Graph databases such as Neo4j have significantly enhanced the comprehension and analysis of complex datasets composed of interconnected elements. Its native graph architecture enables users to efficiently navigate and explore multi-level relationships. According to Francis et al. (2018), the Cypher query language was introduced to explicitly

express complex graph patterns. In addition, Holzschuher and Peinl (2013) demonstrated that Cypher is more readable and performs better than SQL, especially for multi-hop queries in applications such as programming language genealogy.

Graph-based queries are also applicable to large analytical knowledge graphs. Ferencz et al. (2024) demonstrated the efficiency of Neo4j in analyzing large social or computer network datasets. Dörpinghaus and Stefan (2020) proposed semantic queries on biomedical graphs and showed optimizations through graph techniques and embeddings, achieving performance gains of up to 3,839×. This evidence highlights the effectiveness of graph databases in managing context-rich and structured data for large-scale knowledge discovery.

More recently, efforts have been made to integrate causal inference into property graph models. Pachera et al. (2024) proposed several extensions that enable causal reasoning, including hypernodes, structural equations, and enhancements to query languages such as Cypher and GQL. They introduced a framework for querying causal paths, confounders, and interventions based on do-calculus.

3. Methodology

3.1 System workflow and overview

The workflow begins with identifying a list of C-family programming languages on English Wikipedia and algorithmically generating corresponding Thai Wikipedia URLs through automated translation. BeautifulSoup is employed as the web scraping library to retrieve structured infobox data, targeting the "influenced by" and "influences" relationship fields, and converting the extracted data into a standardized JSON format. The system ensures cross-linguistic consistency through automated deduplication algorithms and the integration of the Google Translate API for Thai language localization. This is followed by systematic import into the Neo4j graph database, where programming languages are represented as nodes with directional influence relationships. Finally, Cypher query-based normalization consolidates duplicate nodes and eliminates naming inconsistencies,

enabling comprehensive analysis of programming language evolution patterns and the exploration of hierarchical influence structures within the programming language domain. The system pipeline consists of six phases: (1) data extraction from Wikipedia infoboxes, (2) data extraction and transformation, (3) cross-linguistic translation verification, (4) Import into Neo4j graph database, (5) Knowledge graph normalization, and (6) Knowledge graph visualization.

3.2 Data extraction from Wikipedia infoboxes

This section presents a methodology for extracting programming language relationship data from Wikipedia infobox across bilingual content (English and Thai). The outcome of this process is a comprehensive list of URLs for C-family programming languages in both English and Thai. The data extraction from Wikipedia infoboxes consists of two sub-processes: (1) bilingual programming language URL generation and (2) Data extraction and transformation.

3.2.1 Bilingual programming language URL generation

The target language corpus was constructed based on a list of C-family programming languages from Wikipedia. The programming language list serves as our data source for relational analysis, providing unified programming language names and URLs. Two patterns were observed in English URLs: they either use the programming language name directly or append the suffix "_(programming_language)." Similarly, two such patterns exist in Thai URLs: they either use the programming language name or add the prefix "ภาษา" (meaning "language") before the name.

To enable bilingual analysis, Thai Wikipedia URLs were generated from their English counterparts using a URL conversion algorithm. The Convert_English_URL_to_Thai_URL algorithm extracts article titles from English URLs, performs English-to-Thai translation, and reconstructs the URLs according to Thai Wikipedia formatting standards. Each input URL is first checked for the presence of the "/wiki/" segment. If found, the remaining segment, representing the article path, is extracted. Underscores are replaced with spaces, and the

resulting text is translated into Thai using the Google Translate API. The translated title is then reformatted according to Wikipedia URL standards to construct the final Thai URL.

This method facilitates bilingual URL mapping and supports multilingual data integration by generating Thai Wikipedia URLs directly from English sources. The pseudocode for this algorithm is shown in Figure 1, and Table 1 presents examples of English and Thai URLs generated by this method.

Table 1 Example of English-to-Thai URL.

Programming language	**English URL**	**Thai URL**
C	https://en.wikipedia.org/wiki/C_(programming_language)	https://th.wikipedia.org/wiki/ภาษาซี
Java	https://en.wikipedia.org/wiki/Java_(programming_language)	https://th.wikipedia.org/wiki/ภาษาจาวา
PHP	https://en.wikipedia.org/wiki/PHP	https://th.wikipedia.org/wiki/พีเอชพี
Python	https://en.wikipedia.org/wiki/Python_(programming_language)	https://th.wikipedia.org/wiki/ภาษาไพทอน

```
Algorithm: Convert_English_URL_to_Thai_URL
Input: A list of English Wikipedia URLs
//Example input: https://en.wikipedia.org/wiki/C_(programming_language
1. Initialize an empty list called Thai_URL_List
2. For each URL in the input list do:
3.    If (URL does not contain "/wiki/"), Then Append "Invalid URL" to Thai_URL_List
4.    Else Continue to the next URL
5.    Extract the article title after "/wiki/"          // Result: "C_(programming_language)"
6.    Replace all underscores (_) with spaces   // Result: "C (programming language)"
7.    Translate the title from English to Thai using a translation service
      // Result: "C (programming language)" → "ภาษาซี"
8.    Construct the Thai URL by concatenating: "https://th.wikipedia.org/wiki/" +
      translated_title
      // Result: https://th.wikipedia.org/wiki/ภาษาซี
9.  Append the constructed Thai URL to Thai_URL_List
10. Return Thai_URL_List
```

Figure 1 Pseudocode of Convert_English_URL_to_Thai_URL.

3.3 Data extraction and transformation

The extraction procedure begins by retrieving programming language data from Wikipedia articles in both Thai and English. The infobox is typically located on the right side of each article page. This research explicitly targets two key relationship types: “influenced by” and “influences.” When accessing each language’s article page, we utilize BeautifulSoup—a Python library designed for parsing HTML web content. The extracted data is temporarily stored in system variables before being converted into JSON format, which provides an optimal structure for storage and subsequent processing.

Using the list of English and Thai URLs generated in the Bilingual Programming Language URL Generation process as input for this stage, we found that 35 English-language articles and 14 Thai-language articles contained infobox fields for “influenced by” and “influences.” Figure 2 illustrates an example of related languages listed in the infobox of the C programming language article on both English and Thai Wikipedia.

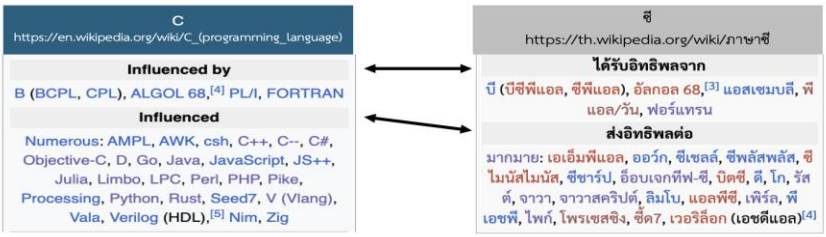


Figure 2 The infobox of C language on English and Thai Wikipedia.

The JSON structure uses the Wikipedia article URL as the root object, labeled <language_article_title>", for each programming language, such as "python_ (programming_language)." Each key contains a nested object with two fields: "influenced" and "influenced by." These fields are arrays listing programming languages that the given language has influenced or been influenced by. This structure is well-suited for graph-based analysis or for constructing knowledge representations of programming language evolution.

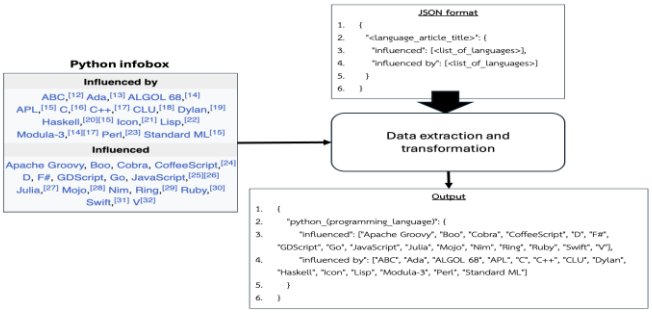


Figure 3 Python language data in JSON format.

An example of Python data saved in a JSON file is shown in Figure 3. According to its JSON structure, Python has been influenced by 14 programming languages such as C, C++, Haskell, and Lisp, and has influenced 16 programming languages such as JavaScript, Go, Swift, and Ruby. This JSON structure offers several advantages, including a query-friendly format for analyzing relationship patterns, direct conversion to graph nodes and edges, and scalability for adding new languages. The standardized format ensures consistency across all programming languages in the dataset and facilitates cross-platform

compatibility, automated processing, and integration with graph visualization tools and relationship analysis algorithms.

3.4 Cross-linguistic translation verification

Data extraction from the Thai Wikipedia infobox fields “influenced” and “influenced by” was the prior step. However, analysis revealed that some programming language names in these fields appeared in both Thai and English, rather than consistently in Thai. For example, the Objective-C article shows both Thai and English names in these fields, as illustrated in Figure 4. To address this, this paper proposes a translation dictionary using the Google Translate API to verify consistency and completeness when creating the Thai-language knowledge graph. By automatically translating English programming language names from Thai Wikipedia into their Thai equivalents, this method facilitates the construction of a knowledge graph and enables efficient visualization in subsequent phases.

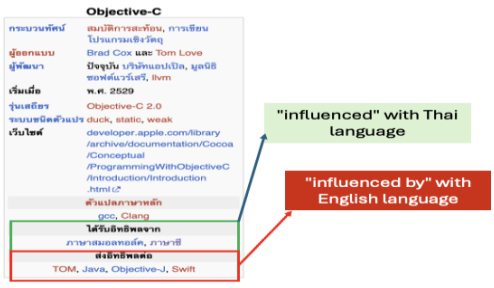


Figure 4 Objective-C infobox with both English and Thai language.

3.5 Import into Neo4j graph database

The structured JSON data is systematically imported into the Neo4j graph database platform, which provides efficient storage and retrieval capabilities for graph-structured data while supporting subsequent analytical operations. The import process organizes the data into two fundamental components: programming language nodes and graph relationships. Programming language nodes serve as the primary entities within the graph

structure. Each programming language is represented as a separate node with attributes such as language names and related programming languages, ensuring that all data is interconnected within the overall graph architecture.

Graph relationships create links between programming language nodes using two distinct relationship types that describe the directionality among programming languages. By establishing backward-looking dependency links between languages influenced by earlier languages, the “influenced by” relationship traces the historical development of programming paradigms. The “influenced” relationship, on the other hand, connects languages that have shaped the evolution of other programming languages, creating forward-looking impact connections that show how foundational languages continue to influence the design and implementation of contemporary programming languages.

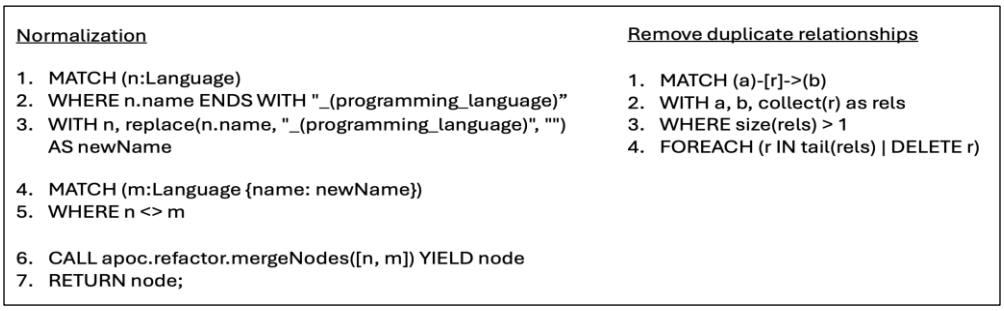
3.6 Knowledge graph normalization

Normalization of programming language names was implemented to remove variations and potential ambiguities in search results caused by inconsistent naming conventions. Many programming languages have unique names that include prefixes or suffixes such as “programming language,” which can lead to redundant node creation and inaccurate search results.

This process removes symbols, punctuation, and suffix phrases from programming language names to minimize such differences. Cypher queries in the Neo4j database are used as part of the normalization procedure to merge duplicate nodes that represent the same programming language but differ in naming conventions. For example, "Python_(programming_language)" is merged with "Python" to ensure consistency within the knowledge graph. The process uses the apoc.refactor.mergeNodes function to correctly merge nodes and prevent data duplication.

After node consolidation, duplicate relationships are eliminated using a complementary Cypher query that identifies multiple relationships between the same pair of nodes, retains one instance, and removes redundant connections. This systematic

approach ensures that the knowledge graph maintains both node uniqueness and relationship integrity throughout the normalization process. The use of Cypher queries for normalization and duplicate relationship removal is illustrated in Figure 5.



Normalization

```
1. MATCH (n:Language)
2. WHERE n.name ENDS WITH "_(programming_language)"
3. WITH n, replace(n.name, "_(programming_language)", "")
   AS newName

4. MATCH (m:Language {name: newName})
5. WHERE n <> m

6. CALL apoc.refactor.mergeNodes([n, m]) YIELD node
7. RETURN node;
```

Remove duplicate relationships

```
1. MATCH (a)-[r]->(b)
2. WITH a, b, collect(r) as rels
3. WHERE size(rels) > 1
4. FOREACH (r IN tail(rels) | DELETE r)
```

Figure 5 Cypher commands for knowledge graph normalization.

3.7 Knowledge graph visualization

After completing the data normalization step, we used the Neo4j graph database to create a graph-based data structure. This process demonstrates the relationships between programming languages as structured, semantically meaningful connections within a knowledge graph context. In particular, we applied graph traversal methods to analyze and visualize these relationships with respect to the C programming language.

The initial analysis focused on direct relationships, providing insight into how languages immediately affect or depend on each other. This process then became more complex by involving multi-level traversals that explore indirect or hierarchical relationships up to three levels deep. Such analysis enables visualization of how programming languages have evolved over time and how they interconnect and interact within the programming landscape. This graph-based approach proves effective in supporting research on programming language evolution.

4. Results and discussion

This section presents the results derived from the methodology section, with a focus on constructing a knowledge graph to represent relationships between programming languages originating from primary language.

4.1 Results from data extraction on Wikipedia

A total of 35 programming languages influenced by the C programming language were identified through data extraction. The analysis revealed multi-level influence chains, with some extending up to three levels (e.g., C ⟶ C++ ⟶ Java ⟶ Kotlin). The data were formatted in JSON and imported into a Neo4j graph database to construct a knowledge graph of these relationships. In total, 35 English-language and 14 Thai-language articles were found to contain infobox fields for “influenced by” and “influenced.”

4.1.1 Programming language in English

Data describing influences of programming languages related to C was obtained via the “influenced by” and “influenced” fields of infoboxes for 35 English Wikipedia articles about C and descendant languages and used to build a hierarchy model of the layered influences of the C programming language on modern programming languages.

Programming languages that were influenced by C were classified into three levels, as shown in Table 2. Level 1 consisted of C-influenced languages, totaling 31 languages. Level 2 consisted of languages influenced by Level 1 languages, totaling 68 languages. Level 3 consisted of the next generation of languages influenced by Level 2 languages, totaling 77 languages. After deducting duplicates across the three levels, a total of 93 unique programming languages were identified as being influenced by C, whether directly or indirectly.

Table 2 Influence Levels of C-family programming languages from English Wikipedia.

Influence Level	Number of Languages	Examples of programming language
Level 1 — Directly influenced by C	31	C++, Objective-C, Perl, Pike, JavaScript, D
Level 2 — Influenced by Level 1	68	Java, C#, Swift, PHP, Rust
Level 3 — Influenced by Level 2	77	Kotlin, TypeScript, Hack, Dart, Nim, CoffeeScript
Total (Unique Languages, Levels 1–3)	**93**	-

4.1.2 Programming language in Thai

The analysis categorized the influenced Thai programming languages into three levels according to their relation to C. The levels of influence are summarized in Table 3. Level 1 comprises languages directly influenced by C in either design or syntax, totaling 21 languages. Level 2 contains languages that were influenced by Level 1 languages, which contain essential elements of C and have developed them further. The total number of Level 2 languages was 41. Level 3 contains languages directly influenced by or derived from Level 2 languages. A total of 37 languages fell under Level 3. After removing all duplicates that occurred across levels, 60 technologies constituted programming languages in Thai that were influenced directly or indirectly by C.

Table 3 Influence Levels of C-family programming languages from Thai Wikipedia.

Influence Level	Number of Languages	Examples of programming language
Level 1 — Directly influenced by C	21 languages	ซีพลัสพลัส (C++), อ็อบเจกทีฟ-ซี (Objective-C), เพิร์ล (Perl), ไพก์ (Pike), จาวาสคริปต์ (JavaScript), ดี (D)
Level 2 — Influenced by Level 1	41 languages	จาวา (Java), ซีชาร์ป (C#), พีเอชพี (PHP), รัสต์ (Rust), ลูอา (Lua)
Level 3 — Influenced by Level 2	37 languages	โคทลิน (Kotlin), ไทป์สคริปต์ (TypeScript), แฮ็ก (Hack), ดาร์ต (Dart), คอฟฟี่สคริปต์ (CoffeeScript)
Total (Unique Languages, Levels 1–3)	60 languages (non-duplicated)	-

4.2 Results from knowledge graph implementation in Neo4j

The programming language knowledge graph in Neo4j provides a bilingual database of English and Thai resources. The programming languages were stored as nodes in the graph and then connected using two directed relationship types: “influenced by” and “influenced,” forming a network showing the development and influence of programming languages. The graph was navigated and queried using the Cypher query language. Figure 6 displays the first-level influence graph of the C programming language based on English Wikipedia data, while Figure 7 is based on Thai Wikipedia. An example of the second-level influence graph of C from Thai Wikipedia is shown in Figure 8.

The experiment in constructing a knowledge graph to illustrate the relationships among programming languages influenced by C-family programming languages was conducted successfully. The results demonstrated that the Neo4j graph database is

capable of efficiently processing and retrieving data. This confirms its suitability for representing complex relational structures such as programming language influence networks. Utilizing the Cypher query language enables fast and accurate data access, allowing comprehensive representation of language relationships across all nodes at various levels.

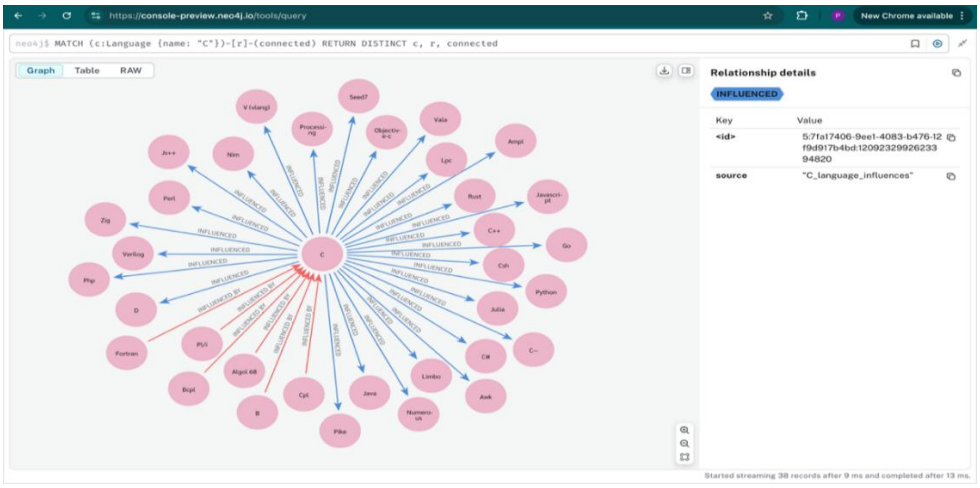


Figure 6 First-level Influence Graph of the C Programming Language from English Wikipedia.

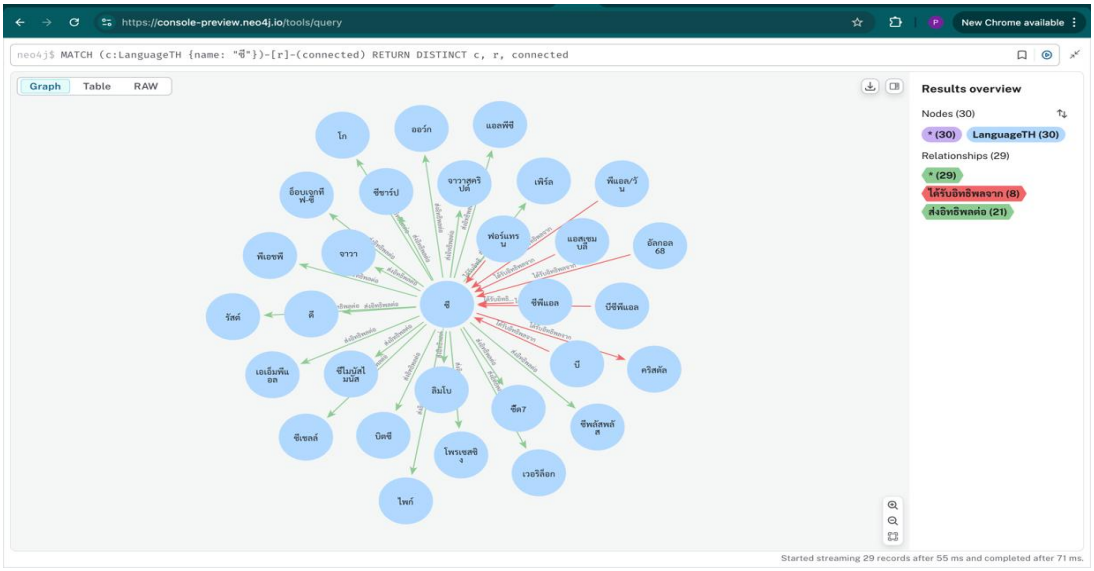


Figure 7 First-level Influence Graph of the C Programming Language from Thai Wikipedia

This experiment also revealed the occurrence of redundant relationships between nodes, with some programming languages appearing as both “influenced by” and “influenced” within the same node. Consequently, the graph exhibited some circular relationships, which contributed to its increased complexity. To avoid this in the future, the system was developed to ensure that relationships are stored in one direction only, reducing structural complexity and improving clarity in presenting data for the analysis of programming language evolution.

4.3 Evaluation of the Knowledge Graph with Quantitative Analysis

The research aimed to evaluate the effectiveness of graph designs representing relationships among programming languages by analyzing key structural components—node connectivity, edge layout, and data layering—and their impact on user comprehension. A total of 50 participants with varying levels of IT experience (none, basic, intermediate, or experienced) completed a questionnaire, rating five different graph

formats using a five-point Likert scale (5 = strongly agree, 1 = strongly disagree) in response to the same evaluation question.

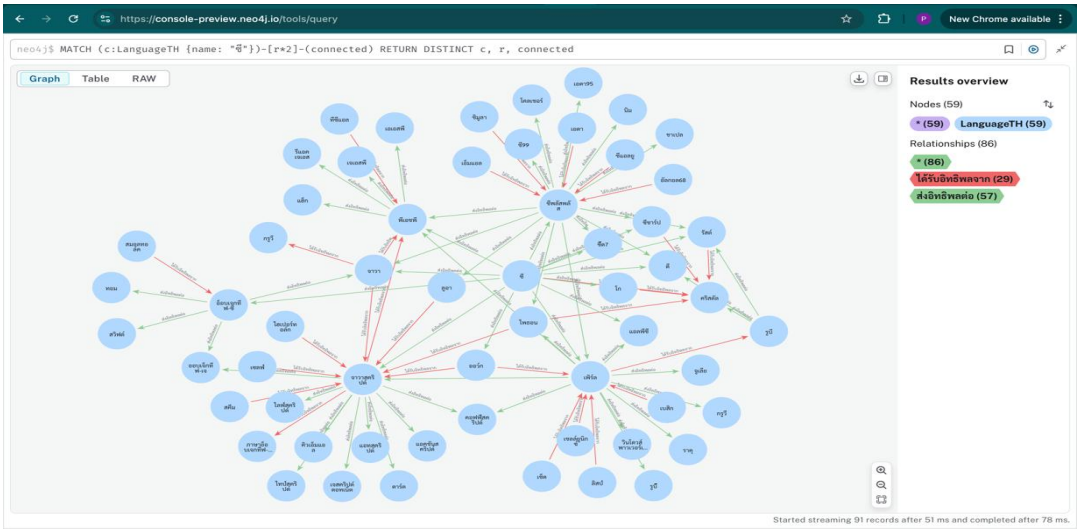


Figure 8 Second-level Influence Graph of the C Programming Language from Thai Wikipedia.

The quantitative analysis of the responses involved calculations of mean and standard deviation to evaluate trends and the consistency of responses, as shown in Table 4. The graph created using the methodology proposed by this study had the highest average score (4.150) and the lowest standard deviation (1.137), and was therefore rated as having good clarity and interpretive consistency. The graph appears to be effective due to its strong organizational structure and hierarchy of nodes, the standardized directional links, and the use of distinct colors to differentiate the dataset.

Table 4 Mean scores and standard deviations of user ratings.

Graph	Mean Score	Standard Deviation (SD)
1	2.650	1.309
2	**4.150**	**1.137**
3	3.100	1.447
4	2.400	1.231
5	2.950	1.191

Conversely, the lowest-rated graph had a cluttered layout, with overlapping links and no clear hierarchy, which hindered comprehension. Overall, the analysis revealed that simple, organized, and logically grouped designs significantly improved the communication of complex relationships. These results support Graph Comprehension Theory, which highlights the importance of visual simplicity, clear navigation paths, and minimizing link complexity to enhance interpretability.

5. Conclusions

This paper proposes the development of a knowledge graph designed using a Neo4j graph database to represent and study influence relationships among programming languages that originated from C. The process involved the systematic extraction of data from Wikipedia, transformation of the data into a structured JSON format, and exploration as a graph. The resulting system, built using the Neo4j knowledge graph, showed improved searchability of connected information. The Neo4j knowledge graph was more effective than a typical manual approach in terms of search accuracy and response time.

By using the Cypher query language, users of the graph were able to query multi-level inheritance and influence relationships that provided relevant context to help connect the influence relationships to the development of programming languages. The architecture of the system is modular and extensible, allowing for future integration of data from additional sources without changing the structure of the original knowledge

graph. This method is not limited to programming language impact and could be employed in other applications involving large datasets used for modeling complex relational information. The results support the viability of graph models for effectively representing and interacting with interconnected information as a research tool in computing.

Future work should focus on expanding the knowledge graph with data beyond Wikipedia—such as GitHub repositories, academic papers, and official documentation—to better illustrate connections between programming languages. Deep learning techniques could support predictive modeling of future language influences and automate relationship extraction from text. The modular design also allows for extension to frameworks, libraries, development tools, and educational sources (e.g., online courses, tutorials, code snippets). Finally, the approach should be validated in another domain (e.g., scientific research networks or tech innovation ecosystems) to demonstrate cross-domain applicability.

References

Bi, Z., Chen, J., Jiang, Y., Xiong, F., Guo, W., Chen, H., & Zhang, N. (2024). **CodeKGC: Code language model for generative knowledge graph construction**. *ACM Transactions on Asian and Low-Resource Language Information Processing, 23*(3), Article 1–16.

Chen, X., Jia, S., & Xiang, Y. (2020). **A review: Knowledge reasoning over knowledge graph**. *Expert Systems with Applications, 141*, 112948.

Cheng, Y. (2022, December). **A learning path recommendation method for knowledge graph of professional courses**. In *2022 IEEE 22nd International Conference on Software Quality, Reliability, and Security Companion (QRS-C)* (pp. 469-476).

Dörpinghaus, J., & Stefan, A. (2020). **Semantic graph queries on linked data in knowledge graphs**. In *The Workshop on Computational Optimization* (pp. 81–102). Springer International Publishing.

Ferencz, K., Rigó, E., Domokos, J., & Kovács, L. (2024). ***Implementation of network data analysis using the Neo4j graph database***. *Acta Universitatis Sapientiae, Electrical and Mechanical Engineering, 16*(2024), 162–176.

Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., & Taylor, A. (2018). **Cypher: An evolving query language for property graphs**. In *Proceedings of the 2018 International Conference on Management of Data (SIGMOD '18)* (pp. 1433–1445). Association for Computing Machinery. https://doi.org/10.1145/3183713.3190657

Holzschuher, F., & Peinl, R. (2013). **Performance of graph query languages: Comparison of Cypher, Gremlin and native access in Neo4j**. In *Proceedings of the Joint EDBT/ICDT 2013 Workshops* (pp. 195–204). ACM.

Ji, S., Pan, S., Cambria, E., Marttinen, P., & Yu, P. S. (2021). **A survey on knowledge graphs: Representation, acquisition, and applications**. *IEEE transactions on neural networks and learning systems*, *33*(2), 494-514.

Meyerovich, L. A., & Rabkin, A. S. (2013, October). **Empirical analysis of programming language adoption**. In *Proceedings of the 2013 ACM SIGPLAN international conference on Object oriented programming systems languages & applications* (pp. 1-18)

Pachera, A., Palmiotto, M., Bonifati, A., & Mauri, A. (2024). **What if: Causal analysis with graph databases**. *arXiv preprint* arXiv:2412.13965. https://arxiv.org/abs/2412.13965

Qu, K., Li, K. C., Wong, B. T. M., Wu, M. M. F., & Liu, M. (2024). **A survey of knowledge graph approaches and applications in education.** *Electronics, 13*(13), 2537. https://doi.org/10.3390/electronics13132537

Valverde, S., & Solé, R. V. (2015). **Punctuated equilibrium in the large-scale evolution of programming languages**. *Journal of the Royal Society Interface, 12*(107), 20150249. https://doi.org/10.1098/rsif.2015.0249

Wang, Q., Mao, Z., Wang, B., & Guo, L. (2017). **Knowledge graph embedding: A survey of approaches and applications**. *IEEE transactions on knowledge and data engineering*, *29*(12), 2724-2743.