Application of Graph Theory in the Analysis of Query Similarity and Complexity in Relational Databases

Beatriz Soares de Souza
Department of Computer Engineering and Automation
UFRN, Natal-RN, Brazil
bsouza@ufrn.edu.br

Luiz Affonso Guedes
Department of Computer Engineering and Automation
UFRN, Natal-RN, Brazil
affonso@dca.ufrn.br

***Abstract*—Information Systems constant changes and refactoring processes eventually result in design debts, one of them being related to the database management. Redundancy and complexity are often found in databases and eventually affect the overall system performance. In this study, the analysis conducted was based on graph analysis, which is an essential technique in several fields demanding data management. The investigation involved the study of relationships and connections in a real-world financial organization database, between SQL queries represented as nodes and edges in a graph structure. By analyzing the graph structure and properties, it was possible to identify important nodes, detect clusters of related data, and uncover hidden relationships and redundancy. The results indicate that 50% of the database queries had medium to high similarity in subgraphs, which allows the organization to gain valuable insights into their data, make informed decisions, and optimize their database performance.**

***Index Terms*—Graph Analysis, Isomorphism, Information Systems, Relational Databases**

I. Introduction

Database Management Systems (DBMS) are intricate programs that allow users to input queries, translate those queries into the correct format to access data, and produce useful results in the shortest amount of time while using the fewest amount of resources [1].

Information Systems (ISs) are constantly changing to meet the demands of consumers, the environment, and new demands for higher-quality information delivery and services. These systems' ability to operate depend on databases, and due to the need to evolve, integrate to new services or shift to new technologies, most systems prefer to extend the underlying design of existing legacy databases. Refactoring is therefore frequently used to evolve databases and their schemas in order to satisfy new requirements or properties.

During the adaptation and refactoring processes, system developers and database designers must adhere to best practices for database architecture. However, this process can be compromised by poor design choices that can result in what we refer to as database design debts [2].

Although databases should be simple to use and maintain, if the architecture and queries are complex or redundant, the efficiency of DBMS software may suffer. Therefore, redundancy and complexity place an additional burden on database management system software, which eventually affects its overall performance by increasing processing time and resource costs [3].

Currently, there are various ways to analyze databases in order to detect redundancies, such as probabilistic matching models, supervised and semi-supervised Learning [4], property entropy grouping clustering (EGC), and finally, the development of frameworks to detect patterns in dirty data based on the obtained relationships by establishing the relationship graph in the data.

Analysis using graph theory can be described as the analysis of existing relationships between the different elements contained in a given data set. The term vertex is used to describe the elements, while the term edge is used to refer to the connections between the different vertices of a network [5].

Thus, the main objective of this research is to investigate the feasibility of using graph theory to analyze redundancy and complexity between queries that are processed in a common relational database system.

This paper is structured as follows. In Section II, the problem characterization and related works are presented. In Section III, we describe the research methods. In Section IV, we present the results. Finally, in Section V, we present the conclusions.

II. Problem Characterization and Related Works

Consider N independent services inside an organization as shown in Figure 1, consuming data from the main dataset, which means, a common source. As the number of services grows, bigger is the chance of having duplicated queries or queries that are a subset of each other. The main goal of this study is to determine whether there are any redundant queries between these services, which indicates query overhead and could be eliminated by combining them into a single query.

Based on this scenario, this work aims to develop an analysis algorithm capable of detecting similarities between relational database queries through graph analysis, and estimating the reduction of queries when eliminating the duplicates.

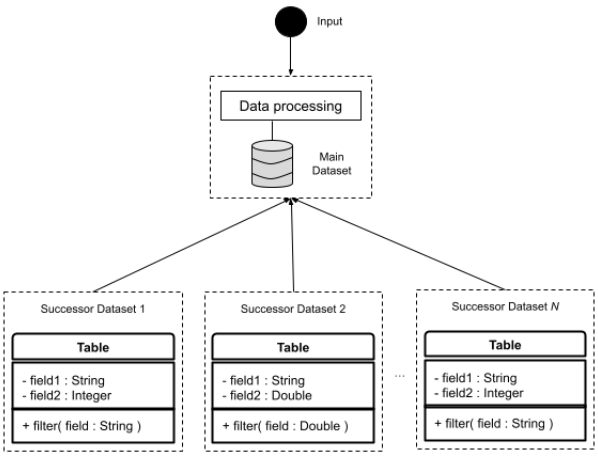

Fig. 1. Diagram of independent datasets consuming from common source.

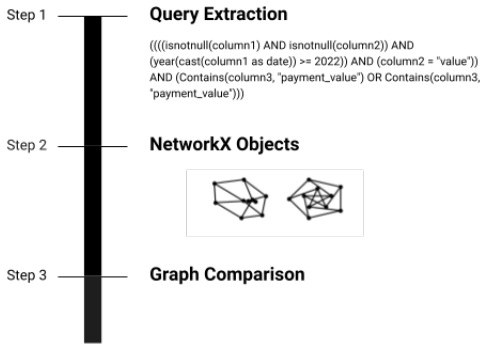

Fig. 2. Stages of proposed approach.

Among its various applications, graphs can represent biological networks at the molecular, protein, or species level. [6] and [7] used graph analysis to find all matches of a pattern graph in biochemical data. [8] proposed the application of graphs for industrial solutions, and there are several graph processing frameworks that have been proposed by both academia and industry for data analysis such as [9], [10] and [11].

To perform this study, the NetworkX Python library [12] was used to compare the graphs, and 102 independent databases from a digital financial service platform were analyzed.

Finally, a tool was developed to calculate the main properties of each graph, such as eccentricity and radius. In this way, it is possible to send insights about potential duplicates and most used dataset fields and columns.

III. Methods

It is typical for distinct services to make similar requests when operating on the same data source as numerous independent services. This query redundancy may lead to inefficient resource usage, which would affect system performance as a whole. Therefore, it is crucial to find solutions that reduce query duplication and maximize the usage of resources.

In order to point out those similarities, the approach chosen in this work was to create a graph representation of each successor dataset, and use the NetworkX Python framework to compare those graphs, which was done in three main steps, as ilustrated in Figure 2.

At the first step, the queries of each independent dataset were computed through the system logs. After that, extraction of the code was performed and a manual annotation of the methods and Structured Query Language (SQL) expressions used: WHERE, WHEREIN, NOT NULL.

In the second stage, the different queries were converted into NetworkX objects, and the graphs created were validated. Each consulted column of the main dataset in this case was represented as a node (vertex) in the graphs, and the queries made are represented by the directed connections that connect these nodes. In this manner, the interactions between the services and their requests are reflected in the directed graph that is produced.

During the third stage, the obtained graphs were used to do the comparison, which lead to the detection of several redundancies and duplicate datasets.

A. Query Extraction

It is possible to interpret a query plan of a dataset as a graphical representation that resembles a graph structure [13]. In this representation, the columns involved in the query are represented as nodes, while the conditions that filter these columns are depicted as edges connecting the corresponding nodes. Additionally, the query plan incorporates properties such as the joining columns and expressions.

Furthermore, a query plan can also be visualized as a tree-like structure, with a hierarchical arrangement of nodes. In this tree structure, the root node represents the primary table from which columns are projected into the final result table. This projection involves selecting specific columns or expressions and populating the result table accordingly.

In this step, the query plans from each individual dataset were extracted and converted into graphs, considering the parameters mentioned above, and ilustrated in Figure 3. The expressions extracted from the organization's system consisted in a row called Input, which contains the list of columns that are queried in the respective dataset and that will be used as the first nodes. And the second row is called Condition, which contains the filters applied in the query.

For this study, all database query extractions have been anonymized. This includes the anonymization of column names, filtered values, and dataset names. As it is a financial organization, which strictly adheres to legal regulations regarding the protection of sensitive information, all data has been appropriately anonymized and treated with utmost confidentiality for the purpose of this research.

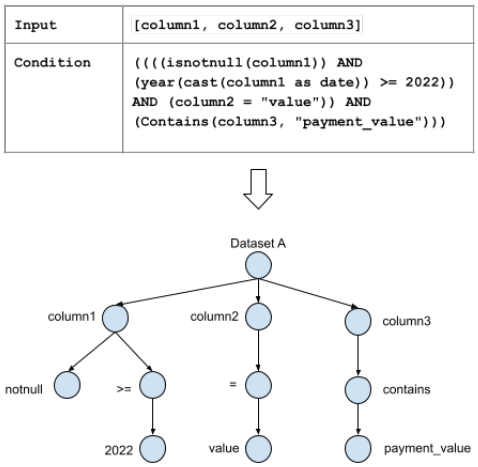

Fig. 3. Example of query extraction done in step 1.

B. NetworkX Objects

Firstly, some widely known graph analysis tools were analyzed for comparison, such as Gephi, Pajek, Networkx, and IGraph. All four software options are freely available and capable of handling large graph sizes. Graph and network analysis tools can be either GUI-based or packages/libraries that can be used within a programming language. Gephi and Pajek are GUI-based network tools, while Networkx and IGraph are package-based tools.

Gephi is an interactive visualization and exploration platform for all types of networks, including dynamic and hierarchical graphs. It is designed for users who need to explore and understand graphs. In the meanwhile, Pajek is a widely used software for drawing networks, and it also offers analytical capabilities. It can compute various centrality measures, identify structural holes, perform block modeling, and more [14].

IGraph is a free software package that provides tools for creating and manipulating graphs. It includes implementations for classic graph theory problems, such as minimum spanning trees and network flow, as well as algorithms for community structure search [15]. It is known for its efficient implementation, allowing it to handle graphs with millions of nodes and edges and can be installed on several different programming languages.

NetworkX is a Python package designed for the exploration and analysis of networks and network algorithms [12]. It offers a wide range of data structures to represent various types of networks, also known as graphs. These graphs can include simple connections, directed relationships, and even complex structures with parallel edges and self loops. In NetworkX, the nodes within these graphs can be represented by any hashable Python object, and the edges have the capability to store diverse data. This level of flexibility makes NetworkX highly adaptable for modeling networks across different scientific disciplines.

The four tools were compared by [16] based on criteria such as platform compatibility, supported graph types, algorithm time complexity, and the NetworkX was proven to offer a good performance when it comes to graph analysis.

Moreover, NetworkX goes beyond providing just basic data structures and hence was chosen for this work. It incorporates numerous graph algorithms that enable the calculation of various network properties and structural measures. These algorithms cover a broad spectrum, including the computation of shortest paths, clustering coefficients, degree distribution, and many more. Leveraging the user-friendly nature and adaptability of the Python programming language, NetworkX emerges as a robust tool for scientific computations and network analysis, and hence was chosen for this work.

C. Graph Comparison

A graph can be represented as G = (V, E) where V and E denote the vertex and edge set [17]. A node-induced subgraph is defined as $Gs = (Vs, Es)$ where Es preserves all the edges between nodes in Vs, i.e. $\forall i, j \in Vs, (i, j) \in Es$ if and only if $(i, j) \in E$.

The problem of subgraph isomorphism is a fundamental concept in graph theory and computational mathematics. It revolves around the question of determining whether a given subgraph can be found within a larger graph, preserving both the structure and the relationships between nodes and edges.

In this work, we aim to detect the Maximum Common induced Subgraph (MCS) by comparing the dataset graphs in pairs, denoted as $MCS(G1, G2)$, in order to find the largest node-induced subgraph contained in both G1 and G2 and identify if one of the graphs can be discarded from the system queries as shown in Figure 4. Graph isomorphism and subgraph isomorphism can be regarded as two special tasks of MCS: $|MCS(G1, G2)| = |V1| = |V2|$ if $G1$ are isomorphic to $G2$, $|MCS(G1, G2)| = min(|V1|, |V2|)$ when $G1$ (or $G2$) is subgraph isomorphic to $G2$ (or $G1$).

However, solving the subgraph isomorphism problem is known to be computationally challenging. It belongs to the class of NP-complete problems, which implies that there is no known efficient algorithm to solve it in polynomial time for all cases. Therefore, researchers have developed various heuristics and approximation algorithms to tackle this problem efficiently for practical scenarios.

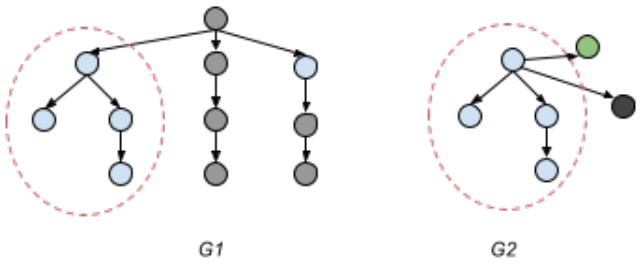

Fig. 4. The highlighted red circle represents the induced connected Maximum Common Subgraph (MCS) for the graph pair (G1, G2). It indicates a probable redundancy between two dataset queries.

The GraphMatcher and DiGraphMatcher are components from the NetworkX framework that provided the role of performing graph matching operations on the directed graphs that were created on the previous stages of the study. Essentially, this entails verifying the presence of an isomorphism between the graphs, although alternative checks could be employed as well. For the porpose of this work, it is useful for detecting whether a subgraph within one graph exhibits isomorphism with a second graph.

The matching process primarily relies on assessing syntactic feasibility, but it also offers the option to evaluate semantic feasibility. In this context, feasibility is determined by the logical conjunction of these two functions, ensuring that both syntactic and semantic criteria are satisfied simultaneously [18].

Using the isomorphic subgraph detection algorithms, we identified these subgraphs within the analyzed datasets. For the purpose of this work and to propose different approaches depending on the similarity, three similarity categories were established based on the percentage of similarity between the subgraphs:

- High Similarity: Subgraphs with a similarity percentage between 70% and 100% were considered highly similar. That means that subgraphs share a large part of their structure and can represent significant patterns or redundancies in the data, and potentially mean that some queries could be droppped from the organization's services.
- Medium Similarity: Subgraphs with a similarity percentage between 50% and 70% were considered moderately similar. These subgraphs have a partial overlap in their structure, indicating partial connections or partially shared patterns, and some indexing or clustering technique could be used in order to improve performance.
- Low Similarity: Subgraphs with a similarity percentage below 50% were considered low similar. These subgraphs have a distinct structure and do not show significant overlap, suggesting that they are already in their independent and performatic form.

IV. Results

The results of the study described below are the outcome of the research conducted using real-world data from a financial organization. With a complex infrastructure which contains 102 independent services, all consuming data from a single shared relational dataset, the analysis focused on examining the behavior of distinct queries from independent services across the system. In total, 102 unique queries were analyzed, they collectively apply 712 filters to the primary dataset and filter 19 different columns, which represent over 100 TB of data in storage available for read-only operations.

A crucial initial step was successfully achieved to ensure the accuracy and reliability of the proposed method: the correct reproduction of each SQL query plan from the relational datasets into directed graphs. This achievement is an important step towards the definition of a reliable graph comparison tool specifically designed for relational databases with independent services. In particular, the ability to efficiently convert queries into graphs and continually update the catalog of graphs, is vital for maintaining the tool's relevance and effectiveness in the face of the ever-growing number of services within the organization.

Upon analyzing the results of the research such as the highlighted graph in Figure 5, the patterns and insights found were remarkable. The subgraph isomorphism algorithm was proven to be a powerful strategy for detecting redundancy within the studied datasets, consistently highlighting the largest instances of query duplication. Through a comprehensive analysis of the generated graphs using isomorphism algorithms, we were able to identify 17 subsets within the organization's database that shared exact characteristics, representing a proportion of 16% of the total queries.

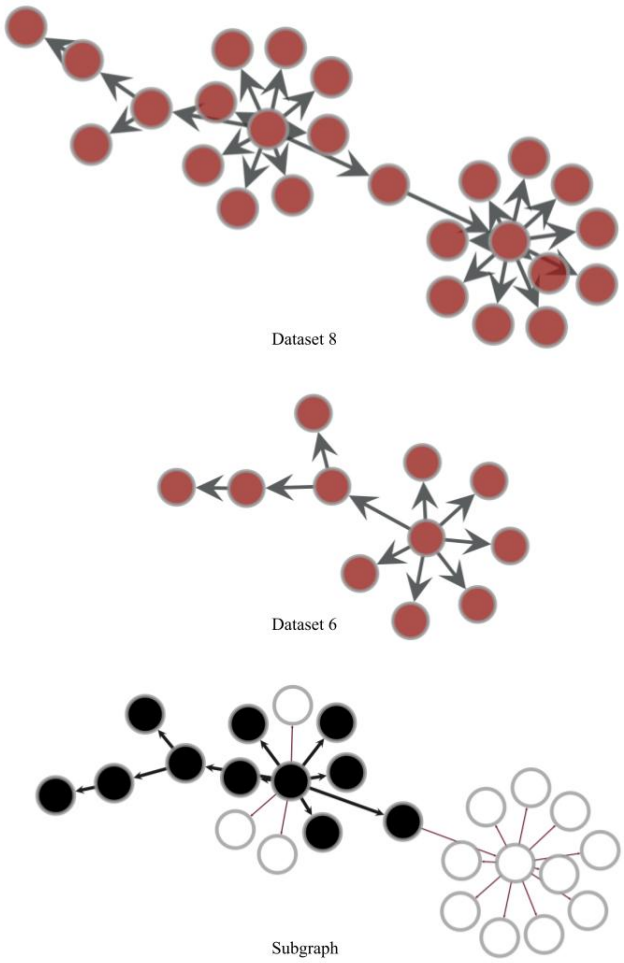

Fig. 5. Highlighted subgraph detected between two datasets.

The insights obtained through the application of the isomorphism algorithm have revealed not only redundant query patterns but also valuable opportunities for optimization and process enhancement. Although some datasets could not be identified as complete subgraphs of each other, there were

queries with a considerably high amount of duplicate nodes and vertexes, that can be studied in order to prevent redundancy.

We obtained high levels of similarity when the program was executed on datasets that belong to the same category of the financial service (24%), and medium similarity in some other queries (10%). When this method was used on more diverse data sets, the similarity levels dropped (49%), and were noticeably different and therefore there was a smaller chance of optimization. As shown in Table I, the total of datasets with considerable similarity was 51%, and at least 26% have a high similarity.

Moreover, the identification of these subsets through graph analysis not only aids in the detection of redundant queries but also opens up opportunities for optimizing the organization's data management processes. By identifying common patterns and recurring query structures, we can streamline database design, enhance indexing strategies, and improve overall query execution times.

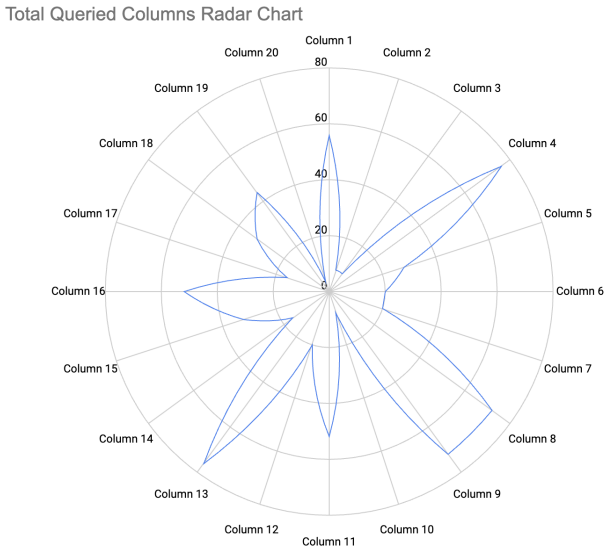

Fig. 6. Radar chart of number of queries for each column in the main dataset.

Indexing is a technique widely used in relational databases to speed up information access and retrieval. It allows the creation of special data structures that make it quicker to find and sort records based on specific values [19]. However, indexing all columns in a table can result in an unnecessary increase in index size and negatively impact the performance of insert, update, and delete operations. Therefore, it is essential to accurately identify the columns that really require indexing. Based on the results obtained from the graph analysis, there are two key points [20] that can be taken into consideration for identifying columns for indexing:

- Query frequency: It is important to analyze how often a column is used as a search criterion in frequent queries. Columns that are frequently used in WHERE clauses or JOINs can benefit from indexing to improve data retrieval speed.
- Query type: The type of query performed against the database also plays an important role in the indexing decision. Queries that involve table sorting, grouping, or joining operations can benefit from proper indexing of the relevant columns.

The radar chart presented in Figure 6 offers a visual representation of the frequency that each column was queried from the common source database and the corresponding number of connections each node possesses within the dataset's graph structure. This visualization allows us to simultaneously assess two essential aspects of the database: the popularity of specific columns in query operations and the level of interconnectedness among the nodes.

The radar chart's outermost layer depicts the frequency of column queries. Each spoke in the chart represents a specific column within the dataset. The length of each spoke is proportional to the number of times a particular column has been queried. By observing the chart, we can identify columns that have been frequently accessed, indicating their significance in the dataset's analytical processes.

By analyzing the radar chart, we can derive valuable insights into the dataset's utilization and structural characteristics. Here are some key observations that can be made from the hypothetical chart:

Columns with longer spokes in the radar chart indicate a higher frequency of queries. These columns play a crucial role in data analysis and decision-making processes, as they are frequently accessed for extracting valuable information and therefore could be the most impactful candidates for indexing.

By examining the chart, we can identify at least four columns that exhibit similar query frequencies: Columns 4, 8, 9 and 13. This may indicate potential optimization opportunities by partitioning query operations and/or indexing these columns in the database.

By exploring the structural similarity of the discovered subsets, it is possible to infer optimization points in the data processing services, which consequently should result in a significant reduction of processing costs. In the context of future work, there are some solutions to explore, considering the results found in this study, specifically focusing on addressing the issue of query redundancy and measuring the impact of the improvements. Conducting comprehensive experiments and evaluations on the datasets can provide valuable insights into the potential benefits of query deduplication such as lowering query execution time, resource consumption, and overall system efficiency.

TABLE I
SUBGRAPH SIMILARITY ANALYSIS RESULTS

Isomorphic	***Similarity***	***Number of datasets***	***% of total***
Yes	High	17	16
No	High	11	10
No	Medium	25	24
No	Low	50	49

V. Conclusion

The results presented show promising results, a high number of redundancy that could be eliminated was detected by using the NetworkX Python Library. The computational development presented in this work showed how it is possible to have proper knowledge of a system's database and infer improvements based on graph analysis.

Finally, the analysis of the generated graphs using isomorphism algorithms provides a valuable basis for future research and exploration. By understanding the similarities among queries, we can develop more efficient query planning and optimization techniques directed to the organization's specific needs. This, in turn, contributes to enhanced decision-making capabilities, as well as improved system scalability and adaptability. Moving forward, further exploration and refinement of these findings will enable enhanced database performance, improved data management practices, and more efficient utilization of resources within the organization.

Acknowledgment

This work was supported by the Department of Computer Engineering and Automation at the Federal University of Rio Grande do Norte (DCA-UFRN).

References

[1] T. M. Connolly and C. E. Begg, *Database systems: a practical approach to design, implementation, and management*. Pearson Education, 2005.

[2] M. Al-Barak and R. Bahsoon, "Database design debts through examining schema evolution," in *2016 IEEE 8th International Workshop on Managing Technical Debt (MTD)*. IEEE, 2016, pp. 17–23.

[3] M. S. Vighio, T. J. Khanzada, and M. Kumar, "Analysis of the effects of redundancy on the performance of relational database systems," in *2017 IEEE 3rd International Conference on Engineering Technologies and Social Sciences (ICETSS)*. IEEE, 2017, pp. 1–5.

[4] A. K. Elmagarmid, P. G. Ipeirotis, and V. S. Verykios, "Duplicate record detection: A survey," *IEEE Transactions on knowledge and data engineering*, vol. 19, no. 1, pp. 1–16, 2006.

[5] A. Ruiz-Frau, A. Ospina-Alvarez, S. Villasante, P. Pita, I. Maya-Jariego, and S. de Juan, "Using graph theory and social media data to assess cultural ecosystem services in coastal areas: Method development and application," *Ecosystem Services*, vol. 45, p. 101176, 2020.

[6] V. Bonnici, R. Giugno, A. Pulvirenti, D. Shasha, and A. Ferro, "A subgraph isomorphism algorithm and its application to biochemical data," *BMC bioinformatics*, vol. 14, no. 7, pp. 1–13, 2013.

[7] D. Fourches and A. Tropsha, "Using graph indices for the analysis and comparison of chemical datasets," *Molecular Informatics*, vol. 32, no. 9-10, pp. 827–842, 2013.

[8] L. Nai, Y. Xia, I. G. Tanase, H. Kim, and C.-Y. Lin, "Graphbig: understanding graph computing in the context of industrial solutions," in *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, 2015, pp. 1–12.

[9] I. Tanase, Y. Xia, L. Nai, Y. Liu, W. Tan, J. Crawford, and C.-Y. Lin, "A highly efficient runtime and graph library for large scale graph analytics," in *Proceedings of Workshop on GRAph Data management Experiences and Systems*, 2014, pp. 1–6.

[10] Y. Low, J. Gonzalez, A. Kyrola, D. Bickson, C. Guestrin, J. M. Hellerstein, and D. GraphLab, "A framework for machine learning and data mining in the cloud," *Proceedings of the VLDB Endowment*, vol. 5, no. 8, 2012.

[11] Y. Song, D. Zhang, X. Li, K. Luo, and J. Liao, "A novel data cleaning framework based on knowledge graph," in *2022 8th International Conference on Big Data Computing and Communications (BigCom)*, 2022, pp. 350–355.

[12] A. Hagberg, P. Swart, and D. S Chult, "Exploring network structure, dynamics, and function using networkx," Los Alamos National Lab.(LANL), Los Alamos, NM (United States), Tech. Rep., 2008.

[13] T. Taipalus, "A notation for planning sql queries," *Journal of Information Systems Education*, vol. 30, no. 3, pp. 160–166, 2019.

[14] G. A. Pavlopoulos, D. Paez-Espino, N. C. Kyrpides, and I. Iliopoulos, "Empirical comparison of visualization tools for larger-scale network analysis," *Advances in bioinformatics*, vol. 2017, 2017.

[15] N. Akhtar and M. V. Ahamad, "Graph tools for social network analysis," in *Research Anthology on Digital Transformation, Organizational Change, and the Impact of Remote Work*. IGI Global, 2021, pp. 485–500.

[16] N. Akhtar, "Social network analysis tools," in *2014 fourth international conference on communication systems and network technologies*. IEEE, 2014, pp. 388–392.

[17] Y. Bai, D. Xu, Y. Sun, and W. Wang, "Glsearch: Maximum common subgraph detection via learning to search," in *International Conference on Machine Learning*. PMLR, 2021, pp. 588–598.

[18] Q. Zhang, R.-H. Li, H. Qin, G. Wang, Z. Zhang, and Y. Yuan, "Stable subgraph isomorphism search in temporal networks," *IEEE Transactions on Knowledge and Data Engineering*, vol. 35, no. 6, pp. 6405–6420, 2023.

[19] S. Chaudhuri and V. R. Narasayya, "An efficient, cost-driven index selection tool for microsoft sql server," in *VLDB*, vol. 97. San Francisco, 1997, pp. 146–155.

[20] S. Das, M. Grbic, I. Ilic, I. Jovandic, A. Jovanovic, V. R. Narasayya, M. Radulovic, M. Stikic, G. Xu, and S. Chaudhuri, "Automatically indexing millions of databases in microsoft azure sql database," in *Proceedings of the 2019 International Conference on Management of Data*, 2019, pp. 666–679.