Scientific Research Communications

Research Article

Sci. Res. Comm. (2024)
Volume 4, Issue 1
doi: 10.52460/src.2024.005

An Efficient Index-Based Algorithm for Exact Subgraph Isomorphism on Bipartite Graphs

Mehmet Burak Koca[1,*], Fatih Erdoğan Sevilgen[2]

[1] Department of Computer Engineering, Gebze Technical University, Kocaeli, Türkiye
[2] Institute for Data Science and Artificial Intelligence, Boğaziçi University, İstanbul, Türkiye
[*] Corresponding author: b.koca@gtu.edu.tr

Received: 16.12.2023 Accepted: 13.01.2024

Abstract

Graphs are widely used to represent various real-world networks, but their non-linear nature and size increase pose challenges for efficient analysis. The subgraph isomorphism problem, which involves identifying subgraphs that are isomorphic to a query graph, plays a crucial role in diverse domains. In this paper, we focus on the exact subgraph isomorphism problem in bipartite graphs and propose a novel index-based solution algorithm. Our algorithm leverages triplet structures for graph embedding and uses a multi-level hash map for efficient filtering. We also introduce an optimized solution building process. Experimental results on real-world datasets demonstrate the performance superiority of our algorithm compared to state-of-the-art algorithms, with 2 to 500 times shorter querying times. Our findings suggest that our algorithm is a powerful and efficient solution for exact subgraph isomorphism in bipartite graphs.

Keywords: Subgraph isomorphism; bipartite graphs; index-based

1. Introduction

Graphs serve as essential data structures for modeling various real-world networks such as roadmaps, social networks, protein interactions, and chemical compound structures. The examination of these graph structures plays a pivotal role in understanding the inherent relationships and substructures they encapsulate. Nevertheless, conducting an efficient analysis of these graphs presents formidable challenges, particularly given their non-linear topology and the ongoing growth in dataset sizes.

The subgraph isomorphism (SI) problem is a fundamental technique in graph analysis, widely used in various scientific fields. It involves identifying subgraphs within a target graph that match a specified query graph (Corneil and Gotlieb, 1970). This challenge is common in diverse domains such as protein interaction networks and communication networks, prompting significant efforts to accurately detect these isomorphic structures. However, the SI problem is known for its high complexity, making it challenging to find practical solutions.

Subgraph isomorphism is an NP-complete problem because it is an extension of the maximal clique issue and the challenge of determining whether a graph has a Hamiltonian cycle (Cook, 2023). A Hamiltonian cycle is a cycle that visits each vertex exactly once. In the context of the subgraph isomorphism problem, we are looking for a specific pattern of vertices and edges

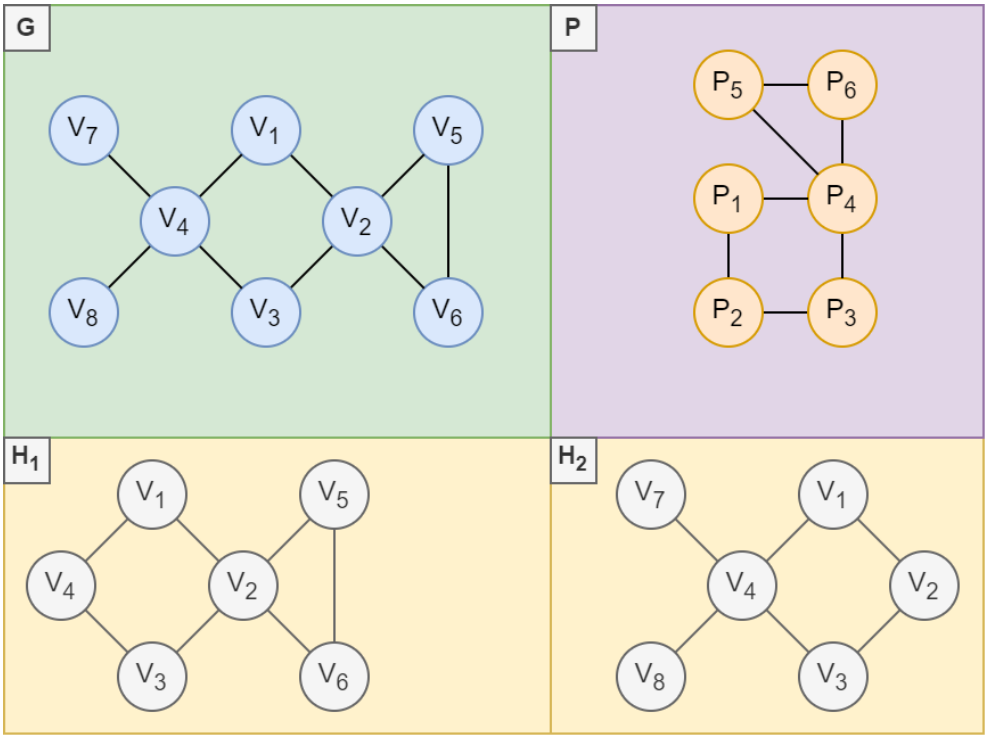

Figure 1. Target graph G, query graph P, exactly isomorphic H_1 and approximately isomorphic H_2 subgraphs of G

within a larger graph that matches the structure of a smaller graph, essentially seeking a subgraph that follows the connectivity pattern of a Hamiltonian cycle. This connection highlights the complexity and generality of the subgraph isomorphism problem.

Moreover, enumerating all isomorphic subgraphs adds an extra layer of difficulty, proving to be more challenging than addressing a singular decision variant (Afrati et al., 2013). Therefore, heuristic algorithms emerge as the main approach for tackling the SI problem.

Two distinct types of subgraph isomorphism problem exist: (I) Exact subgraph isomorphism (Carletti et al., 2015; Ullmann, 1976) and (II) Approximate subgraph isomorphi (Peng et al. 2017). Solutions for the approximate SI problem accommodate for slight variances in vertices, edges, and other semantic properties such as vertex labels, providing an approximate solution. The challenge with exact SI problem, however, rests on finding subgraphs that align flawlessly with the query graph. For example, in Figure 1, subgraph H_1 of G is exactly isomorphic to the query graph P. This isomorphism is achieved by establishing such a mapping between vertices: $P_1 \rightarrow v_1, P_2 \rightarrow v_2, P_3 \rightarrow v_3, P_4 \rightarrow v_4, P_5 \rightarrow v_5, P_6 \rightarrow v_6$. All the edges between any two query vertices, such as $e = (P_1, P_2)$, exists between their corresponding mappings $e' = (v_1, v_2)$. On the other hand, subgraph H_2 requires the presence of an edge (v_7, v_8) to be isomorphic to the query graph P. However, H_2 may be considered as a valid solution while doing approximate querying if the algorithm tolerates a single missing edge.

Solution strategies for the exact SI problem fall into two primary categories: (I) Index-based algorithms and (II) Constraint-based algorithms. In constraint-based algorithms (Cordella et al., 2004; Han, Lee, & Lee, 2013; He and Singh, 2008; Shang et al. 2008; Ullmann, 1976), an initial vertex is recursively extended through the addition of relevant neighbor vertices. This process when suitable vertices for branching are exhausted of the complete isomorphic subgraph is found. On the other hand, index-based algorithms

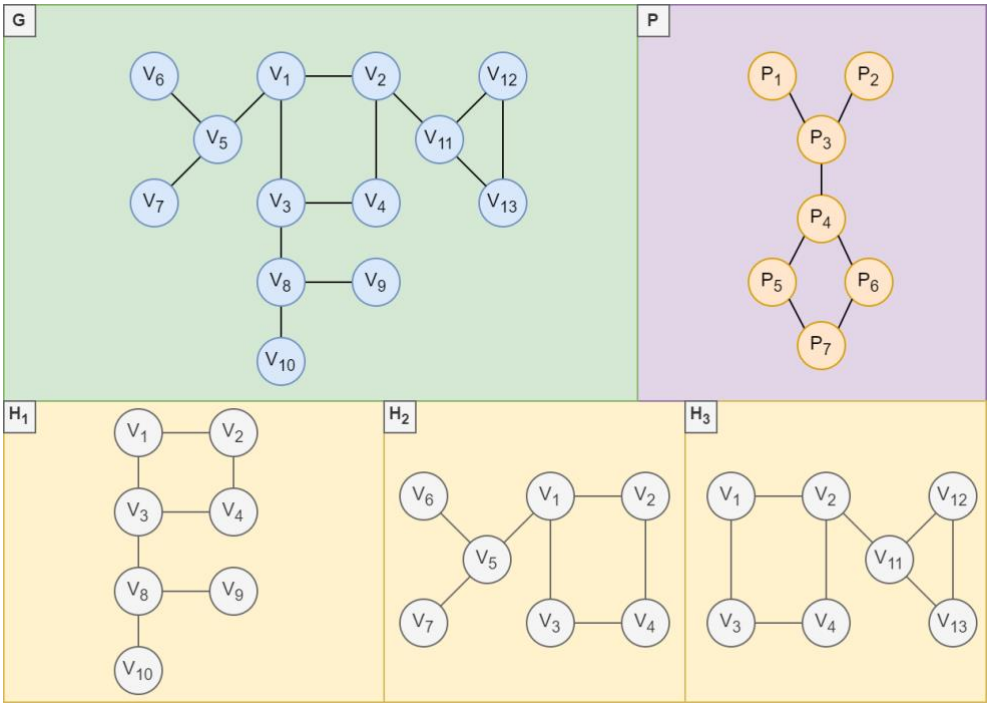

Figure 2. Target Graph G, Query Graph P, isomorphic subgraphs H_1, H_2 and non-isomorphic subgraph H_3

(Bonnici et al., 2010; Cheng et al., 2007; Kim et al., 2023) serve to improve the efficiency when dealing with large and dense graphs through a divide and filter approach. These algorithms manipulate sub regions of the query graph correlating to sub regions of the target graph. The idea behind the index-based algorithms include embedding the target graph into sub regions, filtering required embeddings through the query graph, and building partial solutions.

Despite these efforts, data sources' rapid expansion places immense demands on these algorithms for efficiency and resourcefulness, especially in maintaining efficiency across all graph types in SI problems. There are such algorithms (Peng et al., 2015) resort to hardware technologies and parallel programming in handling large graphs. However, this approach has its limitations, especially when financial resources limit the tasks performed and thus not a practical solution to the problem. By tailoring the solutions to a given graph type, performance can be substantially enhanced (Koca and Sevilgen, 2019).

Bipartite graphs prove its significance in modeling related networks like pathogen-host protein-protein relation networks or customer-product recommendation networks. Thus, an efficient solution to the SI problem on bipartite graphs is critical for revealing latent biological or social patterns in wide scale graphs, often encompassing tens of thousands of vertices.

In this study a novel index-based solution algorithm is proposed for the exact subgraph isomorphism problem on bipartite graphs. The algorithm consists of three primary steps consistent with index-based approaches: Firstly, embedding the target graph G using P_3 (triplet) structures and storing them in a hash-map. Secondly, embedding the query graph P and filtering the hash-map to identify suitable candidate embeddings for P. Finally, joining the candidate triplets to generate the exact solutions.

The proposed algorithm's performance is compared against existing state-of-the-art algorithms based on the execution times. Our algorithm achieved 2 to 500 times shorter querying times than its competitors. These results demonstrate that our algorithm

outperforms state-of-the-art algorithms developed for general graphs in solving the exact subgraph isomorphism problem on bipartite graphs.

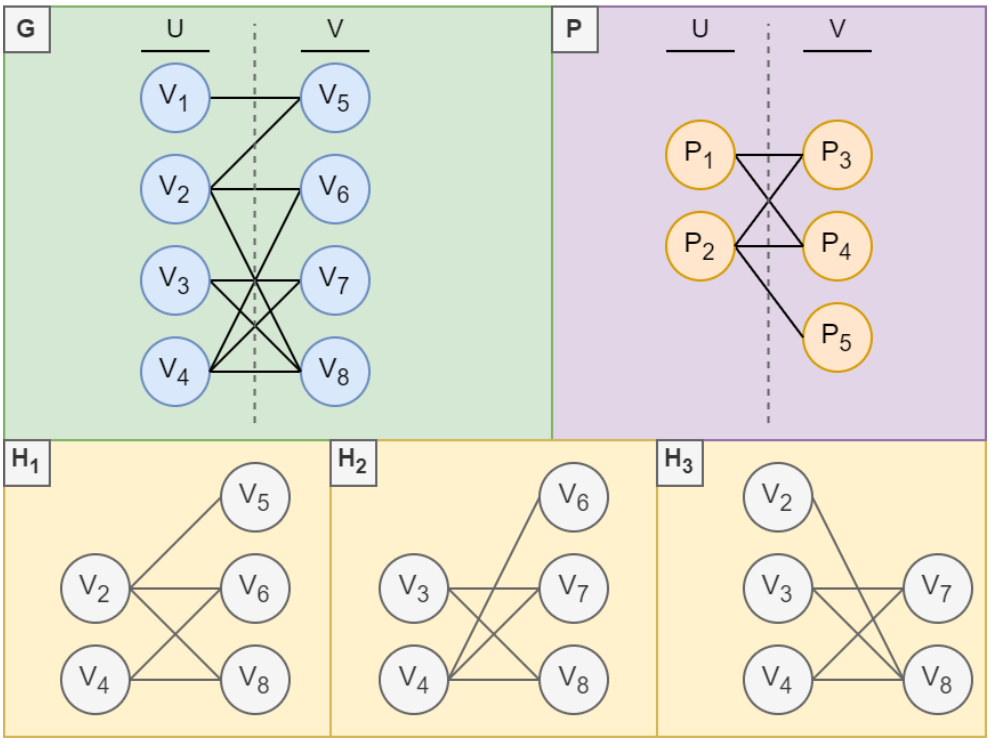

Figure 3. Target bipartite graph G, query bipartite graph P, the subgraphs that bipartite isomorphic to P H_1, H_2 and non-isomorphic bipartite subgraph H_3

2. Preliminaries

The fundamental definitions are presented to enhance the understanding of the study. The subgraph isomorphism (SI) problem is an extension of the graph isomorphism problem.

Definition 1 (Graph isomorphism): $G_1 = (V, E)$ and $G_2 = (V', E')$ are given two graphs. If there is a surjective function A from G_1 to G_2, $A: G_1 \rightarrow G_2$ such that, $\forall\, v \in V, A(v) \in V'$ and $\forall\, e = (v_1, v_2) \in E$, $A(e) = (A(v_1), A(v_2)) \in E'$, then G_1 and G_2 are isomorphic graphs, $G_1 \cong G_2$.

For isomorphic graphs, there is at least one mapping that allows the vertices of one graph to be aligned with the vertices of the other graph, preserving their adjacency relationships. For instance, in Figure 2, graphs H_1 and H_2 are isomorphic graphs. This isomorphism can be achieved by mapping the vertices $V_1, V_2, V_3, V_4, V_8, V_9, V_{10}$ of H_1 to $V_2, V_4, V_1, V_3, V_5, V_6, V_7$, respectively, in H_2.

Definition 2 (Subgraph - induced subgraph): $G = (V, E)$ is a given graph, any substructure $H = (V', E')$ of G such as $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$ is a subgraph of G. If E' contains all the edges in E that are between vertices in the set V' that is $E' = E \cap (V' \times V')$ then, H is an induced subgraph of G.

In the context of the SI problem, it is crucial to distinguish between the terms subgraph and induced subgraph. The solution algorithm may not consider extra edges when searching for non-induced subgraphs. For instance, in Figure 2, all three subgraphs H_1, H_2, and H_3 are isomorphic to P. However, H_3 is not a valid solution for an algorithm targeting induced subgraphs of G. This study focuses on finding isomorphic induced subgraphs; hence, the term subgraph is used to refer to induced subgraphs in this paper.

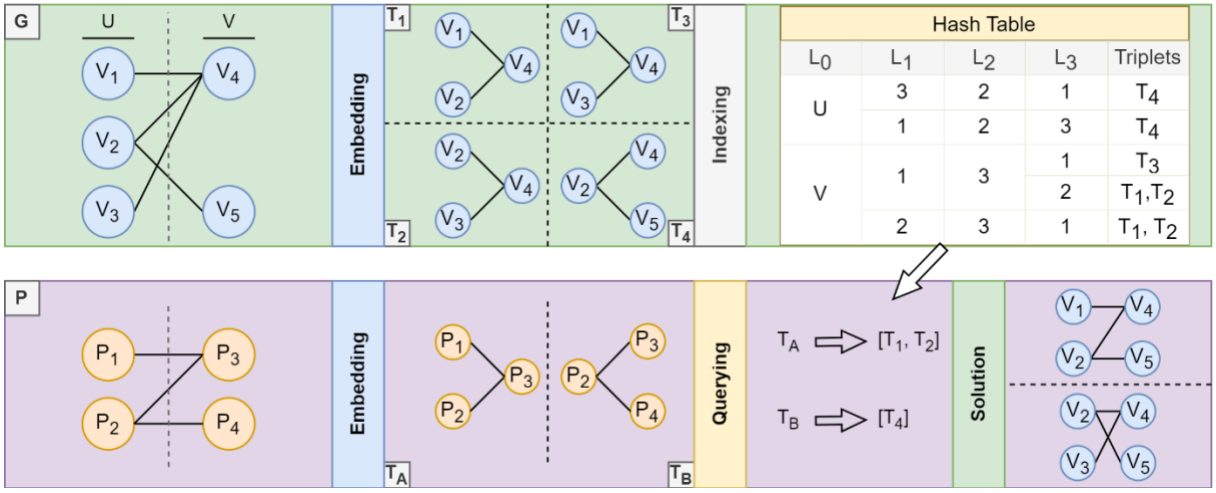

Figure 4. The workflow of the proposed subgraph isomorphism algorithm is illustrated. The target graph is embedded into triplets, and these triplets are stored in a multi-level key hash map (top). During the search for isomorphic subgraphs using a query graph, the query is also initially embedded into triplets. Subsequently, the suitable candidates for each query triplet are selected from the hash map through a filtering process. Finally, the exact solutions are generated by joining candidate triplets while taking into account the connections of the triplets within the query graph (bottom)

Definition 3 (Subgraph isomorphism problem): $G = (V, E)$ and $P = (V', E')$ are given two graphs where $|V'| \leq |V|$, $|E'| \leq |E|$. The subgraph isomorphism problem is finding all subgraphs H in the G such that $H \cong P$.

The graph G in Figure 2 has two subgraphs H_1 and H_2 that isomorphic to P. However, H_3 is not isomorphic to P because the edge $e = (v_{12}, v_{13})$ makes any mapping possible.

Definition 4 (Bipartite graph): $G = (U, V, E)$ is a bipartite graph where U and V are two disjoint and independent vertex sets and E is edge set that consists of edges $e = (u, v)$ where u is in U and v is in V.

In Figure 3, all the graphs depicted are bipartite graphs. Bipartite graphs are characterized by having two disjoint and independent vertex sets, denoted as U and V, as shown in the figure. In a bipartite graph, every edge connects two vertices that belong to different sets, meaning there are no edges within the same set. This property distinguishes bipartite graphs from general graphs where edges can connect any pair of vertices.

Definition 5 (Bipartite isomorphism): $G = (U, V, E)$ and $G' = (U', V', E')$ are given two bipartite graphs. If there is a bijection $I: (V \to V', U \to U')$ such that $\forall u \in U$, $v \in V$ and $e = (u, v) \in E; I(u) \in U', I(v) \in V'$ and $(I(u), I(v) \in E'$ then, G and G' are isomorphic bipartite graphs.

Indeed, in bipartite graphs, the partitioning of vertices into different partite sets is crucial for bipartite isomorphism. The partite sets represent distinct groups of objects, such as clients-servers or human protein-virus protein, and the relationships between them are captured by the edges connecting vertices from different sets. Consequently, mirrored subgraphs, where the partite sets are swapped, do not convey the same meaning.

Figure 3 provides an example of bipartite isomorphic graphs. The graphs H_1 and H_2 are bipartite isomorphic since they exhibit the same vertex-partitioning pattern and maintain the relationships between the two partite sets. However, graph H_3 is not bipartite isomorphic to H_1 and H_2, even though all three graphs are isomorphic in the general sense.

Definition 6 (SI problem in bipartite graphs): $G = (U, V, E)$ and $G' = (U', V', E')$ are given two bipartite graphs such that $|U| \leq |U'|$, $|V| \leq |V'|$, $|E| \leq |E'|$. The SI problem on the bipartite graphs is finding all subgraphs H of G' that are bipartite isomorphic to G.

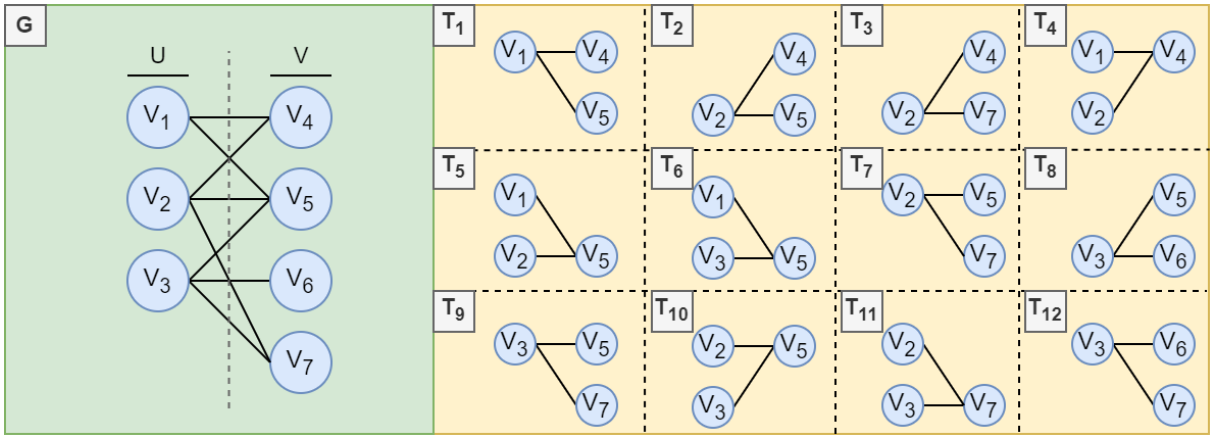

Figure 5. The target graph G and its triplet embeddings

The subgraph isomorphism problem on bipartite graphs is almost the same as the SI problem. However, only bipartite isomorphic subgraphs are considered valid solutions. In Figure 3, the subgraphs H_1 and H_2 are in the solution set of the SI problem for G and P. However, H_3 is not bipartite isomorphic to P because the vertex V_2, which is the mapping of P_5, is in the set V instead of the set U.

As all essential definitions are presented, In Section 3, our novel algorithm is introduced, and all the sub-processes are explained in detail. Moreover, the advantages of the bipartite structure are explicitly described for each sub-process. The experimental results are given and discussed in Section 4. Finally, our conclusions regarding this study are presented in Section 5.

3. Methodology

Our algorithm utilizes an enumerate, filter, and build approach, which is commonly employed in state-of-the-art algorithms for solving the SI problem. The workflow of the algorithm is illustrated in Figure 4. Initially, the target graph G is embedded into triplets and subsequently stored within a multi-level hash map. This hash map is then employed to fulfill subgraph querying requests. During the SI search phase, the query graph is also embedded into triplets as a preliminary step. Relevant triplets within G are extracted through hash map queries to establish potential matches for the query graph's triplets. Ultimately, these appropriate candidates are joined together, accounting for the structural attributes of the query graph, with the aim of constructing exactly isomorphic subgraphs with the query. These procedures have been specifically adapted for bipartite graphs to enhance the performance of SI searches for this graph type. The details of how the proposed algorithm leverages the bipartite structure of the target graph in each procedure and elucidates the resulting improvements in performance.

3.1. Embedding

Many state-of-the-art algorithms addressing the SI problem primarily focus on enhancing the detection of sub regions within the target graph that may potentially contain the query subgraph. These algorithms often employ graph embedding techniques to enumerate the sub-

parts of the target graph. Among the available graph embedding techniques, vertex embedding techniques are widely used due to their ability to provide valuable local information that can effectively aid in sub region filtering. These techniques make use of structural properties of vertices, such as vertex degrees, the count of k-length neighbors, and neighbors' degrees. For example, in (Tian and Patel, 2008), a vertex embedding technique is employed that stores a vector for each neighbor of a vertex. This vector incorporates information such as degree, neighbor count, label, and the sum of neighbor degrees. By utilizing such embedding techniques, algorithms can access highly informative and detailed local information for all vertices, thereby improving the accuracy of the filtering process.

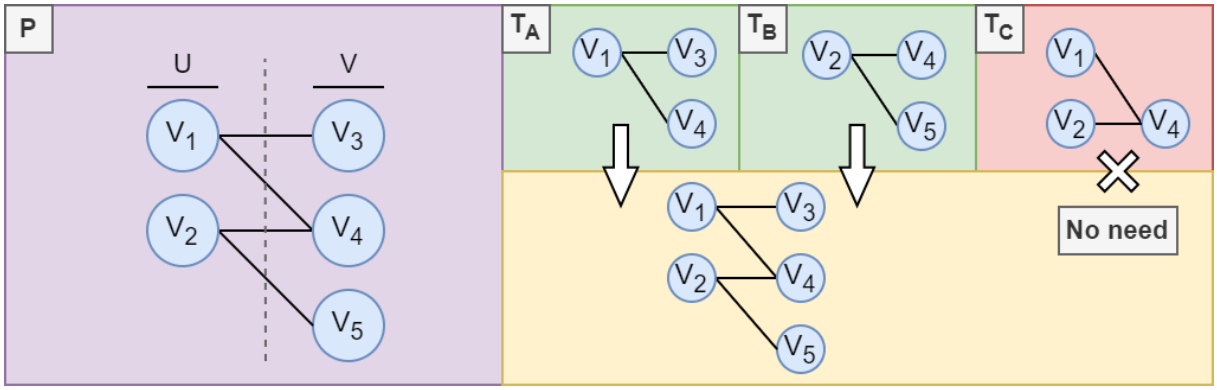

Figure 6. An example of generating solution building triplet set for the query graph P. The T_A and T_B triplets are sufficient to build query graph whereas the T_C triplet is unnecessary

The vertex embedding techniques can be inefficient when applied to huge target graphs. Building solutions by using the filtered candidates can become more challenging since the algorithm needs to extend partial solutions by one vertex at each step. As the number of partial solutions exponentially grows with each step, these techniques can result in being inoperative due to execution times for generating exact solutions.

To address these challenges, methods involve embedding more than one vertices together (Chen et al., 2007; Hong et al., 2015), as opposed to individually embedding vertices. These vertex groups often represent common sub structures such as cycles, paths, or other patterns. By considering these groups, the number of embeddings is reduced, leading to a smaller search space and enabling faster candidate filtering for parts of the query graph. Furthermore, since the embeddings consist of multiple components and the algorithm can add multiple components to partial solutions, the number of steps required for the solution process has the potential to decrease.

In this study, a path embedding technique is used instead of vertex embedding due to the mentioned problems. The length of the path, L, has been optimized since it is crucial for the performance of the embedding technique. If L is set to be too large, the embeddings have high resolution, making the filtering process more demanding. However, long paths lead to complicate the handling of complete solutions. In some cases, the solutions may not be feasible to handle because the algorithm may lack smaller sub structures necessary to construct a complete solution. On the other hand, when the path length is too small, it can lead to problems, as explained for single vertex embedding techniques. Various constraints are taken into account to optimize this parameter for bipartite graphs.

Initially, a path length of 1, denoted as $L = 1$ or P_2 is considered. Such path consisting of two vertices connected by a single edge. This sub structure is promising for embedding bipartite

graphs because each embedding consists of two vertices that hosted in different partitions. Such an information can be used for efficient filtering. However, in an undirected complete bipartite graph with partitions of sizes k and m, the total number of embeddings is equal to $k * m$. Since the solution algorithm benefits from the partitions of vertices in the embeddings, the partial solutions can be expanded only by suitable candidates that belong to the same partition order as their corresponding query embeddings.

The $L = 2$ structure, also known as P_3 or triplet, is evaluated since it also potentially suitable for embedding bipartite graphs. Each triplet consists of two tail vertices in one partition and one head vertex in the other partition, regardless of their adjacencies. While the number of embeddings are increased when using P_3 instead of P_2, triplets offers an opportunity to further enhance filtering and solution building performance compared to P_2. Triplets can handle three vertices and two edges simultaneously, leveraging the same structural information as P_2.

The $L = 3$ paths are also considered to explore if P_4 structures offer any additional benefits compared to triplets. However, it seems that P_4 structures provides no additional information specific to the bipartite structure. In fact, they can be precisely represented using two triplets. Therefore, it is concluded that values of L greater than 2 are not suitable for the bipartite graphs. Moreover, P_4 structures and longer paths have the potential to create loops, which must be checked by the algorithm to ensure compliance with the conditions of the induced subgraph isomorphism problem. However, such a control mechanism would add computational load to the algorithm. Hence, it is determined that the most suitable path length is 2 for bipartite graphs, as any path longer than 2 would decrease the algorithm's performance.

The embedding path structure can be expanded without altering the parameter L by incorporating three neighbors of the head vertex. This structure, known as the claw, contains the same information as triplets but allows for an increase in the number of components at each step of partial solution extension. However, the concatenation process of two claws takes more time than triplet concatenation due to the additional vertex that needs to be checked for suitability with the current partial solution. Another challenge with the claw structure arises during the generation of embeddings, as the presence of an extra vertex increases the number of embeddings. This increase in the number of embeddings can significantly impact the performance of the embedding and filtering processes.

Several of these structures have been tested, and the comprehensive experimental results are presented in the fourth part. The performance results obtained support our theoretical assumptions, indicating that triplets are the optimal data structure for embedding bipartite graphs.

3.2. Filtering

The performance of the algorithm heavily relies on the filtering process, which is as important as the embedding process. The filtering process starts by enumerating triplets in the query graph using the same embedding method used in the target graph. The embeddings of the target graph are then filtered to identify candidates for the query triplets based on their structural properties. While query graphs typically consist of a few triplets, the number of embeddings in the target graph can be in the millions, making effective filtering techniques essential to maintain the algorithm's efficiency.

Our algorithm leverages vertex degrees and partite information for efficient filtering. The triplet structure provides favorable locality for the solution building process. Since exact solutions require identical adjacency relations, vertex degrees serve as a powerful indicator for identifying suitable candidates.

Incorporating partite information into the algorithm ensures the elimination of unsuitable candidates during the solution building process, which can be computationally expensive. Furthermore, the partite information plays a crucial role in generating exact solutions in bipartite graphs, preventing the algorithm from producing mirrored solutions where all vertices are placed in the opposite partition.

Table 1. The hash-map of embeddings in Figure 5.

Level-0	Level-1	Level-2	Level-3	Triplet List
P1	1	3	2	T12
			3	T8
	2	2	3	T1
		3	1	T12
			2	T3
			3	T2, T7, T9
	3	2	2	T1
		3	1	T8
			2	T2, T7, T9
P2	2	2	3	T4
		3	3	T5, T6
	3	2	2	T4
			3	T11
		3	2	T5, T6
			3	T10

The embeddings of the target graph, as shown in Figure 6, are stored in a multi-level hash map similar to Table 1. This hash map enables efficient filtering with $O(1)$ access time. Pushing the embeddings into the hash map requires a total execution time of $O(n)$ due to the nature of map data structures ($O(1)\ per\ insertion\ *\ n\ embeddings$).

The filtering process aims to avoid complex search operations and provide $O(1)$ or, at most, $k\ *\ O(1)$ access time, ensuring that the keys satisfy the filtering constraints to a large extent.

The level-0 key of the map has two possible values, P_1 and P_2, representing the two partitions of the bipartite graph. The partite of a triplet is determined by the partite of its head vertex. For example, if the head vertex of a query triplet belongs to partition 1, the candidates for that triplet will be limited to the embeddings with a level-0 key value of P_1. By filtering the embeddings based on their partite information at the level-0 key, the algorithm significantly reduces the search space.

The level-1, level-2, and level-3 keys in the hash map store the vertex degrees of the triplets. Level-1 corresponds to the degrees of the left tail vertices, level-2 corresponds to the degrees of the head vertices, and level-3 corresponds to the degrees of the right tail vertices. It should be noted that the order of the left and right tails is interchangeable.

During the filtering process, it is necessary to consider the mirrored values of the level-1 and level-3 keys. To avoid the additional computational load of handling mirrored values, each embedding is stored in the hash map with both the regular key combination and an alternate key combination where the level-1 and level-3 keys are interchanged (as shown in Table 1, T_1). Although this approach doubles the memory usage for storing the embeddings, it reduces the CPU requirements of the filtering process.

In the filtering process, it is important to account for the presence of additional edges between vertices that are not part of the sub region in an isomorphic subgraph. These extra edges do not violate the definition of exact subgraph isomorphism but can increase the vertex degrees. To prevent filtering out valid candidates, the algorithm filters only the triplets that have a lower vertex degree for each key level. For example, if the left tail vertex of the query triplet has a degree of d, then only triplets with a vertex degree equal to or greater than d for the left tail vertex will remain for the next levels.

The elimination process involves searching the keys of the hash map, resulting in a filtering process that takes more than $O(1)$ execution time. However, since there can be at most n different degrees in a graph, the filtering process takes $3 * n * O(1) = O(n)$ execution time in the worst case. In practical scenarios, the degree diversity is typically much less than the number of vertices, so the filtering process is usually completed quickly.

By default, the proposed algorithm doesn't consider vertex labels or edge directions, as the focus is on solving the subgraph isomorphism problem in bipartite graphs in the most general form. However, the algorithm can support the inclusion of these constraints by adding additional levels to the data structure. For example, vertex labels can be incorporated into the filtering process by adding three distinct keys to the data structure, allowing the checking of triplet vertices based on their labels before considering their compatibility with vertex degrees. Similarly, the algorithm can be adapted to support inexact searches by adding one or two levels for vertex labels. The absence of keys for the labels provides flexibility, enabling researchers or end-users to customize the algorithm according to their specific needs.

3.3. Building solution

The process of building exact solutions involves joining candidates of query triplets together to generate partial solutions. These partial solutions are expanded by adding exactly one candidate on each step. If a partial solution cannot be expanded and is isomorphic to the query graph, it is considered a valid exact solution. However, if a partial solution has missing vertices and there is no possibility for further expansion, it is eliminated.

When a candidate joins a partial solution, it is done while considering the adjacency relations of its owning triplet. The partial solution represents a mapping of a sub region in the query graph, and some vertices of the owner triplet may already be mapped in the current partial solution. This means that some vertices in the partial solution must match the vertices of the new candidate. Therefore, partial solutions can be expanded with candidates that share the same adjacency relation between their owner triplet and the mapped sub-region. By considering these adjacency relations, the algorithm ensures that the candidate being added is compatible with the existing mapping.

This step-by-step expansion and mapping process allows the algorithm to gradually build exact solutions by iteratively adding compatible candidates to the partial solutions, ultimately leading to a complete and valid mapping of the query graph.

The joining process of two triplets involves considering common vertices between them. Candidates of one triplet can generate partial solutions by joining with candidates of the other triplet if they share a common vertex. Some candidates may not be compatible due to mismatched mappings. Joining triplets allows the algorithm to gradually build partial solutions by expanding them with compatible candidates.

Not all triplets in the query graph are necessary for building complete solutions. Triplets whose vertices have already been mapped in the current partial solution do not contribute

additional information. Joining such triplets with existing partial solutions does not generate new solutions or eliminate existing ones.

Selecting the optimal set of triplets, known as the Solution Building Triplet Set (BTS), is crucial for algorithm performance. The optimal BTS minimizes the number of joining processes required, as joining triplets can be computationally expensive. However, finding the optimal BTS is challenging, and there is no greedy method that guarantees an optimal solution. Brute-force algorithms can be used to generate all possible BTSs and estimate their join count based on candidate numbers. While this approach can yield efficient BTSs, it becomes impractical for larger BTS sizes due to excessively long execution times.

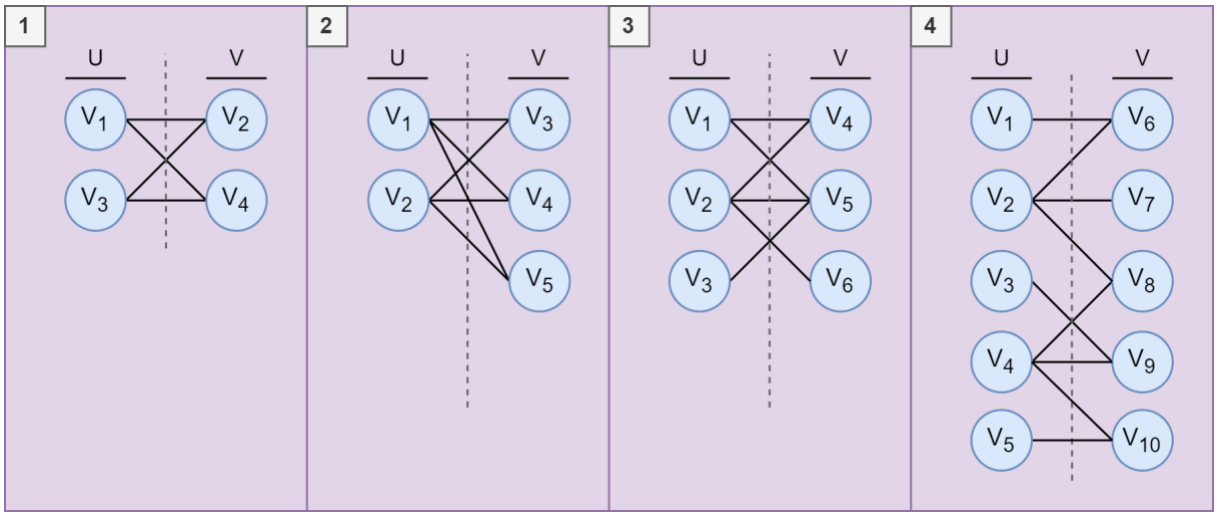

Figure 7. The four query graphs used in both experiments

The proposed greedy algorithm aims to find efficient Solution Building Triplet Sets (BTS) in a time-efficient manner. The algorithm sorts query triplets based on the number of candidates they have, with the triplet having the fewest candidates added as the first triplet in the BTS. Privileged triplets are marked and added to the BTS based on the number of candidates they have. Triplets with more uncontained edges become privileged for each step.

The ordering of triplets within the BTS is also optimized to reduce the number of unsuitable partial solutions. The triplet with the highest number of candidates is placed at the head of the ordered BTS. The next triplet is selected from the triplets that have more contained vertices by the ordered BTS, with priority given to triplets with the highest number of candidates. This selection reduces the generation of unsuitable partial solutions and decreases the number of partial solutions for subsequent steps.

By utilizing these greedy approaches, the algorithm aims to minimize unnecessary computations by reducing the number of joins and unsuitable partial solutions, ultimately improving the performance of building exact solutions.

4. Results and Discussions

4.1. Experimental Setup

The performance of the proposed algorithm is assessed both with and without the enhancements suggested for addressing sub-problems. Additionally, the empirically best

version of the algorithm is compared against state-of-the-art subgraph isomorphism algorithms to demonstrate its efficiency. The execution time of the querying process of algorithms is used as the performance metric. Both the proposed algorithm and its competitors are implemented in the Python programming language. The comparative algorithms are evaluated using their publicly accessible implementations. The experiments are carried out on a computer equipped with a 3.3 GHz i7 CPU and 16 GB of RAM, running the Ubuntu operating system.

4.2. BTS Selection Methods Comparison

Two distinct experimental setups were conducted to assess improvements in determining the reconstruction strategy for the query graph solution. Initially, the selection algorithms of the BTS were evaluated in terms of the time required to calculate BTS and the number of joins needed to identify all solutions using the computed BTS. Subsequently, the algorithms for ordering BTS to construct solutions were assessed based on their execution time for generating solutions.

The protein-protein interaction network between Adenoviridae proteins and Human proteins obtained from PHISTO (Durmuş Tekir et al., 2013). Database is used as target graph in both experiments. This interaction graph comprises 324 vertices with 379 interactions among them. The query graphs, as depicted in Figure 7, were employed in both experiments to evaluate how the tested algorithms perform with query graphs of varying sizes. Each individual test was repeated 10 times, and the average execution times are presented.

The greedily selection approach for generating BTS was compared with the brute-force selection algorithm, revealing that the greedy algorithm is 10 to 1000 times faster than its alternative, as indicated by the experimental results. This substantial difference in BTS selection holds particular significance in fields that frequently encounter changing query graphs. The results suggest that the brute-force approach tends to become impractical as the size of the query graphs increases, while the greedy approach exhibits greater resilience to such increases in query graph size. However, it is worth noting that BTS selection is a one-time process, and the brute-force approach may still be preferable in situations where time constraints are not a primary concern, particularly in fields that consistently encounter the same query graphs.

The algorithms are further assessed based on the quality of the BTS they generate. The performance of a BTS is gauged by comparing the total number of joins required to generate all solutions using that BTS. The results, as presented in Table 2, indicate that the greedy algorithm largely produced optimal solutions for three out of the four query graphs. However, for query 4, the brute-force algorithm's optimal solution outperformed the BTS generated by the greedy algorithm in terms of join count. It is important to note that the greedy algorithm may increase query time due to the growing number of join operations needed for constructing solutions. Therefore, the choice of BTS selection method should be made based on the frequency of query changes within the specific field of application.

Table 2. Performance comparison of reconstruction set selection algorithms.

Algorithm		Query (1)	Query (2)	Query (3)	Query (4)
Greedy	Exec time	3.17×10^{-5}	8.37×10^{-5}	8.20×10^{-5}	1.15×10^{-4}
	# of Joins	73984	458480	122600	37550538
Brute-Force	Exec time	1.48×10^{-4}	1.54×10^{-2}	6.56×10^{-3}	7.38×10^{-1}
	# of Joins	73984	458480	122600	20959821

The evaluation of the algorithms for ordering the triplets within the BTS is conducted by measuring the query execution times incurred when building solutions by joining the triplets in the order in which they were generated. There are two types of strategies for ordering BTS, distinguished by whether they take into account the connection with the current partial query when selecting the next triplet for ordering. Additionally, various approaches for selecting the most advantageous triplet among the current candidates can be explored. In this study, we tested the strategy of selecting the next triplet based on the number of candidates it has in the target graph.

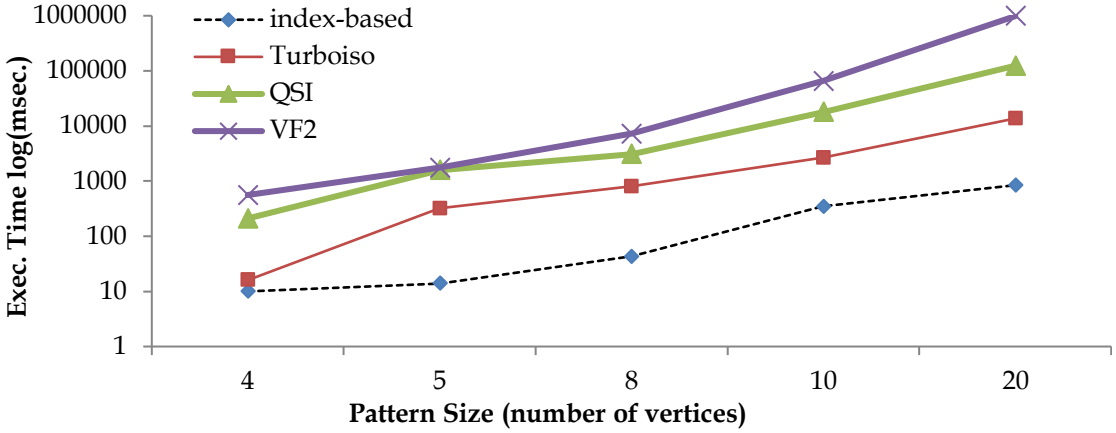

Figure 8. Performance of comparison algorithms in subgraph isomorphism search on the protein-protein interaction graph of Poxviridae. The proposed algorithm consistently outperforms all other algorithms in the search for all subgraphs

The experimental results presented in Table 3 revealed that the strategy of selecting the next triplet from those that don't break connectivity exhibited superior performance when compared to competing strategy. Additionally, giving priority to triplets with the highest number of candidates in the target graph led to a remarkable enhancement in performance. As a result, the greedy approach was employed for candidate selection within the BTS in subsequent experiments. Furthermore, the selection of the next triplet during solution construction was determined by choosing the triplet with the maximum number of candidates from among the remaining triplets connected with the current partial solution.

Table 3. Comparative execution times of querying with various BTS ordering algorithms.

Approach	Candid count	Query (1)	Query (2)	Query (3)	Query (4)
Connected	Maximum	1.22×10^{-5}	5.41×10^{-2}	8.21×10^{-1}	2.47
	Minimum	9.05×10^{-4}	9.17×10^{-1}	2.25	98.25
Not connected	Maximum	1.10×10^{-5}	4.67	103.25	173.10
	Minimum	1.51×10^{-5}	1.21	19.37	125.92

4.3. Comparing With the State-of-art Methods

The proposed index-based solution method is compared with state-of-the-art subgraph isomorphism algorithms VF2 (Cordella et al., 2004), QuickSI (Shang et al., 2008) and Turbo^{iso} (Han et al., 2013). Two experiments are designed to evaluate the performance of our algorithm in bipartite graphs. In the first experiment, the performance of the algorithms is evaluated by comparing their execution times with queries of varying sizes. The protein-protein interaction dataset of Poxviridae serve as the target graph for these experiments, comprising 210 proteins and 454 interactions among them. The query graphs depicted in Figure 7 are employed in experiments. Furthermore, a larger query graph, represented as a complete bipartite graph with 20 vertices, is also utilized.

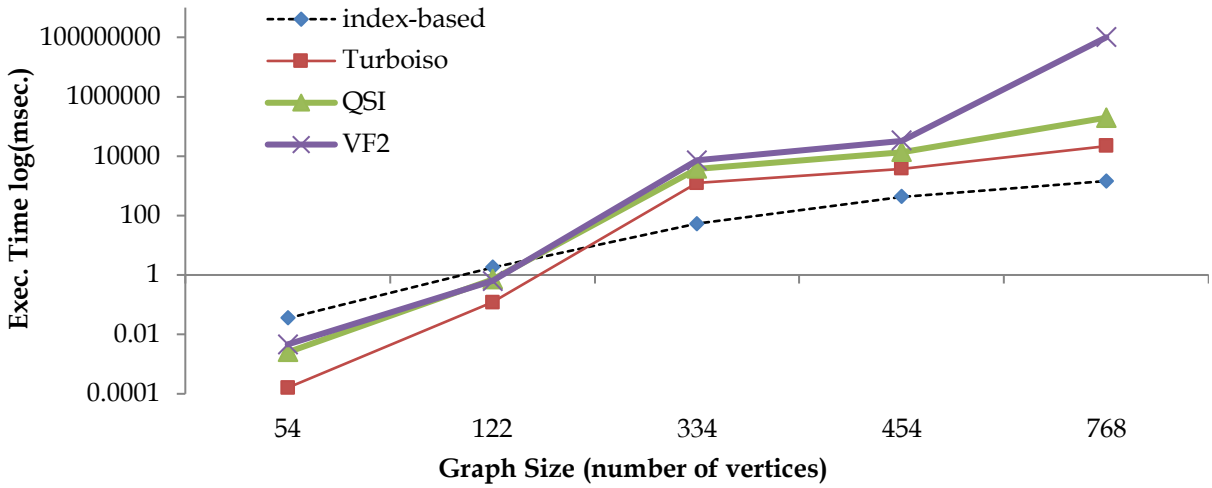

Figure 9. Performance comparison of state-of-the-art algorithms on different target graphs. The results indicate that our algorithm outperforms others when the target graph size exceeds a few hundred vertices. However, in small graphs, the algorithm exhibits poorer performance, primarily due to its preprocessing step

The tests are conducted ten times for each algorithm across all five query graphs. The resultings, as depicted in Figure 8, demonstrated the superior performance of the proposed algorithm, with notable improvements ranging from 2 to 500 times shorter querying times. It is worth noting that our algorithm consistently exhibited the best performance across all query sizes, while all methods exhibited acceptable execution times in the case of small query graphs. Particularly, the Turbo^{iso} algorithm displayed the second-best performance across all query graphs. As anticipated, the VF2 algorithm exhibited the poorest performance, in line with previous studies that have established the superior performance of both Turbo^{iso} and QuickSI. Nevertheless, it remains valuable to present VF2's performance as a reference point for baseline performance assessment.

In contrast to the performance in small query graphs, it is evident that the execution times of the algorithms increase quadratically with the query sizes, whereas our algorithm demonstrates a linear increase. This highlights the sustained usability of the proposed algorithm for larger query graphs, especially as its competitors become impractical for such scenarios. Consequently, the performance results showcase the algorithm's superiority, underscoring its robustness in the face of expanding query sizes.

We also assessed the algorithm's execution times on target graphs with varying sizes, and the performance results are depicted in Figure 9. These graphs encompass protein-protein interactions involving five different viruses and their proteins in relation to human proteins. Our algorithm exhibited poor performance than its competitors on small graphs containing 120 or fewer vertices, primarily due to its preprocessing step for index generation. However, the results illustrate that our algorithm demonstrates superior performance when the size of the target graphs exceeds a few hundred vertices.

The proposed algorithm exhibits greater robustness for growing target graph sizes compared to all other algorithms in the comparison. However, Turbo^{iso} and QuickSI demonstrate greater resilience to increasing target graph sizes than to increases in query sizes. VF2 shows the poorest performance and robustness. The experimental results provide compelling evidence that our index-based algorithm stands as a prominent solution for addressing the subgraph isomorphism problem within bipartite graphs.

5. Conclusions

Our study presents substantial contributions to the field of subgraph isomorphism in bipartite graphs. We have addressed the pivotal challenge of identifying subgraphs that are isomorphic to a given query graph, acknowledging the complexities stemming from the non-linear nature of graphs and the ever-expanding size of datasets.

To address the exact subgraph isomorphism problem, we have introduced a novel index-based solution algorithm tailored specifically for bipartite graphs. Our approach employs a divide and filter strategy; the target graph is embedded using triplet structures and subsequently filtered based on vertex degrees and partitions of search graph. This approach significantly reduces the search space and enhances the overall efficiency of the algorithm.

Our experimental results conclusively demonstrate the superiority of our index-based algorithm when compared to state-of-the-art methods. It consistently achieves 2 to 500 times shorter querying times, underscoring its effectiveness and scalability for medium to large bipartite graphs.

Furthermore, our study has underscored the critical importance of reconstruction set selection in achieving computational efficiency. The careful selection of a subset of triplets that minimizes the number of join operations has proven to be highly effective for improving overall algorithm performance. Additionally, the ordering of the reconstruction set, starting with triplets featuring the maximum number of candidates and selecting neighboring triplets with high candidate counts, has proven to be a valuable strategy. Nonetheless, there is still room for enhancing the algorithm's performance by refining the ordering strategy. Additionally, exploring the possibility of joining multiple triplets that share intersecting join processes may be worthwhile, as it holds the potential to reduce solution construction time.

In summary, our study provides a comprehensive evaluation of different algorithms for solving the exact subgraph isomorphism problem on bipartite graphs. The proposed index-based algorithm stands out for its remarkable efficiency and effectiveness, establishing itself as a superior solution for large-scale bipartite graphs like pathogen-host protein-protein interaction networks.

Acknowledgements

We would like to thank Dr. Saliha Durmuş and Merve Yaşar Semiz for their contribution to setting up experiments and providing helpful feedback on our study.

Author Statement

The authors confirm their contributions to the paper as follows: algorithm conception and design: M.B. Koca, F.E. Sevilgen; implementation and experimentation: M.B. Koca; analysis of experimental results: M.B. Koca, F.E. Sevilgen; draft manuscript preparation: M.B. Koca, F.E. Sevilgen. All authors reviewed the results and approved the final version of the manuscript.

Conflict of Interest

The authors declare no conflict of interest.

References

Afrati, F. N., Fotakis, D., & Ullman, J. D. (2013). Enumerating Subgraph Instances Using Map-Reduce. *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, 62–73.

Bonnici, V., Ferro, A., Giugno, R., Pulvirenti, A., & Shasha, D. (2010). Enhancing Graph Database Indexing by Suffix Tree Structure. In T. M. H. Dijkstra, E. Tsivtsivadze, E. Marchiori, & T. Heskes (Eds.), *Pattern Recognition in Bioinformatics*. Berlin, Heidelberg: Springer.

Carletti, V., Foggia, P., & Vento, M. (2015). VF2 Plus: An Improved Version of VF2 for Biological Graphs. In C.-L. Liu, B. Luo, W. G. Kropatsch, & J. Cheng (Eds.), *Graph-Based Representations in Pattern Recognition*. Cham: Springer International Publishing.

Chen, C., Yan, X., Yu, P. S., Han, J., Zhang, D.-Q., & Gu, X. (2007). Towards Graph Containment Search and Indexing. *Proceedings of the 33rd International Conference on Very Large Data Bases*, 926–937. Vienna, Austria: VLDB Endowment.

Cheng, J., Ke, Y., Ng, W., & Lu, A. (2007). Fg-index: Towards Verification-free Query Processing on Graph Databases. *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data*, 857–872. New York, NY, USA: Association for Computing Machinery.

Cook, S. A. (2023). The Complexity of Theorem-Proving Procedures. *In Logic, Automata, and Computational Complexity:* The Works of Stephen A. Cook (1st ed.). New York, NY, USA: Association for Computing Machinery.

Cordella, L. P., Foggia, P., Sansone, C., & Vento, M. (2004). A (Sub)graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 26(10), 1367–1372.

Corneil, D. G., & Gotlieb, C. C. (1970). An Efficient Algorithm for Graph Isomorphism. *Journal of the ACM*, 17(1), 51–64.

Durmuş Tekir, S., Çakır, T., Ardıç, E., Sayılırbaş, A. S., Konuk, G., Konuk, M., Ülgen, K. Ö. (2013). PHISTO: Pathogen–host interaction search tool. *Bioinformatics*, 29(10), 1357–1358.

Han, W.-S., Lee, J., & Lee, J.-H. (2013). Turboiso: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*, 337–348. New York, NY, USA: Association for Computing Machinery.

He, H., & Singh, A. K. (2008). Graphs-at-a-time: Query Language and Access Methods for Graph Databases. *Proceedings of the 2008 ACM SIGMOD International Conference on*

Management of Data, 405–418. New York, NY, USA: Association for Computing Machinery.

Hong, L., Zou, L., Lian, X., & Yu, P. S. (2015). Subgraph Matching with Set Similarity in a Large Graph Database. *IEEE Transactions on Knowledge and Data Engineering*, 27(9), 2507–2521.

Kim, H., Choi, Y., Park, K., Lin, X., Hong, S.-H., & Han, W.-S. (2023). Fast Subgraph Query Processing and Subgraph Matching via Static and Dynamic Equivalences. *The VLDB Journal*, 32(2), 343–368.

Koca, M. B., & Sevilgen, F. E. (2019). A Novel Approach for Subgraph Isomorphism Problem on Bipartite Graphs. *2019 27th Signal Processing and Communications Applications Conference (SIU)*, 1–4.

Liang, Y., & Zhao, P. (2017). Similarity Search in Graph Databases: A Multi-Layered Indexing Approach. *2017 IEEE 33rd International Conference on Data Engineering (ICDE)*, 783–794.

Peng, Y., Fan, Z., Choi, B., Xu, J., & Bhowmick, S. S. (2015). Authenticated Subgraph Similarity Searching Outsourced Graph Databases. *IEEE Transactions on Knowledge and Data Engineering*, 27(7), 1838–1860.

Shang, H., Zhang, Y., Lin, X., & Yu, J. X. (2008). Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *Proceedings of the VLDB Endowment*, 1(1), 364–375.

Tian, Y., & Patel, J. M. (2008). TALE: A Tool for Approximate Large Graph Matching. *2008 IEEE 24th International Conference on Data Engineering*, 963–972.

Ullmann, J. R. (1976). An Algorithm for Subgraph Isomorphism. *Journal of the ACM*, 23(1), 31–42.