mathematics

MDPI

Article

Subgraph Query Matching in Multi-Graphs Based on Node Embedding

Muhammad Anwar [1], Aboul Ella Hassanien [2,3], Václav Snášel [4] and Sameh H. Basha [1,5,*]

[1] Department of Mathematics, Faculty of Science, Cairo University, Giza 12613, Egypt
[2] Faculty of Computers and Information, Cairo University, Giza 12613, Egypt
[3] Scientific Research Group in Egypt (SRGE), Giza 12613, Egypt
[4] Faculty of Electrical Engineering and Computer Science, VŠB-Technical University of Ostrava, 708 33 Ostrava, Czech Republic
[5] Faculty of Science, Galala University, Suez 43511, Egypt
* Correspondence: samehbasha@cu.edu.eg or samehbasha@gu.edu.eg

Abstract: This paper presents an efficient algorithm for matching subgraph queries in a multi-graph based on features-based indexing techniques. The KD-tree data structure represents these nodes' features, while the set-trie index data structure represents the multi-edges to make queries effectively. The vertex core number, triangle number, and vertex degree are the eight features' main features. The densest vertex in the query graph is extracted based on these main features. The proposed model consists of two phases. The first phase's main idea is that, for the densest extracted vertex in the query graph, find the density similar neighborhood structure in the data graph. Then find the k-nearest neighborhood query to obtain the densest subgraph. The second phase for each layer graph, mapping the vertex to feature vector (Vertex Embedding), improves the proposed model. To reduce the node-embedding size to be efficient with the KD-tree, indexing a dimension reduction, the principal component analysis (PCA) method is used. Furthermore, symmetry-breaking conditions will remove the redundancy in the generated pattern matching with the query graph. In both phases, the filtering process is applied to minimize the number of candidate data nodes of the initiate query vertex. The filtering process is applied to minimize the number of candidate data nodes of the initiate query vertex. Finally, testing the effect of the concatenation of the structural features (orbits features) with the meta-features (summary of general, statistical, information-theoretic, etc.) for signatures of nodes on the model performance. The proposed model is tested over three real benchmarks, multi-graph datasets, and two randomly generated multi-graph datasets. The results agree with the theoretical study in both random cliques and Erdos random graph. The experiments showed that the time efficiency and the scalability results of the proposed model are acceptable.

Keywords: multigraph mining; pattern mining; matching problem; core number; KD-tree; node embedding

MSC: 05C85; 68R10; 05C60; 05C90

Citation: Anwar, M.; Hassanien, A.E.; Snášel, V.; Basha, S.H. Subgraph Query Matching in Multi-Graphs Based on Node Embedding. *Mathematics* **2022**, *10*, 4830. https://doi.org/10.3390/math10244830

Academic Editor: Ezequiel López-Rubio

Received: 10 October 2022
Accepted: 8 December 2022
Published: 19 December 2022

1. Introduction

Data are critical and essential to deal with in every field. Due to a huge amount of internet and database applications, there is a need to deal with huge data. Many real-world datasets can be represented by a graph which consists of set of vertices and set of edgesepresenting nodes, and interaction between entities is represented as edges [1]. A simple graph is a graph where only one edge is allowed between its vertices. Most of the real-world datasets are represented by multi-graphs in which more than one edge type between vertices are allowed in such graphs. Although dealing with multi-graph data is very important, most of the research in graphs deal with a simple graph. The simple graph represents many interaction systems such as social networks, Resource Description

Framework (RDF) data, chemical compounds, protein–protein interaction (PPI) networks, physical interactions, and complex networks. Moreover, the multi-graph represents the more powerful presentation of real interaction systems because its presentation has many relationships between objects of the system, so the multi-graph has a lot of information to explore valuable patterns.

One of the most important mining problem patterns in dealing with graphs is the subgraph-matching problem [2]. This problem can be represented as given G, which is a graph (simple graph or multi-graph) (for more details see Definitions 1 and 2), and given q, which is a query graph (simple graph or multi-graph). The subgraph-matching problem is to find all matches of q in G. This problem has two directions: exact or inexact (approximate) graph matching. Exact graph matching is known as graph isomorphism and requires the function between two graphs' vertices to preserve the adjacency. Moreover, it needs to compute the optimal mapping. However, inexact graph matching is based on computing a sub-optimal mapping as an assignment problem.

Generally, the subgraph isomorphism problem is a Non-deterministic problem (NP-complete problem). However, some specific graphs can also have a small complexity [3], for instance, the particular case in which the big graph is a forest and the small one to be matched is a polynomial-complexity tree. Some existing scenarios deal with such a problem as the feature-based indexing and the enumeration approach. In the feature-based indexing approach, which is defined by mapping vertices of a graph to vector space (feature space), which will be used to minimize the candidate set for each query vertex, this vector space is used in similarity, classification, and clustering tasks. Feature-based indexing is followed by filtering and verification. Some graph patterns are chosen as indexing features throughout the filtering in order to reduce candidate graphs. In the verification, they are checking for the subgraph isomorphism using the selected candidate.

The enumeration method relies on backtracking techniques. This approach aims to find embedding by the growth of partial solutions and avoids using indexing. There exist other approaches; one of them is based on defining the equivalence classes at query or database level or both and defining them by exploiting vertex relationships.

Cordella et al. [4] presented the first version of their subgraph isomorphism algorithm, where they examined its performance for the isomorphism of small and medium-size graphs. The algorithm, using a set of feasibility rules, allows us to significantly prune the search space and the computational cost of the matching process.

Most research in subgraph isomorphism problems focuses on obtaining small candidate sets, producing effective matching orders, and improving searching methods. Filtering methods are used to obtain small candidate sets. Defining new data structures aims to acquire small candidate sets as well as produce effective matching orders, while using symmetry breaking has a main role in improving the searching methods.

In [5] they presented an algorithm for graph/subgraph isomorphism suited for dealing with large graphs. The first version algorithm is improved by analyzing the features in detail with special reference time and memory requirements. The technique offers a wide range of applications because the graph topology is not constrained. A state-space representation (SSR) of the matching process and five feasible rules for decreasing the search tree are described. The selected representation enables one to simultaneously compare the syntactic and semantic properties of the node–pair pairs that need to match. The primary improvement made to an early implementation of the VF algorithm, described in [4], is that the data structures utilised during the exploration of the search area are set up in a way that significantly reduces memory requirements. As a result, the method may match graphs with a lot of nodes and branches.

Moreover, some early subgraph-matching algorithms such as VF and other (see [4,6–9]) find candidate sest by using local filters that consider the neighborhood of vertices. On the other hand, $Turbo_{iso}$ and others such as [10–13] build auxiliary data structures on a query and data graph to obtain small candidate sets and produce effective matching orders by estimating as precise a search cost as possible.

In [14], the search process is performed via a backtracking algorithm that, at each step, reduces the bit-vector domain. Before starting the search, it completes two preliminary steps. The first one, Prematch, fills domains by using vertex invariant to select them based on labels and topology. The second step avoids this by locally guaranteeing that two pattern vertices cannot match the same target vertex. After the initial stages, the pattern's vertices are arranged as indicated by a static search method. The pattern vertex with the most branches between it and the partial solution is the one that will be matched next. The sequence begins with each pattern vertex that has a single, compatible target vertex. It chooses the vertex with the biggest sum of its neighbours' degrees when there are two vertices that are equally qualified to be the following vertex in the ordering.

The constraint fulfilment issue with subgraph isomorphism can be explained. Finding an assignment of values (target vertices) to all variables such that all criteria are met given a collection of variables (pattern vertices) and a set of constraints constitutes a constraint fulfilment problem for the subgraph isomorphism problem.

Christine Solnon in [15] showed that the constraint fulfilment issue with subgraph isomorphism can be explained. Finding an assignment of values (target vertices) to all variables such that all criteria are met given a collection of variables (pattern vertices) and a set of constraints constitutes a constraint fulfilment problem for the subgraph isomorphism problem.

Messmer and Bunke in [16] proposed a new method for graph and subgraph isomorphism detection based on a decision tree representation. The decision tree is generated offline from a priori-known model graphs. At run time, the decision tree is used to detect all graph and subgraph isomorphisms from an input graph to any of the model graphs in time that is only polynomial in the graphs' size and independent of the number of model graphs.

In [17] the authors introduced a symmetry-breaking node equivalence for pruning the search space in backtracking algorithm for subgraph-matching problems, and also proved that backtracking algorithm for the monomorphism search problem (i.e., a general framework for subgraph matching) which is that its complexity equals the number of one-to-one function between query and data graph vertices.

The subgraph-matching problem in multi-graphs is relatively new. Ingalalli et al. [18] presented SuMGra, a feature-based indexing method that supports subgraph matching in a multi-graph. SuMGra mapped nodes to a six-features vector such that one of them is a structural feature and the remaining features are information of multi edges. SuMGra uses a high-dimension R-tree index for indexing feature vectors and Ordered Trie with Inverted List for multi indexing edges of nodes, which use file techniques.

Subgraph matching has many applications in different fields such as network analysis [19], RDF query processing [20], cheminformatics [21], bioinformatics [22,23], and malware detection [24]. Moreover, the problem is used as a black box or subroutine in finding frequent subgraphs [1,25], so the problem is a partner with the frequent subgraph mining problem in applications of finding frequent subgraphs such as analysis and understanding of complex networks, finding motifs and graphlets [26], and biology [27]. Moreover finding frequent subgraph is useful in classification [28].

This paper proposes a node embedding-based solution for multi-graph matching. The proposed model is composed of two stages. The first step's primary concept is that the density-like neighborhood structure is found in the data graph for the densest extracted vertex in the query graph to obtain the densest subgraph, then the k-nearest neighborhood query is found. For each layer graph, the second step, mapping the vertex to the feature vector (Vertex Embedding), improves the model proposed. The principal component analysis (PCA) approach is used to minimize the node embedding size to be effective with the KD-tree indexing. To eliminate the redundancy in the generated pattern matching the query graph, symmetry breaking conditions will also be used. The filtering method is implemented to minimize the number of candidate data nodes of the initiate query vertex. Finally, the effect of the concatenation of the structural features (orbits features) with the meta-features (summary of general, statistical, information-theoretic, etc.) for signatures of

nodes on the model performance is tested. The proposed method guarantees that when a query graph has a match in the data graph, the candidate set has at least one vertex arrive at a solution. Using symmetry breaking helps to generate all distinct embeddings of the query graph.

The rest of this paper is organized as follows: Theoretical background and steps of the proposed model are presented in Section 2 and 3, respectively. Experimental scenarios and discussions are introduced in Section 4. Finally, conclusions and future work are presented in Section 5.

2. Theoretical Background

Firstly, it is assumed that all graphs are unweighted (i.e., edges have no weight) and loop-free (which means no edges from any vertex to itself).

Definition 1. *Unlabelled, Undirected Simple Graph. An unlabelled undirected simple graph is a pair (V, E) such that V is the set of vertices and E is the set of undirected edges,* $E \subseteq \binom{|V|}{2}$ *edges.*

A simple graph is a very simple concept in graph theory, but it is useful as a model of some problems.

Definition 2. *Unlabelled, Undirected Multigraph Graph. An unlabelled, undirected multi-graph G is a tuple of four parts (V, E,* L_G, S_G*) where V is the set of vertices, E is the set of undirected edges,* $E \subseteq V \times V$, S_G *is the set of labels and* $L_G : V \times V \rightarrow 2^{S_G}$, *such that* 2^{S_G} *is the power set of* S_G.

The multi-graph is the more effective way to present real interaction systems because it has many relationships between system objects, giving it a lot of information to look at valuable patterns.

Furthermore, the multi-graph is a simple concept in graph theory and it is generalized of a simple graph. However, it is very powerful, interesting, and useful as the model of a lot of problems. A data multi-graph G is shown in Figure 1.

One of the most important features is the graph core number for each vertex. To calculate the graph core number for each vertex the multigraph will be transformed into the simplified graph.

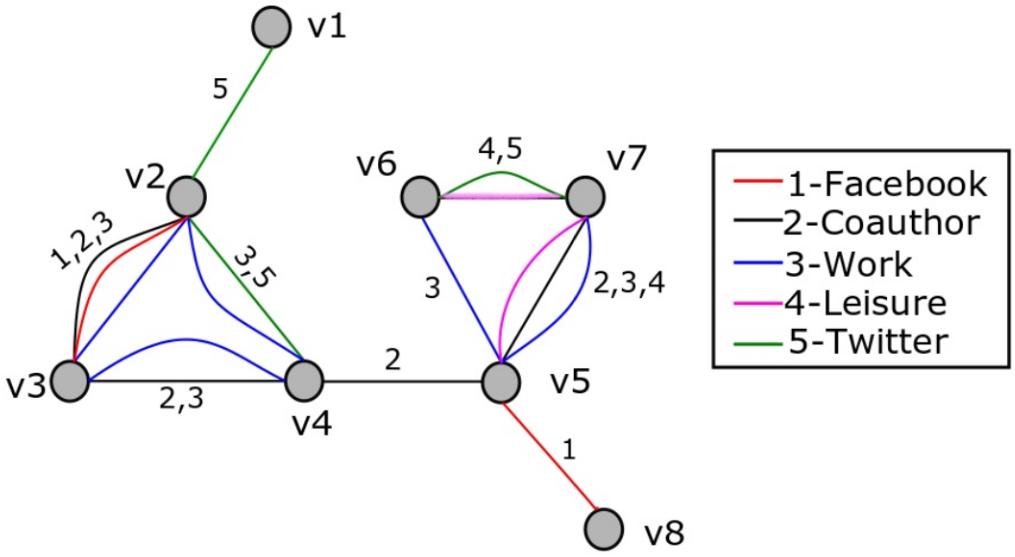


Figure 1. A data multi-graph G.

Definition 3. $V(G)$ *is the set of vertices of (multi)graph G,* $E(G)$ *is the set of edges of (multi)graph G and* $deg(v) = |N(v)|$. *Max degree is denoted by d.*

Definition 4. *Neighboring Graph. Given an unlabelled undirected simple graph H, the set of a neighbor of a vertex v :* $N(v) = \{u \in V(H) \mid (v,u) \in E(H)\}$.

Definition 5. *Simple Subgraph. A simple subgraph h of H is a graph such that* $V(h) \subseteq V(H)$ *and* $E(h) \subseteq E(H)$.

Definition 6. *K-core. Given an unlabelled, undirected simple graph H and an integer k, the K-core of H is the maximal sub-graph h of H such that the minimum degree of h is at least k. That is, every vertex in h is connected to at least k other vertices in h (i.e.,* $deg(v) \geq k, \forall v \in V(H)$).

Definition 7. *Core Number. Given the unlabelled, undirected simple graph H, the core number of vertex v in H, denoted core(v), is the largest k such that the K-core of H contains v.*

Definition 8. *Simplified Graph. Given an unlabelled, undirected multigraph Graph G = (V, E,* L_G, S_G*). Then, the simplified graph* $\Sigma(G) = (V_s, E_s)$ *such that* $V_s = V$ *and* $E_s = E$*, means the simplified graph is the multigraph without multi-edges as in Figure 2.*

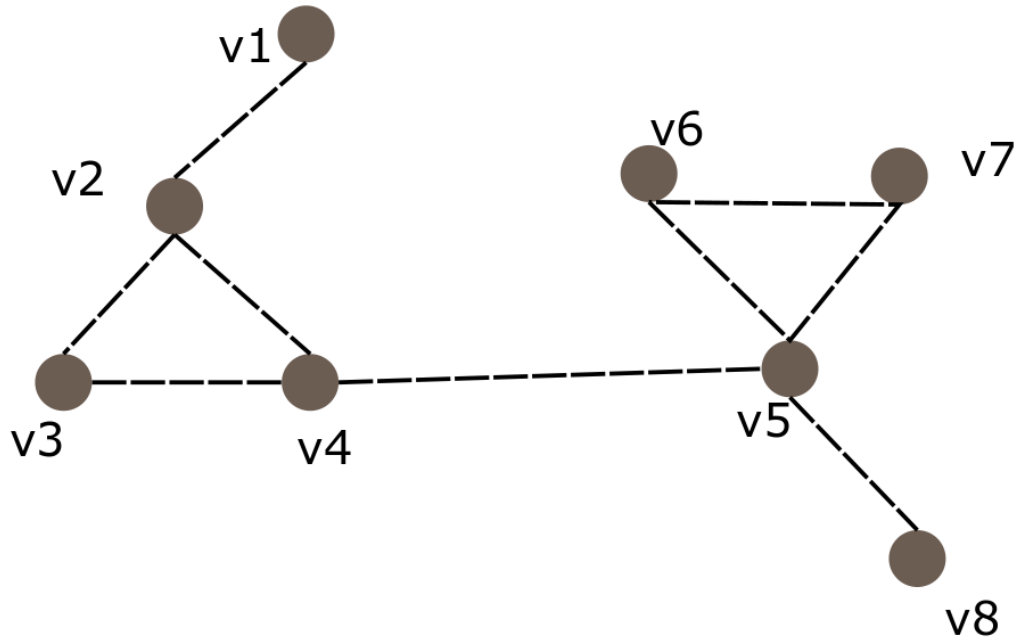


Figure 2. A Simplified graph $\Sigma(G)$.

The simplified graph is the multi-graph after replacing all multi-edges with one edge, indicating the existence of relations between these vertices regardless of the exact number of edges and their relationships. The simplified graph is used to obtain the core number.

The core number is used to build features of vertices. By definition of core number, observe that core number is the dense measure of a vertex. If the core number is high, then the vertex will be with high density. In the following sections, we will know the importance and how to compute it by an efficient algorithm. For example, the core number of $V(Q1)$ of a simplified graph $\Sigma(Q1)$, a multi-graph Q1 is shown in Table 1 which is extracted from Figure 3.

Table 1. Core number and vertex signatures for the multigraph in Figure 3.

u_i	Core (u_i)	ω (u)
u_1	1	{{5}}
u_2	2	{{2},{5},{2,4}}
u_3	2	{{2},{2,3}}
u_4	2	{{2,3},{2,4}}

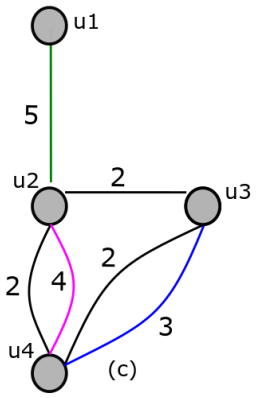


Figure 3. A query multigraph Q1.

Definition 9. *Triangle number of the vertex. Given a simple unlabeled graph, the triangle number of v(i.e., tr(v))is the number of triangles that v is in.*

Remark 1. *core(v)* $\leq$ *deg(v).*

Definition 10. *Multigraph homomorphism. Given a multigraph* $H = (V_H, E_H, L_H, S_H)$ *and a multigraph* $G = (V_G, E_G, L_G, S_G)$*, the subgraph isomorphism from Q and G is a function* $\Phi : V_H \rightarrow V_G$ *such that :* $\forall (u_i, u_j) \in E_H$*, then* $(\Phi(u_i), \Phi(u_j)) \in E_G$*.*

The multigraph homomorphism is computed implicitly in the proposed model

Definition 11. *Multigraph isomorphsim. Given a multigraph* $H = (V_H, E_H, L_H, S_H)$ *and a multigraph* $G = (V_G, E_G, L_G, S_G)$*, the subgraph isomorphism from Q and G is a bijection function (i.e., one-to-one and onto function)* $\Phi : V_H \rightarrow V_G$ *such that :* $\forall (u_i, u_j) \in E_H$ *iff* $(\Phi(u_i), \Phi(u_j)) \in E_G$*.*

Definition 12. *Multigraph automorphism. Given a multigraph* $G = (V_G, E_G, L_G, S_G)$*, the subgraph isomorphism from Q and G is a bijection function (i.e., one-to-one and onto function)* $\Phi : V_G \rightarrow V_G$ *such that :* $\forall (u_i, u_j) \in E_G$ *iff* $(\Phi(u_i), \Phi(u_j)) \in E_G$*.*

Remark 2. *Given* $u,v \in V(G)$*. Define an equivalence relation over V(G),* $\exists$ *an automorphism such that* $\Phi(u) = v$*, then u and v in a equivalent class. The equivalence classes are called orbits.*

Definition 13. *Subgraph isomorphism for multigraphs. Given a query multigraph* $Q = (V_Q, E_Q, L_Q, S_Q)$ *and a data multigraph* $G = (V, E, L_G, S_G)$*, the subgraph isomorphism from Q and G is an injective function (i.e., one-to-one function)* $\Phi : V_Q \rightarrow V$ *such that :* $\forall (u_i, u_j) \in E_Q, \exists (\Phi(u_i), \Phi(u_j)) \in E, L_Q(u_i, u_j) \subseteq L_G(\Phi(u_i), \Phi(u_j)) \forall i \neq j$*.*

Definition 14. *Vertex Signature. The vertex signature of vertex v is a multiset contains labels of multi edges that are in the incident on v. Mathematically,* $\omega(u) = \bigcup_{v \in N(u)} L(u, v)$*.*

Remark 3. *By definition of vertex signature, observe that deg(u) equals the number of sets in* $\omega(u)$ *(i.e., Cardinality of the vertex signature).*

For instance, in Q1, $\omega(u2)$ = {{2},{5},{2,4}}, all vertex signatures of vertices of the multigraph of Figure 3 are depicted in Table 1.

Definition 15. *Candidate set. Given a graph Q, then the candidate set of vertex u C(u) is defined as* $C(u) = \{v \in V(G) | \ \omega(u) \Subset \omega(v)\}$*, such that* $\Subset$ *is the subset operation on a multiset. Define* $\Subset$ *operation as a function F from* $\omega(u)$ *to* $\omega(v)$ *as an injective function (i.e., one-to-one),* $F(x) = y$*,* $x \subseteq y$ *and* $\forall\ v \in C(u)$ *is similar to u.*

The candidate set is an essential concept in many algorithms and techniques. It is like a starting point for algorithms to obtain solutions. The difficulty is in obtaining a candidate set and computing it efficiently. The next sections will give more details of importance, use, and obtaining candidate sets in the algorithm.

3. Subquery Matching in Multigraph

In this section, the general framework of the proposed algorithm is presented. The proposed algorithm consists of two phases: indexing for store feature vectors and multi-edges (Section 3.4). Subgraph search space to enumerate all functions(i.e., embeddings) (Section 3.5). In the two phases, K-core, core number, and vertex orbits counting are the basic tools (Subsection 3.3).

3.1. Problem Definition (Sub-Multigraph Query Matching)

Given a connected query multi-graph Q such that the set of vertices of Q ($|V(Q)|$) ≥ 3, data multi-graph G such that $|V(G)| \geq |V(Q)|$, the sub-multigraph query matching problem is to enumerate all the embeddings of Q in G, so that each embedding is isomorphic to the query Q. Mathematically, sub-multigraph query matching is to enumerate all possible functions Φ from Q to G. Given G is shown in Figure 1 and Q2 is shown in Figure 4, Algorithm 1 has to enumerate all possible embeddings (i.e., enumerate all possible functions Φ) of Q2 to G. All embeddings are as follows:

$$Match1 := \{(u1, v4), (u2, v5), (u3, v6), (u4, v7)\}.$$

Match1 is a set of ordered pairs such that the first coordinate is the query vertex u_i and the second coordinate is the data vertex v_j, which is the image of u_i. Q2 in Figure 4 has one embedding in G. If the query multi-graph has at least two embeddings in G then all embeddings are isomorphic to each other.

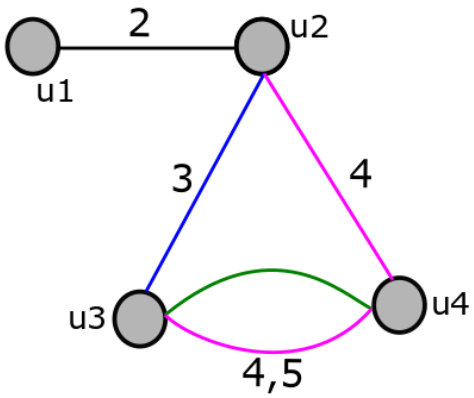


Figure 4. A query multigraph Q2.

Algorithm 1 : The proposed algorithm framework

1: Building off-line index KD-tree
2: u_{init} = GetInitiate($V(Q)$)
3: U = OrderQueryVertices($V(Q)$)
4: $C_{u_{init}}$ = SelectCandidate(u_{init})
5: All Matching = $\emptyset$
6: **for all** v $\in C_{u_{init}}$ **do**
7: $Matched_q = u_{init}$
8: $Matched_g$ = v
9: Matched List = [$Matched_q$, $Matched_g$]
10: All Matching := RoutineSubgraphBacktracking (All Matching, Matched List, Q, G)
11: **end for**

3.2. Overview of the Proposed Algorithm

Generally, in the proposed Algorithm 1 the following steps are performed. Firstly, build index KD-tree off-line (Line 1). Find initiate query vertex (u_{init}) through GetInitate function by ordering query vertices using the proposed order (Section 3.5). In (Line 3), order all query vertices by effective order, representing U to use it in a subgraph search. Use index KD-tree to find candidate set $C_{u_{init}}$ that matches for u_{init} (Line 4). Matching starts with sub-match or partial matching by match u_{init} with v in $C_{u_{init}}$ and calling RoutineSubgraphBacktracking function (Sections 3.4 and 3.5) to match remain query vertices (Lines 6–10).

3.3. K-Core, Core Number and Vertex Orbits Counting

A graph's k-core has interesting properties: applications such as community search, locating influential, keyword extraction from text, link spam detection, real-time story identification, dense subgraphs, and clustering. The k-core of a graph is the maximal subgraph with a minimum degree of at least k (k-core is well-defined) [29]. It is readily demonstrated that this subgraph is unique by contradiction (i.e., maximal propriety). The maximum k such that G has a k-core, which is the maximum core number of G. There are a lot of algorithms to compute and obtain a k-core graph and core number of vertices [29]. The core number of vertices, can be found in polynomial time ($\mathcal{O}(n^2)$), but can actually be obtained in $\mathcal{O}(\mathrm{m})$ time such that $n = |V(G)|$ and $m = |E(G)|$, by modifications on bin sorting in implementation (where $\mathcal{O}(.)$ refers to the time complexity). However, the algorithm in [30] is embedded in the Algorithm 2 which has an $\mathcal{O}(m)$ complexity.

Peeling technique can be performed with a list (or array) linear heap data structure or CoreD-Local (or CoreD-Local-opt) algorithm with h-index in $\mathcal{O}(mh - index(G))$ [31]. The peeling algorithm's idea attractively removes (i.e., peels) the minimum-degree node of a graph. The step of obtaining min from the set of nodes which are not visited nodes by the linear heap data structure can be achieved in constant time $\mathcal{O}(1)$. Algorithm 2 is the fastest one applied to the suggested datasets. So, the Algorithm 2 is used to acquire the core number feature for each vertex in the data and query graph. The relation between core number and degree of a vertex is not necessarily positive correlation (or correlated) (for example, the star graph is a simple graph of order n such that each vertex has degree equals 1 except that one of them has degree equal to $n - 1$. Then, the core number of each vertex is $1, \ldots$). However, this property is satisfied in some datasets.

Vertex orbits counting in simple graph, given orbit θ and vertex v, $\theta_i(v)$ = number of occurrences of v in orbit θ_i as subgraph. $\Theta_i(v)$ = number of occurrences of v in orbit θ_i as induced subgraph such that $i \in \{0, 1, 2, \ldots, 72\}$ [32].

Algorithm 2 : The Core Number Algorithm

```
1: Compute the degrees of vertices, deg.
2: order the set of vertices V(G) in increasing order of their degrees.
3: for all v in V(G) in the order do
4:     core[v] = deg[v].
5: end for
6: for all u in N(v): Neighbour of vertex v do
7:     if deg[u] > deg[v] then
8:         deg[u] = deg[u] − 1.
9:         reorder V(G).
10:    end if
11: end for
12: Return the list of core numbers
```

3.4. Indexing

This subsection shows how to use indices (i.e., KD-tree and set-trie) and features usefully in the proposed algorithm. It also details how to compute them effectively. The indexing is split into two phases, offline and on-line indexing, as follows:

1. offline indexing: In this phase, we compute the useful and efficient features and build indices on data graph G (i.e., indexing feature vectors of data vertices and multi edges).
2. On-line indexing: In this phase, using indices to find candidate sets for matching query Q in graph G.

Firstly, our criteria are choosing the features such that they are not constant in each vertex of the data graph for at most cases of a graph, distinct features, and no two different features are equivalent. Secondly, splitting the features into two kinds of density features and similarity features. They assume that density features have more priority than similarity features to reach into the dense subgraph before obtaining similar data vertices. By the word "dense", we mean roughly that the subgraph contains a many edges and it is well connected.

Surely, there exist features that will be the density and similarity feature.

3.4.1. Off-Line KD-Tree Index

Mapping each vertex of the data graph into D-vector (D = 8) space and each coordinate corresponds to the features for reducing the sub-problem of graph mining to the geometric problem. Mathematically, ζ : V(G)$\cup$ $\{u_{init}\} \rightarrow \mathbb{N}^8$. Table 2 shows features and its descriptions.

Table 2. Features that were used in our proposed algorithm.

Feature f_i	Description of Feature
f_1	Core number
f_2	Cardinality of vertex signature (i.e., degree)
f_3	Triangle number
f_4	The number of unique edge types in the vertex signature
f_5	The number of all occurrences of the edge types (repetition allowed)
f_6	Minimum index value of the edge type alphabet (position of the sequenced alphabet)
f_7	Maximum index value of the edge type alphabet (position of the sequenced alphabet)
f_8	Maximum cardinality of the vertex sub-signature

The density features f_1, f_2, and f_3 are used to reach the dense subgraph. The similarity features f_i such that $4 \leq i \leq 8$ are used to obtain the similar vertices given query vertex in the query graph Q. In other words, f_1, f_2 and f_3 are for density and the remaining features are for similarity. Constructing the KD-tree index by organizing the information supplied by some efficient features are in Table 2, the feature vectors of vertices of data graph are shown in Table 3.

To compute the core number for each vertex in the data graph, we have to construct the simplified graph $\Sigma(G)$ and then compute the core number for each vertex in $\Sigma(G)$. The first feature, f_1, is a core number used to obtain the dense vertices (i.e., have a lot of relations) to reach dense subgraph to search in it. The core number feature is computed by the Algorithm 2 in $\mathcal{O}(m)$. The triangle number feature f_3 was used to obtain more dense vertices and symmetry area, computing it for all vertices by the algorithm in [32]. The remaining features can be computed directly (by using graph representation adjacent list). We keep on inserting data graph feature 8-vectors into the KD-tree. Then the offline index is built.

Table 3. Feature vectors of data vertices of data graph Figure 1.

Data Vertices	**Features**							
u_i	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8
u_1	1	1	0	1	1	5	5	1
u_2	2	3	1	4	5	1	5	3
u_3	2	2	1	3	5	1	3	3
u_4	2	3	1	3	5	2	5	2
u_5	2	4	1	4	6	1	4	3
u_6	2	2	1	3	3	3	5	2
u_7	2	2	1	4	5	2	5	3
u_8	1	1	0	1	1	1	1	1

3.4.2. On-Line KD-Tree Index Query

Given G, which is a multi-graph (Definition 2), and given Q, which is a query graph, finding candidate sets in the proposed algorithm is very critical and important for matching a query graph Q in graph G. In (Line 4) in Algorithm 1. $C_{u_{init}}$ is the candidate set for matching the initiate query vertex u_{init}, which is found by querying the KD-tree index (Section 3.4.1).

KD-tree index solves many problems such as the nearest neighborhood, k-nearest neighborhood, range query problems, and many geometric problems.

Finding candidate sets is a problem approaching the k-nearest neighborhood problem as the solution for computing candidates. K is used as a parameter in the method. Its value changes according to a dataset and query graph (i.e., clique or non-clique graph). This parameter gives flexibility for finding candidates. The challenge is finding dense area using feature-based in sparse (i.e., $|E_s(\mathrm{Q})| = \mathcal{O}(|V_s(Q)|)$) query graphs, vice versa when query graph is dense (i.e., $|E_s(\mathrm{Q})| = \mathcal{O}(|V_s(Q)|^2)$).

In on-line indexing, we query with query vertex (i.e., initiate query vertex u_{init}) of query graph Q to obtain candidate set $C_{u_{init}}$.

The solution of k-nearest neighborhood equals the first version of the candidate set of given query vertex. Then we can compute the candidate set by the effective way by this reduction. Before finding $C_{u_{init}}$ it is necessary to find u_{init} (Section 3.5).

3.4.3. On-Line and Off-Line Set-Trie Index

This subsection aims for finding candidate sets for rest query vertices to match it in the graph G. In Sections 3.4.1 and 3.4.2 responsible for finding candidate set for u_{init}, using KD-tree index.

Definition 16. *SuperSetQuery (v,{x,y}). It is a function that has two parameters, namely a data vertex v and a multi-edge between x and y {x,y}. The data vertex v is used to construct a set-trie index for its neighborhood subgraph structure. The multi-edge between x and y {x,y} from E_Q of query graph as a query. This function returns all data vertices which are in N(v) such that the multi-edges between them and v (i.e., $\omega(v)$) are a superset of query multi-edge {x,y}.*

The set-trie index helps to find all possible candidates for the rest of query vertices. During recursion of Algorithm 3 it refines this set of all possible candidates to another set for reducing search space in Algorithm 4 .

A set-trie is a tree data structure similar to an ordinary trie data structure. It builds the set-trie, such as building the ordinary trie data structure [33]. In the proposed algorithm, building a set-trie of multiedges for a data vertex (i.e., vertex signature $\omega(v_i)$), for instance v_5, is $\omega(v_5)$ = {{1},{2},{3},{2,3,4}}, its neighbourhood subgraph of v_5 is shown in Figure 5a and its set-trie is shown in Figure 5b. For finding all possible candidates, assume building a set-trie index for some data vertex such as v_5 and querying in the index by a multi-edge such as the multiedge between u2 and u3 (i.e., {u2,u3}). We then have the output set of candidate data vertices for u_{nxt}. This set satisfies SuperSetQuery and its output is {v6,v7}.

Moreover, set operations operated using a succinct data structure which is a compressed bitmap, and for datasets that have at most 32 kinds of relations, bitmasks are used.

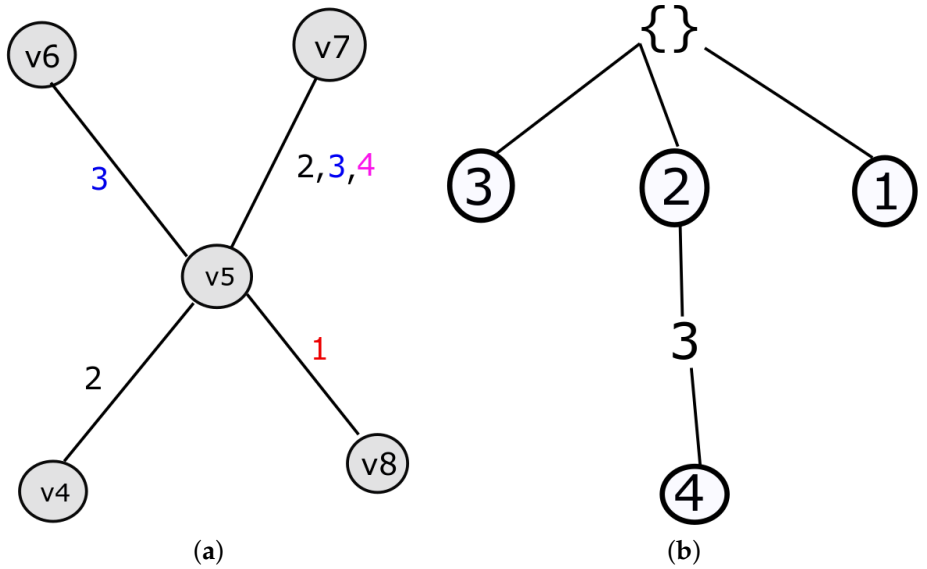


Figure 5. Building set-trie Index for data vertex v_5. (**a**) Neighbourhood subgraph structure of v_5, (**b**) set-trie index representation for v_5.

Algorithm 3 : Routine Subgraph Backtracking

```
Get u_nxt
MatchCandidateVertices = FindJoinable (Matched_q , Matched_g , u_nxt)
for all v_nxt ∈ Match candidate vertices do
    Matched_q.add(u_nxt)
    Matched_g.add(v_nxt)
    Matched List = [Matched_q , Matched_g] /*such that Mathced_g didnot matched, without this condition the algorithm compute homomorphism (see Definition 9). */
    Routine Subgraph Backtracking (All Matching, Matched List, Q, G)
    if All Matching.size == Q.order then
        All Matching.union(Matched List)
    end if
    Matched_q.pop()
    Matched_g.pop()
end for
return All Matching
```

Algorithm 4 : RoutineFindJoinable

```
MatchedNeibor_q = Mathced_q ∩ adj(u_nxt)
MatchedNeibor_g := Corresponding Matched vertices of MatchedNeibor_q
PossibleCandidateVertices = ⋂_i SuperSubSetQuery(MatchedNeibor_g[i], {MatchedNeibor_q[i] , u_nxt})
for all v ∈ PossibleCandidateVertices do
    if ω(u_nxt) ∈ ω(v) then
        MatchCandiateVertices.add(v)
    end if
end for
return MatchCandiateVertices
```

3.5. Subgraph Query

This section presents the method of finding embeddings(i.e., functions from query and data graph) of query multi-graph, preserving connection and structure of Q and ordering.

3.5.1. Finding Elegant Initiate Query Vertex

Definition 17. *Elegant vertex. An elegant vertex is in a dense area (or subgraph).*

Finding an elegant initiate query vertex is important for finding the candidate set in Line 4 of Algorithm 1. We order V(Q) in decreasing order of three score functions.

1. Core number, computing core number over $V(Q)$ efficiently by Algorithm 2, $score_1(u)$ = core(u), $\forall u \in V(Q)$.
2. Degree of $V(Q)$ (i.e., number of incident edges) in $\Sigma(Q)$, $score_2(u)$ = deg(u), $\forall u \in V(Q)$.
3. Triangle number, $score_3(u)$ = tr(u), $\forall u \in V(Q)$.

Assume that the score priorities in ascending order are $score_3$, $score_2$, $score_1$ (i.e., ordering according by $score_1$ and if two query nodes have the same $score_1$ then ordering according by $score_2$). The initiate query vertex u_{init} will be the top of the order. In $Q2$ as showed in Figure 4, the Table 4 shows the core number and degree of query vertices of $Q2$. After applying the previous method, finding the total order of $V(Q2)$ is $[u2, u3, u4, u1]$, then the elegant vertex of $V(Q2)$ is $u2$ which is u_{init}.

Table 4. Core number, degree and triangle number of query vertices of query graph Figure 4.

Query Vertices u_i	Core (u_i)	Deg (u_i)	Tr (u_i)
u_1	1	1	0
u_2	2	3	1
u_3	2	2	1
u_4	2	2	1

Lemma 1. *The initiate query vertex u_{init} is elegant.*

Proof. The proof is obtained in the previous method of obtaining u_{init}. □

3.5.2. Query Vertex Ordering

The order which is in line 3 in Algorithm 1 is very critical because it is used in query searching to obtain u_{nxt} (Line 1 in Algorithm 3) which is the vertex that should be matched.

Define the total order $\prec_\sigma$ over $V(Q)$ such that it satisfies the condition: Assume u, v, x and $y \in V(Q)$ such that for every 3-tuple (u,v,x) if u $\prec_\sigma$ v $\prec_\sigma$ x , {u,v} $\notin$ E(Q) and {u,x} $\in$ E(Q) there exists y such that y $\prec_\sigma$ u and {u,y} $\in$ E(Q). When expanding a vertex, sort its not-visited adjacent vertices in decreasing order by three score function $score_1(u)$, $score_2(u)$, $score_3(u)$.

After obtaining u_{init}, apply the total order $\prec_\sigma$ using it. In Q2 as shown in Figure 4, after applying the total order $\prec_\sigma$ obtaining $\sigma = [u2, u4, u3, u1]$. Score functions depend on the structure of the subgraph (or graph) to reach dense and similar subgraphs.

3.5.3. Candidates for Initial Query Vertex

In Section 3.4 we showed on-line indexing and some of the details for finding. This subsection shows the remaining details of finding $C_{u_{init}}$. First, selecting candidate set $C_{u_{init}}$ given initial query vertex u_{init} and then start matching and perform the Subgraph Search. By the definition of the candidate set, candidate set has to satisfy the following two conditions:

1. $C(u) = \{v \in V(G) | \omega(u) \Subset \omega(v)\}$.
2. $\forall v \in C(u)$ is similar to u.

To satisfy the first condition and save the time of subset operation on multiset computation, using the necessary condition "candidate set has all of the data vertices except f_6 feature that has features greater than features measures u_{ini}". Mathematically, $ALL(u_{init}) := \{v \in V(G) | i \neq 6, F_{u_{init}(i)} \leq F_{v(i)}\}$.

To satisfy the second condition (i.e., similarity condition), we define the similarity set of u_{init} as follows: $Sim(u_{init}) := \{v \in V(G) | u_{init} \approx_{sim} v\}$. Now we compute $Sim(u_{init})$ using the k-nearest neighborhood query in KD-tree. This condition is a refinement for the candidate set. Now we find the candidate set by merging two queries (i.e., range and k-nearest neighborhood), find the 7-upper similar query using the modified k-nearest neighborhood query, and use k as a parameter, as mentioned in Section 3.4.2. After computing and filtering $C_{u_{init}}$ such that it satisfies the two conditions, we sort it in decreasing order of the feature vectors, with decreasing priorities from f_1 to f_8, to get maximized priority candidate vertices.

Theorem 1. *SelectCandidate (u_{init}) function outputs at least one valid candidate vertex.*

Proof. The proof is obtained in the previous method in the Section 3.5.3. □

3.5.4. Subgraph Searching

In subgraph search Algorithm 3 there is the function responsible for finding or finishing the matching the query graph Q in the graph G, after matching u_{init} with the candidate vertex. Backtracking is used as a searching technique for this function. In this function, after the partial matching (Line 9 in Algorithm 1), obtain u_{nxt} for matching it (Line 1) by the total order $\prec_\sigma$. In (Line 2) we finding candidate data vertices for u_{nxt}. If v, which is in Line 6 of Algorithm 1, is a good matching for u_{init}, the backtracking grows the matching Q in G, else (i.e., v is not suitable matching for u_{init}), the backtracking reverts back and tries the next candidate vertex (Lines 3–13).

In the FindJoinable Algorithm 4 there is the function responsible for preserving the structure and connections of query graph Q. First, we find matched neighbor $MatchedNeibor_q$ of u_{nxt} and the corresponding matched vertices $MatchedNeighbor_g$ in a data graph (Lines 1–2). Second, a candidate for query vertices is found.

Now we Find possible candidate vertices for u_{nxt} (Line 3) by querying SuperSetQuery in set-trie index, building an index over multi-edges of neighbors of the $MatchedNeibor_g$[i] vertex, query multi-edge between $MatchedNeibor_q$[i] and u_{nxt} and use intersections operations over vertices that are output of SuperSetQuery. Now and after, we obtain the vertices to satisfy the structural connectivity of the query graph as in Lines 4–8 for reducing the number of candidates by checking vertex signature of v $\omega(v)$ (Line 4). Vertex signature of v is a superset (using $\Subset$ operation) vertex signature of $u_{nxt}(\omega(u_{nxt}))$.

To solve the problem that is the superset between $\omega(v)$ and $\omega(u_{nxt})$, we approach the problem as a maximum matching problem on a bipartite graph in $\mathcal{O}(|E|\sqrt{|V|})$ using [34] algorithm, such that E will be number of edges (i.e., number of subset or superset relations between the two partitions of bipartite graph) and V will be number of vertices (i.e., number of elements of $\omega(u_{nxt})$ and $\omega(v)$ which equals $(|\omega(u_{nxt}) + |\omega(v)|)$). To approximate the subset in multiset $\Subset$ usingthe embedding in Section 3.6 we just check that $emb_{unxt}[i] \leq emb_v[i]$ in O(1). RoutineFindJoinable guarantees that preserving the structure and connections of the query graph is proved by induction on order of vertices $V(Q)$ [18].

3.6. Improvement Algorithm by Vertex Features Using Structural and Meta Features and Symmetry Breaking

In this subsection, we apply the embedding vertex (i.e., mapping vertex to feature vector) to structural features for each layer graph (i.e., is simple graph induces one relationship) and meta-features for vertex signature (i.e., information about multi-edges such as the number of multi edges, statistics information, information-theoretic measures, etc.). The embedding will be useful for computing orbits (i.e., partitioning nodes such that

each partition has nodes with the same structure) in the graph (i.e., query or data graph), and orbits will be used in symmetry breaking.

3.6.1. Vertex Features Using Orbits and Meta Features

We embed vertices of the query and data graphs to structural and meta-features. Structural features are number of occurrences (induced or non-induced) of the vertex for all 73 orbits in [32] for each layer of the graph. These are concatenated; the algorithm uses induced occurrence. Table 5 shows the structural features of data nodes up to pattern order 3 of induced graph or layer of the graph of a working relationship.

Meta features are summaries (i.e., min, max, mean, ...) of general, statistical, information-theoretic, itemset, complexity, clustering features... using [35] for signatures of nodes. So, node embedding will be concatenation structural and meta features. Furthermore, the embedding preserve the order in Section 3.5.3 (i.e., v $\in C_{u_{init}}$ then for all dim. d $F_{u_{init}(d)} <= F_{v(d)}$), for example v_7 in Figure 6 can be a candidate of u_3 or u_4 in Figure 7 and v_2, v_4, v_5 and v_7 in Figure 6 can be candidates of u_1. However, node embedding size will be large and it is not efficient for KD-tree indexing for acquiring a candidate for u_{init}. So, using dimension reduction techniques such as PCA, embed node feature to 2D feature vector is expedient.

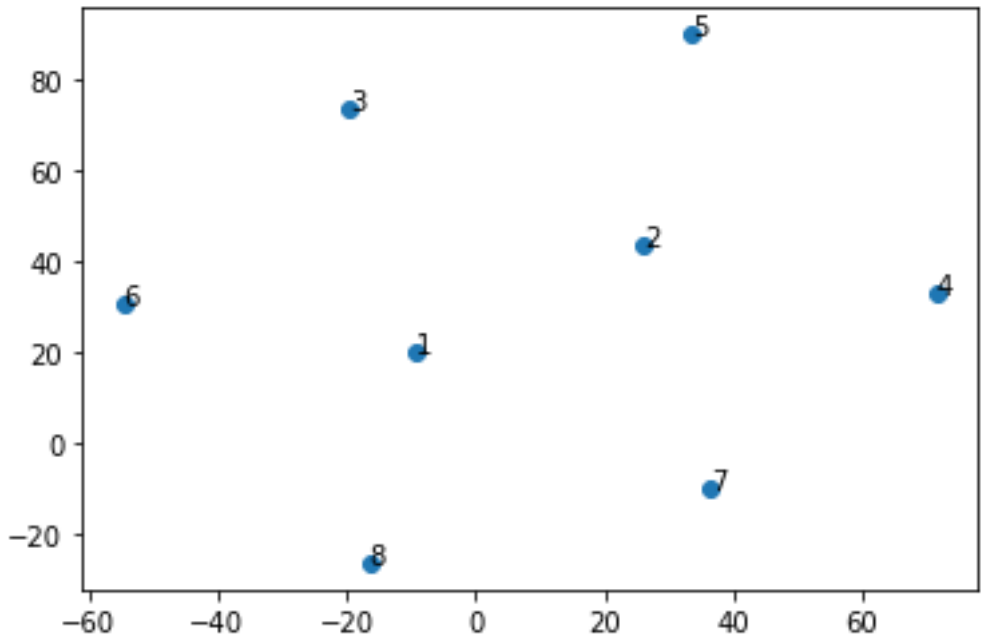


Figure 6. Embedding nodes of graph G in Figure 1 using features in Section 3.6.

Table 5. Vertex orbits of pattern order 3 counting of induced graph of relation 3 (i.e., work relationship) of data graph Figure 1.

v_i	Θ_0	Θ_1	Θ_2	Θ_3
v_1	0	0	0	0
v_2	2	0	0	1
v_3	2	0	0	1
v_4	2	0	0	1
v_5	2	0	1	0
v_6	1	1	0	0
v_7	1	1	0	0
v_8	0	0	0	0

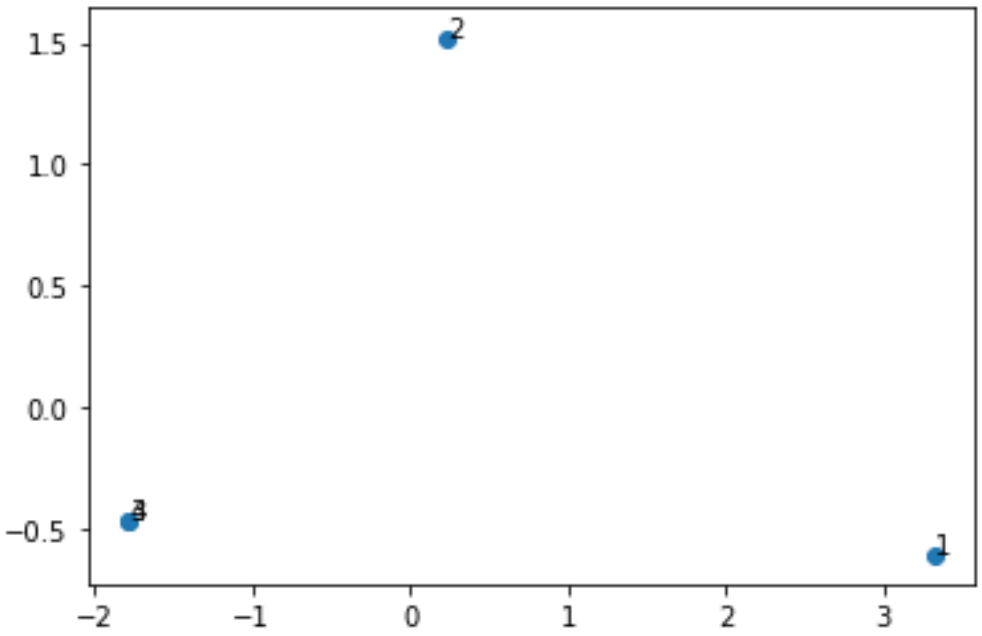


Figure 7. Embedding node of Q2, if $\omega(u_3) = \{\{4,5\},\{4\}\}$ and $\omega(u_4) = \{\{4,5\},\{4\}\}$.

3.6.2. Symmetry Breaking Condition

In Subgraph Backtracking Algorithm 4 will generate all pattern matches with the query graph but among generated patterns there exist repeated patterns or redundancy in patterns. So, symmetry breaking will solve the redundancy problem.

The graph's automorphism is a graph isomorphism between the graph and itself. Thus, graph automorphism generates all corresponding one-to-one mappings between the graph and itself, orbits of the graph which are a partition of vertices, and each partition has nodes which are permuted such that the graph still has the same structure. So, symmetry breaking breaks symmetry in each orbit. Then Algorithm 4 will generate all different matchings.

Computing orbits of the query graph, using the embedding in Section 3.6.1 preserves node orbits or structure and clustering embedding using any clustering algorithm such as k-means. Each cluster is an orbit with similar nodes. Moreover, after acquiring a candidate of u_{nxt} and before adding mapping in line 4 and 5 in Algorithm 4, we then check the condition (Symmetry Breaking Filter):

[if u_1 and $u_2 \in V(Q), u_1 \prec_\sigma u_2$ and are in the same orbit then $\Phi(u_1) < \Phi(u_2)$.]

Lemma 2. *Given a multi-graph Q, Aut(Q) let be the set of all automorphisms of Q. The Symmetry Breaking filter generates all symmetry breaking conditions.*

Proof. We need to prove that if u_i and $u_j \in V(Q)$ and they are in the same orbit, then $u_i \prec_{symmetrybreaking} u_j$ or vice-versa—that total order according to specific indexing of vertices. Symmetry Breaking filter generates that if u_i and $u_j \in V(Q)$ and they are in the same orbit and according of query vertex order then $u_i \prec_\sigma u_j$ or vice-versa, then $u_i \prec_{symmetrybreaking} u_j$ or vice-versa for all i,j $\in \{1,2,\ldots,|V(Q)|\}$. □

Theorem 2. *The Algorithm 1 with Symmetry Breaking filter generates all distinct embeddings of query graph Q in data graph G.*

Proof. Let O be an embedding of Q in G and assume that for all i, j, a, b $\in \{1,2,\ldots,|V(Q)|\}$, i $\neq$ j and a $\neq$ b, o_i and o_j are vertex indices (i.e., ids) in O that can be mapped to the same vertex $u_a \in V(Q)$. Since Φ is a one-to-one function, then there exists $u_b \in V(Q)$. Moreover there exists two automorphsim u_a and u_b are matched. Two possibilities exist: firts, (i) that the algorithm breaks. That means $u_a \prec_{symmetrybreaking} u_b$ or vice versa, in this case, using

Symmetry Breaking filter will break one of $o_i < o_j$ or $o_j < o_i$ and then the algorithm does not generate this mapping. (ii) the algorithm does not break. Then the algorithm generates mappings that contain $\{\ldots,(u_a,o_i)\ldots,(u_b,o_j),\ldots\}$ and $\{\ldots,(u_b,o_i)\ldots,(u_a,o_j),\ldots\}$ means that the automorphism maps u_a to u_b or vice-versa without symmetry break condition, from previous lemma. Then, by contradiction from case (ii), O will be mapped only once. □

In Figure 4 Q2, if $\omega(u_3) = \{\{4,5\},\{4\}\}$ and $\omega(u_4) = \{\{4,5\},\{4\}\}$, then orbits will be $\{\{u_3,u_4\},\{u_1\},\{u_2\}\}$. In Figure 7 u_3 and u_4 have the same embedding.

Moreover, after obtaining a candidate of the initial query vertex $C_{u_{init}}$ in Algorithm 1, it is refined using subset operation on multiset $\Subset$ in def 2.16.

4. Experimental Results and Discussions

In this section, the performance of the proposed algorithm on real and synthetic (random) multi-graphs is evaluated.

The evaluation is performed by comparing the proposed method with querying multi-graphs via an efficient indexing algorithm (SuMGra) [18]. The proposed model consists of two phases. The first phase's main idea is finding the densest query vertex, applying the filtering process to minimize the number of candidate data nodes of the initiate query vertex, finding the density similar neighborhood structure in the data graph, and finding the k-nearest neighborhood query to extract the densest subgraph.

The main idea of the second phase is mapping the vertex-to-feature vector (Vertex Embedding) for each layer graph, using a dimension reduction, principal component analysis (PCA) method to reduce the node embedding size to be efficient with the KD-tree indexing, using symmetry breaking condition to remove the redundancy in the generated pattern that matches with query graph, applying the improved filtering process to minimize the number of candidate data nodes of the initiate query vertex, and testing the effect of the concatenation of the structural features (orbits features) with the meta-features (summary of general, statistical, information-theoretic, etc.) for signatures of nodes on the model performance.

For ease of reading, we use the word "Proposed" to indicate the first phase of the proposed model, while we use "W/O" to indicate the second phase of the proposed without meta-features and "improvement" to indicate the second phase of the proposed method with the meta-features.

All the experiments were run on a PC, with Intel Inside CORE i3 processors 2.00 GHz, and 4 GB RAM, running on Windows10 OS. All algorithms in the paper have been implemented using python.

4.1. Description of Datasets and Query Subgraphs

In this research, experiments are executed over five datasets (three real multi-graph and two random multi-graph datasets). The three real multi-graph datasets are available at [36]. Dataset descriptions and some statistics are shown in Table 6. The genetic multi-graph HUMAN-HIV1 takes into account several genetic connections for biological organisms. These interactions include physical association, direct interaction, colocalization, association, and suppressive genetic interaction, which is determined by inequality. PLASMODIUM is a genetic multi-graph that concerns Plasmodium Falciparum and has relations such as Direct interaction, physical association, and association. EU-AIR is a TRANSPORTATION multi-graph, which is composed of thirty-seven different layers, each one corresponding to a different airline operating in Europe.

The two random multi-graph datasets are generated to test the efficiency of all algorithms. We generate query subgraphs with three types of random multi-graphs ordered from three to six nodes, and the following kinds of graphs: (i) a tree that is sparse (i.e., n − 1 edges) connected acyclic graphs; classes of graph such as paths and stars are also generated. (ii) Erdos-Renyi random graph with probability 0.5 to add edge which is dense (i.e., O(n^2))

graph. (iii) clique, which is a complete graph (i.e., every pair of a node is connected). All nodes of all types of graphs of signature sizes from 1 to 4 are randomly generated. For each multi-graph dataset, we generated 1000 samples for each kind. We report the average time for the first 1000 embedding for each query graph, and the queries which have no embedding are not counted in our experiment.

Synthetic data which are a random multi-graph are generated as a data graph that has 100 to 200 nodes and is a dense graph. We generated two kinds of synthetic data. The first has five relationships and the second has 10 relationships.

Table 6. Multigraph datasets description.

Dataset	Nodes	Edges	Dim
humanHIV1	1005	1355	5
plasmodium	1023	2521	3
EUAir	450	3588	37

4.2. Performance of the Proposed Algorithms

In this section, the results are presented using different metrics, such as the average time for all orders of query and 517 kinds of graphs, the average size of $C_{u_{init}}$ for all kinds of queries (i.e., trees, cliques, and random 518 graphs), and the embedding of a node feature in the plan (i.e., node embedding).

In the Figure 8 shows the avg size of C_{uinit} for Human HIV4, plasmodium, and EUAi datasets. Figure 9 shows the avg size of C_{uinit} for the two random multigraph random 1 and random 2 datasets.

The average size of C_{uinit}, in the second phase of the proposed algorithm gives improvement of the first phase with/without meta-features methods. Filtering is better than the proposed and SUMGRA. In Figure 8 SUMGRA the difference is remarkable, and in the first phase of the proposed algorithm the difference is remarkable as shown in in Figure 8.

Generally, the second phase of the proposed with/without meta-features methods works well about time and has meaningful embedding and robust filtering.

Figures 10–12 show the query time for Human HIV4, plasmodium, and EUAi datasets respectively.

Figures 10–12 show that in both random cliques and Erdos random graphs these queries are dense, so the best methods according to time are the second phase of the proposed with/without meta-features. However, in random trees, the first stage of the proposed method is the best according to time and that is due to this query is a sparse graph not dense so there is no need to find the symmetries. Finding the symmetries as proposed in with/without meta-feature in a sparse graph (i.e., not dense) will spend more time without result improvement.

Theoretically, if the query has a lot of symmetry, then the improvement with/without meta-features algorithms work well. Observably, improvement with/without meta-features methods work well in cliques and dense Erdos random graphs. The improvement method with meta-features works well on tree patterns except in Figure 11c, such that the proposed is the best in it. Generally, the first phase of the proposed algorithm is better than SUMGRA, and the second phase, which is the improvement with/without meta-features and methods, is better than the first phase of the proposed method.

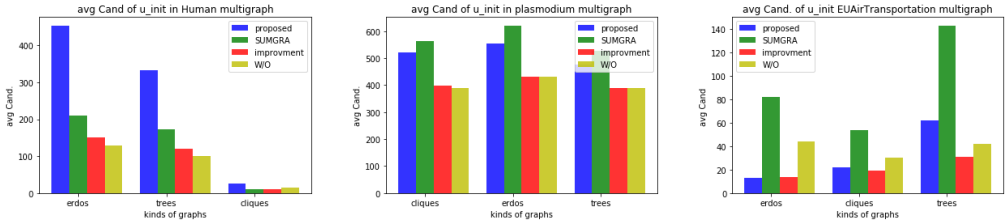
Figures 13 and 14 show the query time for the two random multigraph datasets random 1 and random 2 respectively.

It agrees with the theoretical study in both random cliques and Erdos random graph. These queries are dense, so the best methods are the second phase of the proposed with/without meta-features. However, despite the convergence of all results in the random trees, the second phase of the proposed meta-feature is the best one. Although the random trees query is sparse, the original data are generated as a dense graph.

Figures 15–17 show the node embedding Query for Human HIV4, plasmodium, and EUAi datasets respectively. In Figure 18 shows the node embedding Query for the two random multi-graph datasets.

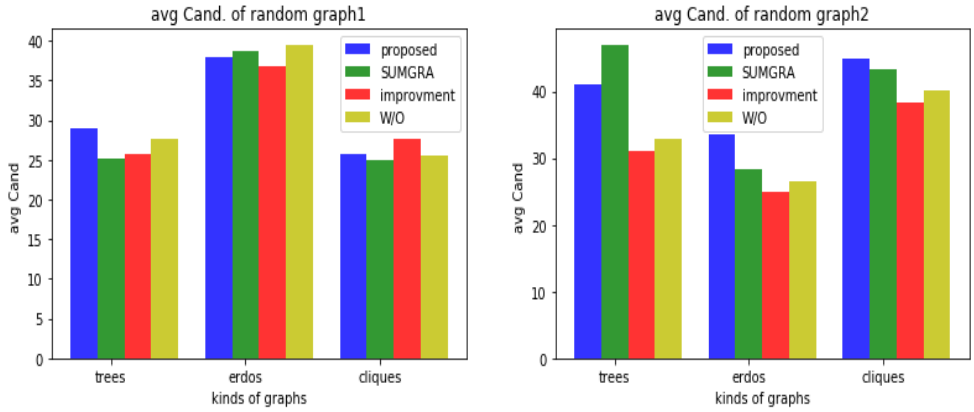
In the Figures 15–18 Principal Component Analysis (PCA) was used so, the x-axis and y-axis are represented the first and second principal component respectively.

In figures of node embedding such as Figure 16b or Figure 17b reflecting the order in Section 3.6, the order preserves structural and multi-edges information. On the contrary, the first phase of the proposed algorithm and SUMGRA algorithm preserve some properties in a simple version of multi-graphs such as degree, k-core number, and triangle number of nodes and not sufficient proprieties of multi edges. Furthermore, node embedding of random multi-graphs Figure 18 has no pattern- being dense in the center means nodes have the same dense properties.



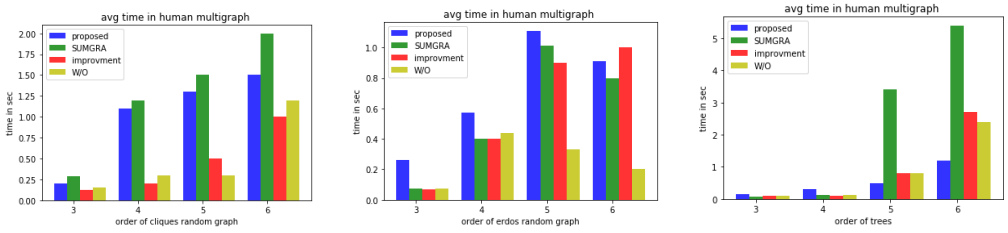
(**a**) (**b**) (**c**)

Figure 8. avg size of C_{uinit} on (**a**) humanHIV dataset, (**b**) plasmodium dataset and (**c**) EUAir dataset.



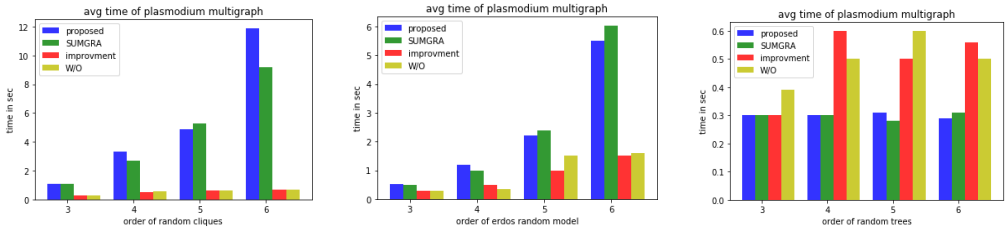
(**a**) (**b**)

Figure 9. avg size of C_{uinit} on (**a**) random 1 dataset, and (**b**) random 2 dataset.



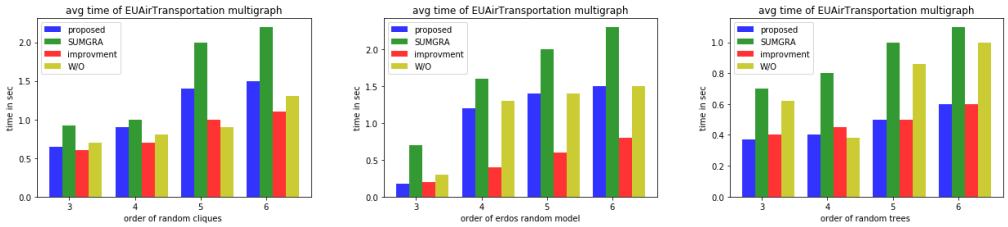
(**a**) random cliques (**b**) erdos random graph (**c**) random trees

Figure 10. Query time on Human HIV4 dataset.



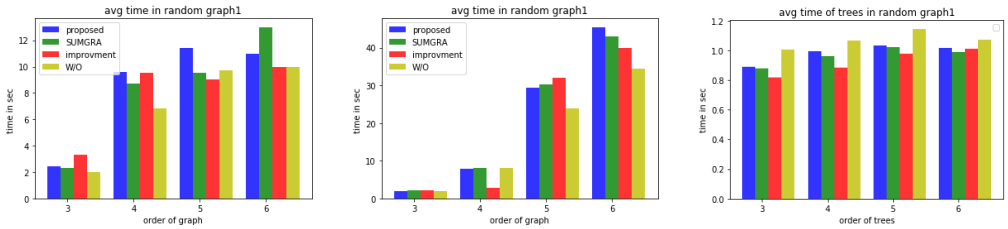
(**a**) random cliques (**b**) erdos random graph (**c**) random trees

Figure 11. Query time on plasmodium dataset.



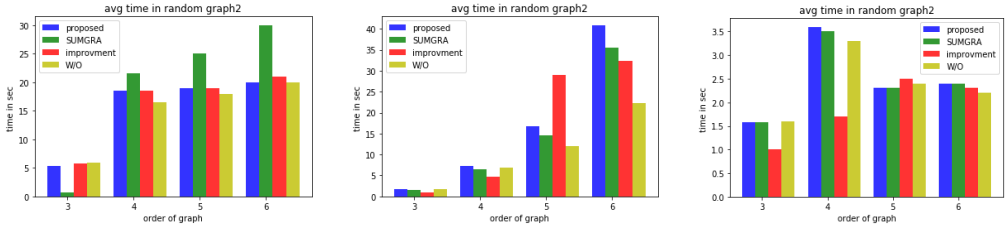
(**a**) random cliques (**b**) erdos random graph (**c**) random trees

Figure 12. Query time on EUAi dataset.



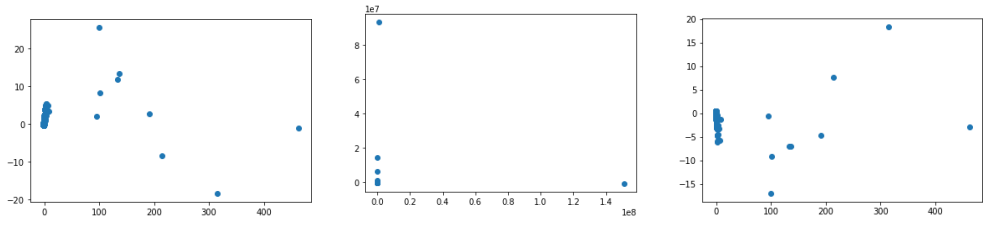
(**a**) random cliques (**b**) erdos random graph (**c**) random trees

Figure 13. Query time on random graph1 dataset.



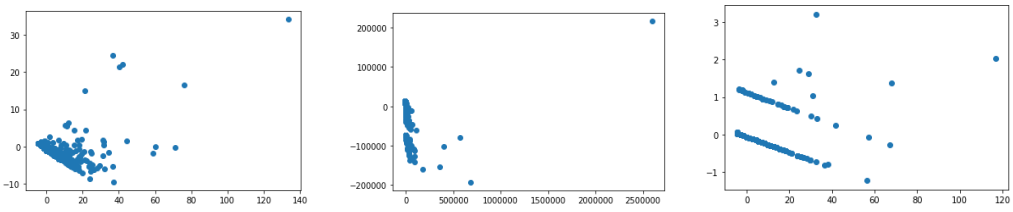
(**a**) random cliques (**b**) erdos random graph (**c**) random trees

Figure 14. Query time on random graph2 dataset.



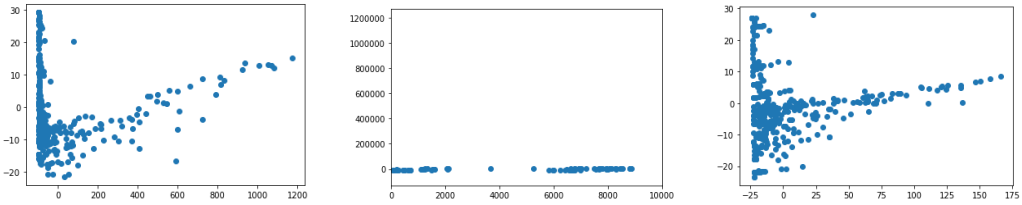
(**a**) node embedding of the first phase (**b**) node embedding of the second phase (**c**) node embedding of SUMGRA.

Figure 15. Node embedding of Human HIV4 dataset: (**a**) node embedding of the first phase of the proposed, (**b**) node embedding of the second phase of the proposed with/without meta-features, and (**c**) node embedding of the SUMGRA.



(**a**) node embedding of the first phase (**b**) node embedding of the second phase (**c**) node embedding of SUMGRA.

Figure 16. Node embedding of plasmodium dataset: (**a**) node embedding of the first phase of the proposed, (**b**) node embedding of the second phase of the proposed with/without meta-features, and (**c**) node embedding of the SUMGRA.



(**a**) node embedding of the first phase (**b**) node embedding of the second phase (**c**) node embedding of SUMGRA.

Figure 17. Node embedding of EUAi dataset: (**a**) node embedding of the first phase of the proposed, (**b**) node embedding of the second phase of the proposed with/without meta-features, and (**c**) node embedding of the SUMGRA.

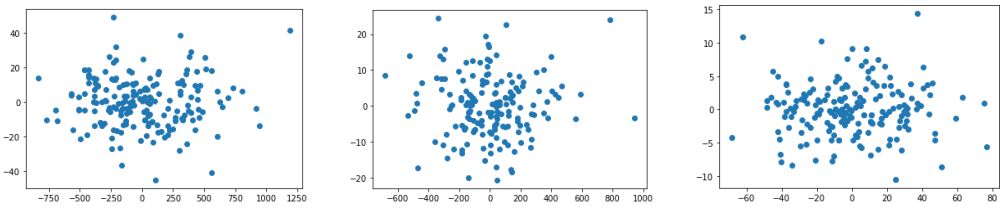


Figure 18. Node embedding of random graph with 5 and 10 relations dataset.

5. Conclusions and Future Work

This paper proposes a novel model for matching subgraph queries in multi-graphs based on features-based indexing techniques. The proposed model has two main phases. First, the densest extracted vertex in the query graph finds the density-similar neighborhood structure in the data graph. It is followed by finding the k-nearest neighborhood query to acquire the densest subgraph. In the second phase, and for each graph layer, mapping the vertex-to-feature vector reduces the node embedding size to be efficient with the KD-tree indexing a dimension reduction. Symmetry breaking conditions were used to remove the redundancy in the generated pattern matching with the query graph. They test the effect of the structural features' concatenation with the meta-features for signatures of nodes on the model performance. In both phases, the filtering process is applied to minimize the number of candidate data nodes of the initiate query vertex. The first phase of the proposed algorithm is better than SUMGRA, and the second phase, which is the improvement with/without meta-features and methods, is better than the first phase of the proposed method. A promising future work would be to learn features instead of feature engineering by graph neural network.

Author Contributions: Conceptualization, A.E.H. and S.H.B.; supervision, A.E.H. and S.H.B.; methodology, M.A.; validation, M.A.; formal analysis, M.A. and V.S.; writing—original draft preparation, M.A., A.E.H., V.S. and S.H.B.; writing—review and editing, M.A., A.E.H., V.S. and S.H.B. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: In this research, experiments are executed over five datasets (three real multi-graph and two random multi-graph datasets). The three real multi-graph datasets are available at Domenico, M.D. Complex Multilayer Networks Lab at FBK. https://manliodedomenico.com/data.php (accessed on 9 October 2022). The two random multi-graph datasets are generated to test the effiiency of all algorithms.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ingalalli, V.; Ienco, D.; Poncelet, P. Mining Frequent Subgraph in Multigraphs. *Inf. Sci.* **2018**, *451–452*, 50–66. [CrossRef]
2. Djenouri, Y.; Lin, J.C.W.; Norvrag, K.; Ramampiaro, H.; Yu, P.S. Exploring Decomposition for Solving Pattern Mining Problems. *ACM Trans. Manag. Inf. Syst.* **2021**, *12*, 15. [CrossRef]
3. Hopcroft, J.E.; Wong, J.K. Linear time algorithm for isomorphism of planar graphs (Preliminary Report). In Proceedings of the Sixth Annual ACM Symposium on Theory of Computing (STOC '74), Seattle, WA, USA, 30 April–2 May 1974.
4. Cordella, L.; Foggia, P.; Sansone, C.; Vento, M. *Performance Evaluation of the VF Graph Matching Algorithm*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germnay, 1994; Volume 1035, pp. 1177–1192.
5. Cordella, L.; Foggia, P.; Sansone, C.; Vento, M. A (Sub) Graph Isomorphism Algorithm for Matching Large Graphs. *IEEE Trans. Pattern Anal. Mach. Intell.* **2004**, *26*, 1367–1372. [CrossRef] [PubMed]
6. Zhao, P.; Han, J. On Graph Query Optimization in Large Networks. *Proc. VLDB Endow.* **2010**, *3*, 340–351. [CrossRef]
7. Shang, H.; Zhang, Y.; Lin, X.; Yu, J.X. Taming Verification Hardness: An Efficient Algorithm for Testing Subgraph Isomorphism. *Proc. VLDB Endow.* **2008**, *1*, 364–375. [CrossRef]
8. SnÃąÅąel, V.; DrÃąÅ¿dilovÃą, P.; PlatoÅą, J. Cliques Are Bricks for k-CT Graphs. *Mathematics* **2021**, *9*, 1160. . 10.3390/math9111160. [CrossRef]
9. Moorman, J.D.; Tu, T.K.; Chen, Q.; He, X.; Bertozzi, A.L. Subgraph Matching on Multiplex Networks. *IEEE Trans. Netw. Sci. Eng.* **2021**, *8*, 1367–1384. [CrossRef]
10. Han, M.; Kim, H.; Gu, G.; Park, K.; Han, W.S. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In Proceedings of the 2019 International Conference on Management of Data (SIGMOD '19), Amsterdam Netherlands, 30 June–5 July 2019; Association for Computing Machinery: New York, NY, USA, 2019; pp. 1429–1446. [CrossRef]
11. Bi, F.; Chang, L.; Lin, X.; Qin, L.; Zhang, W. Efficient Subgraph Matching by Postponing Cartesian Products. In Proceedings of the 2016 International Conference on Management of Data (SIGMOD 16), San Francisco, CA, USA, 26 June–1 July 2016; Association for Computing Machinery: New York, NY, USA, 2016; pp. 1199–1214. [CrossRef]
12. Han, W.S.; Lee, J.; Lee, J.H. Turboiso:Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD 13), New York, NY, USA, 22–27 June 2013; Association for Computing Machinery: New York, NY, USA, 2013; pp. 337–348. [CrossRef]
13. He, H.; Singh, A.K. Graphs-at-a-Time: Query Language and Access Methods for Graph Databases. In Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data (SIGMOD 08), Vancouver, BC, Canada, 9–12 June 2008; Association for Computing Machinery: New York, NY, USA, 2008; pp. 405–418. [CrossRef]
14. Ullmann, J. Bit-vector algorithms for binary constraint satisfaction and subgraph isomorphism. *J. Exp. Algorithmics (JEA)* **2010**, *15*, 1–64. [CrossRef]
15. Solnon, C. AllDifferent-based filtering for subgraph isomorphism. *Artif. Intell.* **2010**, *147*, 850–864. [CrossRef]
16. Messmer, B.T.; Bunke, H. Subgraph isomorphism detection in polynomial time on preprocessed model graphs. In *Recent Developments in Computer Vision, Proceedings of the Second Asian Conference on Computer Vision, ACCV '95, Singapore, 5–8 December 1995*; Lecture Notes in Computer Science; Springer: Berlin/Heidelberg, Germany, 1999; pp. 1177–1192.
17. Cibej, U.; FÃijrst, L.; Mihelic, J. A Symmetry-Breaking Node Equivalence for Pruning the Search Space in Backtracking Algorithms. *Symmetry* **2019**, *11*, 1300. [CrossRef]
18. Ingalalli, V.; Ienco, D.; Poncelet, P. SuMGra:Quering Multigraph via efficient Indexing. In *Database and Expert Systems Applications, Proceedings of the 27th International Conference, DEXA 2016, Porto, Portugal, 5–8 September 2016*; Lecture Notes in Computer Science; Springer: Cham, Switzerland, 2016.
19. Fan, W. Graph Pattern Matching Revised for Social Network Analysis. In Proceedings of the 15th International Conference on Database Theory (ICDT 12), Berlin, Germany, 26–29 March 2012; Association for Computing Machinery: New York, NY, USA, 2012; pp. 8–21. [CrossRef]
20. Kim, J.; Shin, H.; Han, W.S.; Hong, S.; Chafi, H. Taming Subgraph Isomorphism for RDF Query Processing. *Proc. VLDB Endow.* **2015**, *8*, 1238–1249. [CrossRef]
21. Zhu, Q.; Yao, Y.; Li, F.; Cai, W.; Liao, Q. Superstructure Searching Algorithm for Generic Reaction Retrieval. *J. Chem. Inf. Model.* **2005**, *45*, 1214–1222. [CrossRef] [PubMed]
22. Riesen, K.; Jiang, X.; Bunke, H. Exact and Inexact Graph Matching: Methodology and Applications. In *Managing and Mining Graph Data*; Springer: Boston, MA, USA, 2010; pp. 217–247.

23. Tian, Y.; McEachin, R.C.; Santos, C.; States, D.J.; Patel, J.M. SAGA: A subgraph matching tool for biological graphs. *Bioinformatics* **2006**, *23*, 232–239. [CrossRef] [PubMed]
24. Babić, D.; Reynaud, D.; Song, D. Malware Analysis with Tree Automata Inference. In *Computer Aided Verification, Proceedings of the 23rd International Conference, CAV 2011, Snowbird, UT, USA, 14–20 July 2011*; CAV 11; Springer: Berlin/Heidelberg, Germany, 2011; pp. 116–131.
25. Wu, J.; Chen, L. A Fast Frequent Subgraph Mining Algorithm. In Proceedings of the 2008 The 9th International Conference for Young Computer Scientists, Hunan, China, 18–21 November 2008; pp. 82–87. [CrossRef]
26. Wegner, A.E. Subgraph Covers: An Information-Theoretic Approach to Motif Analysis in Networks. *Phys. Rev. X* **2014**, *4*, 041026. [CrossRef]
27. Keller, S.; Miettinen, P.; Kalinina, O.V. Frequent subgraph mining for biologically meaningful structural motifs. *bioRxiv* **2020**. [CrossRef]
28. Kesavan, L. Frequent Subgraph Mining Algorithms—A Survey and Framework for Classification. *Comput. Sci. Inf. Technol.* **2012**, *2*, 189–202. [CrossRef]
29. Bickle, A. *The K-Cores of a Graph*; Western Michigan University: Kalamazoo, MI, USA, 2010.
30. Batagelj, V.; Zaversnik, M. An O (m) Algorithm for Cores Decomposition of Networks. *arXiv* **2003**, arXiv:cs/0310049.
31. Chang, L.; Qin, L. *Cohesive Subgraph Computation over Large Sparse Graphs: Algorithms, Data Structures, and Programming Techniques*, 1st ed.; Springer Publishing Company, Incorporated: Cham, Switzerland, 2018.
32. Pashanasangi, N.; Seshadhri, C. Efficiently Counting Vertex Orbits of All 5-Vertex Subgraphs, by EVOKE. In Proceedings of the 13th International Conference on Web Search and Data Mining, Houston, TX, USA, 3–7 February 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 447–455.
33. Savnik, I. Index Data Structure for Fast Subset and Superset Queries. In *Availability, Reliability, and Security in Information Systems and HCI, Proceedings of the IFIP WG 8.4, 8.9, TC 5 International Cross-Domain Conference, CD-ARES 2013, Regensburg, Germany, 2–6 September 2013*; Springer: Berlin/Heidelberg, Germany, 2013.
34. Hopcroft, J.; Karp, R. An n5/2 Algorithm for Maximum Matchings in Bipartite Graphs. *SIAM J. Comput.* **1973**, *2*, 225–231. [CrossRef]
35. Alcobaca, E.; Siqueira, F.; Garcia, L.P.; Rivolli, A.; Oliva, J.; de Carvalho, A. MFE: Towards reproducible meta-feature extraction. *J. Mach. Learn. Res.* **2020**, *21*, 1–5.
36. Domenico, M.D. Complex Multilayer Networks Lab at FBK. Available online: https://manliodedomenico.com/data.php (accessed on 9 October 2022).