SubISO: A Scalable and Novel Approach for Subgraph Isomorphism Search in Large Graph

Muhammad Abulaish, SMIEEE
Department of Computer Science
South Asian University, Delhi, India
abulaish@ieee.org

Zubair Ali Ansari
Department of Computer Science
Jamia Millia Islamia, Delhi, India
zuberaliansari@gmail.com

Jahiruddin
Department of Computer Science
Jamia Millia Islamia, Delhi, India
jahir.jmi@gmail.com

***Abstract*—Querying large graphs to retrieve information in permissible time is an emerging research problem, and it has roots in various application domains, mainly to analyse large networks. For a given query graph, the aim of subgraph isomorphism finding in a data graph is to identify all its subgraphs that are isomorphic to the query graph, and it has become the central problem for querying large graphs. Though different research groups have proposed many techniques using graph compression, postponing cartesian products, and candidate region exploration in the recent past, most of them show exponential behaviour for some query graphs on a large data graph. In this paper, we propose a subgraph isomorphism finding method, SubISO, which uses an objective function based on the eccentricity and some isomorphic invariants of the vertices of the query graph to minimize the number and size of the candidate regions in the data graph. Since subgraph isomorphism finding has a large number of solutions, especially in a large graph with repeated node labels, we propose to limit the maximum number of recursive calls of the generic *subgraph search* function to complete the execution of SubISO and return at most k-relevant matches. The SubISO finds at most k-relevant solutions in reasonable elapsed time over the queries for which existing state-of-the-art methods show exponential behaviour.**

***Index Terms*—Data Mining, Graph Mining, Graph Searching, Subgraph Isomorphism**

I. Introduction

Graphs are the promising tools to study large networks. Recently information acquisition from large graphs has attracted the attention of researchers from different fields, due to the growing need of large networks to model connected data. However, querying large graphs in permissible time is not feasible without the development of the efficient and scalable methods to identify all occurrences of subgraph isomorphisms. The subgraph isomorphism finding has found numerous applications in diverse application areas, such as *pattern recognition*, *plagiarism detection*, *biological network analysis*, *customers interactions analysis*, and *social network analysis*.

For given data and query graphs, the objective of the subgraph isomorphism search problem is to extract all subgraphs of the data graph that have an isomorphic image of the query graph. The data graph is usually largely connected labeled graph or a set of various smaller connected graphs. On the other hand, query graph is generally a small connected labeled graph. Although subgraph search problem is NP-complete [1], recently significant efforts have been emerged to find subgraph isomorphisms in reasonable elapsed time in large data graphs [2]–[5]. However, there are still some open challenges, as given below, that need to be addressed.

1) *Number of candidate regions and their size*: Both number and size of candidate regions play an important role in subgraph isomorphism finding. Some algorithms like Turbo$_{\text{ISO}}$ [4] use a ranking function based on the query graph's vertex properties to identify candidate regions in the data graph to prune search space. However, when labels and degrees of the vertices are not effective for identifying candidate regions, the ranking function is unable to divide the whole data graph into candidate regions and thereby finding query graph isomorphism needs to be performed on whole data graph.
2) *Matching order selection*: Although, imposing matching order on the vertices of the query graph could speed-up the matching process, ordering query graph vertices has a serious problem because of all state-of-the-art matching orders shows exponential behaviour for some kind of query graphs.

It can be observed that most the researchers have limited the number of solutions to be printed due to the existence of an enormous number of solutions for a given query graph into the data graph. It can also be observed that the generic `SubgraphSearch` function performs a substantial number of recursive calls for some query graphs, and accordingly, the elapsed time increases exponentially. To deal with this problem, researchers applied various techniques, such as appropriate matching order selection, optimizing an objective function, graph compression, etc. However, besides all these the `SubgraphSearch` function can still perform a large number of recursive calls, and thereby unable to complete the execution in reasonable time for some sets of queries and datasets. Therefore, if we relax the subgraph isomorphism search problem to find at most k-relevant solutions and restrict the maximum number of recursive calls for the `SubgraphSearch` function, then we may get at most k-relevant solutions for all query sets and datasets under permissible elapsed time.

In this paper, we propose a subgraph isomorphism searching method, SubISO, that uses an objective function based on the eccentricity and some isomorphic invariants of the vertices of query graph to minimize the number and size of the candidate regions in the data graph. In our experiment, it is found that the elapsed time exponentially grows for a very large number of recursive calls, whereas the number of embeddings does not grow at the same rate. Therefore, we also propose to restrict the maximum number of recursive calls to a reasonable value. By restricting the number of recursive calls, relevant solutions can be found in permissible elapsed time for all those queries for which existing methods are unable to complete execution and print any solution.

In brief, the main contributions of this paper are summarized as follows:

- Introduction of an objective function based on the eccentricity and isomorphic invariants of the query graph vertices to minimize both size and number of candidate regions in the data graph.
- Proposal to restrict the maximum number of recursive calls of `SubgraphSearch` function to complete the execution and find k-relevant solutions of the subgraph isomorphism Search Problem.
- A solution to handle query graphs for which one of the contemporary method $\text{Turbo}_{\text{ISO}}$ is unable to complete execution and find any solution.

The remaining part of the paper is organized as follows. Section II presents an overview of the existing works on subgraph isomorphism searching. Section III presents a mathematical formation of subgraph isomorphism search problem. It also presents a brief description of the related concepts. Section IV presents a detailed description of our proposed method for subgraph isomorphism search in large graphs. Section V presents the experimental setup and evaluation results. It also presents a comparison of the proposed approach with a popular state-of-the-art method for subgraph isomorphism search. Finally, section VI concludes the paper and reveals future directions of research.

II. Related Works

Since subgraph isomorphism serach probelm has its root in various application domains, a number of solutions have been proposed by various researchers, e.g., Ullmann's algorithm [6], `VF2` [7], `QuickSI` [8], `GraphQL` [9], `GADDI` [10], `SPath` [11], `NOVA` [12], `STW` [5], $\text{Turbo}_{\text{ISO}}$ [4], `CFL-Match` [3], and Sum_{ISO} [2]. Ullmann's algorithm [6] was the first attempt to address subgraph isomorphism search problem. However, its time complexity is greater than the most of the existing algorithms bacause it neither consider any kind of matching order nor any pruning method to prune unmatched vertices. `VF2` [7] exploited connectivity and already matched vertices for pruning candidate vertices. `QuickSI` [8] proposed a matching order based on infrequent labels. `GraphQL` [9] generated search space using neighborhood subgraphs or their profiles for pruning the candidate region and then reducing the search space at the same time using global structural information. `GADDI` [10] was specially designed for biological networks and used neighbourhood discriminative substructures for making the index. `SPath` [11] constructed a new graph indexing technique by utilizing neighborhood signatures of vertices. However, index creation for large graphs is a costly process. Using labels distribution of the neighborhood vertices, the authors in [12] proposed `NOVA` algorithm and index building process. Although `NOVA` performs good over sparse graphs, it does not work on the graphs having vertices of degree 10 or larger [13]. In [5], the authors presented an algorithm named `STW` for subgraph matching using distributed memory. However, they did not use any type of graph indexing; instead, they used parallel computing for graph exploration and query graph processing.

Han et al. proposed $\text{Turbo}_{\text{ISO}}$ in [4], which is centered around identifying candidate regions. They used combination and permutation to find all solutions for a given query graph. In an effort to minimize the number of regions $\text{Turbo}_{\text{ISO}}$ uses a ranking function which could not work if degree and label of the vertex are not discriminative. Lately, `CFL-Match` [3] and Sum_{ISO} [2] has been proposed, reporting a comparatively better performance. `CFL-Match` aims to decompose query graph into *core*, *forest*, and *leave* structures, and thereafter it performs subgraph matching using these fine-grained structures. On the other hand, Sum_{ISO} does graph compression and finds all existing matches of the query graph in uncompressed version of the data graph. `BoostIso` was proposed in [14] to speed up the process of finding subgraph isomorphisms. The authors in [15] proposed `VF3`, which is suitable to search subgraph isomorphisms on huge and dense graph.

A comparison of five subgraph isomorphism finding algorithms, namely `GraphQL`, `SPath`, `QuickSI`, `GADDI`, and `VF2` on real-world datasets is presented in [16]. `SGMatch` was proposed in [17] which performs graph decomposition (i.e., representation of a graph into smaller unit, called graphlets) and uses the graphlets to find subgraph isomorphism. Apart from exact subgraph matching, inexact matching or approximate matching is another active research area, and a number of similarity models have been introduced by the different research group to find approximate match of query graph in the data graph.

III. Background

Starting with a brief introduction of the graph and related concepts, in this section, we present a mathematical formation of the subgraph isomorphism search problem.

Definition 1. `Graph`: *A graph $G =< V, E, l_V, \Sigma >$ can be defined as a four-tuple, where V represents a non-empty set of objects known as vertices, E represents a subset of the Cartesian product of the V with itself, and the elements of E are known as edges, Σ denotes a set of labels that can be assigned to the vertices, and $l_V : V \rightarrow \Sigma$ is a function, which assigns a unique label from Σ to each vertex of V.*

Definition 2. `Maximum Neighborhood-Degree`: *The maximum neighborhood-degree of a vertex $u \in V(G)$, denoted by $\overline{\mathcal{N}_{V(G)}deg(u)}$, is the maximum degree of a vertex in the neighborhood of u. i.e., $\overline{\mathcal{N}_{V(G)}deg(u)} = \max_{v \in \mathcal{N}(u)}\{deg(v)\}$.*

Definition 3. `Eccentricity`*: The eccentricity of a vertex $u \in V(G)$ is denoted by $Eccen(u)$, and it is the maximum-length shortest path from u to any other vertex of G, i.e., $Eccen(u) = \max_{v \in V(G)}\{\delta(u, v)\}$, where $\delta(u, v)$ is the distance between u and v, and it is defined as the length of the shortest path from u to v.*

Definition 4. ϵ`-neighbourhood` *The ϵ-neighbourhood of a vertex $u \in V(G)$ is denoted by $\mathcal{N}_\epsilon(u)$, and it is the collection of all those vertices of $V(G)$ who are at most ϵ-distance from u in G.*

Definition 5. `Subgraph`*: A graph $G_i =< V_i, E_i, l_{Vi}, \Sigma_i >$ is a subgraph of a another graph $G_j =< V_j, E_j, l_{Vj}, \Sigma_j >$ if $V_i \subseteq V_j$, $E_i \subseteq E_j$, $\Sigma_i \subseteq \Sigma_j$, and $l_{Vi}(u) = l_{Vj}(u), \forall u \in V_i$.*

Definition 6. `Subgraph Isomorphism`*: A graph $G_q =< V_q, E_q, l_{Vq}, \Sigma_q >$ is subgraph isomorphic to another graph $G_d =< V_d, E_d, l_{Vd}, \Sigma_d >$ if there exists a injective mapping $f : V_q \to V_d$ with the following properties:*

1) $l_{Vq}(u) = l_{Vd}(f(u)), \forall u \in V_q$
2) $\forall (u, v) \in E_q \ \exists \ (f(u), f(v)) \in E_d$

where u,v$\in V_q$.

Definition 7. `Isomorphic Invariants`*: Isomorphic invariants are the properties of a given graph that are preserved under any isomorphism defined on the graph.*

For example, the number of edges, number of vertices, vertex degree, maximum neighborhood degree, and number of cycles are the well-known isomorphic invariants.

For a given query graph G_q and data graph G_d, the subgraph isomorphism search problem is to enumerate all subgraphs of G_d that have isomorphic image of G_q.

`Problem Statement`: For a given query graph G_q, we need to find at most k subgraphs of the data graph G_d that have isomorphic image of the query graph G_q.

Query graphs are usually connected and small in size, e.g., dozens of nodes or edges, whereas data graphs are often large graphs having a million or billion number of vertices and edges. For example, `Friendster` [18] is an online social network dataset in which nodes represent users and edges represent friendship relation among the users. The `Friendster` dataset consists of more than 65 million and 180 billion vertices and edges, respectively.

IV. Proposed Method

In this section, we present a novel method, SubISO, for subgraph isomorphism finding. Starting with finding a pivot vertex in the query graph using an objective function based on eccentricity and isomorphic invariants of the vertices of the query graph, SubISO aims to identify various candidate regions and their size in the data graph. Thereafter, it minimizes the size of candidate regions by removing all those vertices that can not be a likely match with the query graph vertices. Finally, SubISO calls `SubgraphSearch` function to enumerate all matches in the candidate regions that are isomorphic to the query graph. Algorithm 1 presents the overall work-flow of SubISO formally. Further details about its various functioning modules, such as *pivot vertex selection*, *region exploration*, and *subgraphs enumeration* are given in the following subsections.

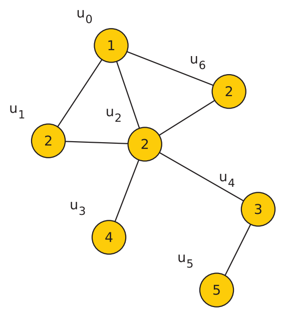

Figure 1: An exemplar query graph G_q

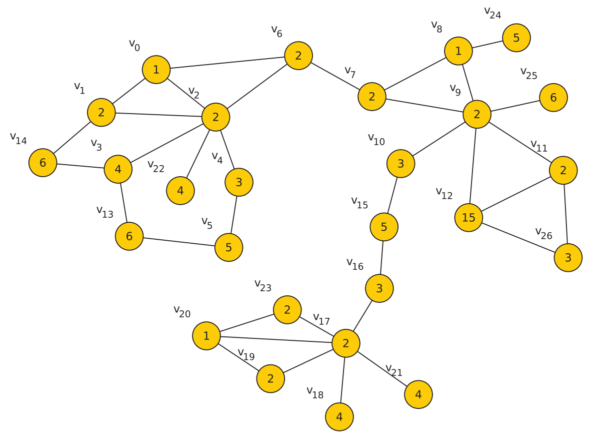

Figure 2: An exemplar data graph G_d

A. Pivot Vertex Selection

The first step of the proposed SubISO method is to identify the pivot vertex $\hat{u}$ in the query graph which is formally explained in algorithm 2. In this algorithm, our aim to minimize the objective function $(|\psi(u)| \times Eccen(u))$, where $u \in V(G_q)$ and $\psi(u)$ is the collection of all vertices of the given data graph G_d that are possible candidate matches of u. Figures 1 and 2 present an exemplar query and data graphs, respectively to illustrate the working of the pivot vertex selection process and remaining steps of the SubISO.

After applying steps 1-17 of algorithm 2 on the query graph G_q, $\mathcal{L}_3$ has three vertices u_0, u_2, and u_4, where u_0 has two matches v_0 and v_{20} in data graph G_d and $Eccen(u_0)$ is 3. Similarly, u_2 has two matches v_2 and v_{17} in G_d and $Eccen(u_2)$ is 2, and u_4 has three matches v_4, v_{10}, and v_{16} in G_d with $Eccen(u_4)$ as 2. Since u_2 minimizes $(|\psi(u)| \times Eccen(u))$, it is considered as the pivot vertex $\hat{u}$. If we choose any other vertex of the query graph, then either number or size of the candidate region increases.

Algorithm 1: SubISO(G_d, G_q)

```
Input  : $G_d$: data graph, $G_q$: query graph
Output: A set $\mathcal{M}_G$ of all embeddings of $G_q$ in $G_d$
1  $\hat{u}, \psi(\hat{u}) \leftarrow PivotVertexSelection(G_d, G_q)$
2  $\mathcal{M}_G \leftarrow \emptyset$
3  $\epsilon \leftarrow Eccen(\hat{u})$
4  foreach $v \in \psi(\hat{u})$ do
5      $\mathcal{C}(\mathcal{R}) \leftarrow RegionExploration(G_d, G_q, v)$
6      if $\mathcal{C}(\mathcal{R}) = \emptyset$ then
7          continue
8      end
9      $\mathcal{M} \leftarrow \emptyset$
10     $\mathcal{M} \leftarrow \mathcal{M} \cup \{< \hat{u}, v >\}$
11     $\mathcal{M}_\mathcal{R} \leftarrow SubgraphSearch(G_q, G_d, \mathcal{C}(\mathcal{R}), \mathcal{M})$
12     $\mathcal{M}_G \leftarrow \mathcal{M}_G \cup \mathcal{M}_\mathcal{R}$
13 end
14 return $\mathcal{M}_G$
```

B. Region Exploration

The region exploration process aims to identify candidate regions in data graph corresponding to each vertex $v \in \psi(\hat{u})$. Since the region exploration process aims to explore each candidate region corresponding to the every match of the pivot vertex $\hat{u}$ in G_d, the maximum number of candidate regions is determined by $|\psi(\hat{u})|$, and the maximum size of each candidate region is determined by $Eccen(\hat{u})$, as proved in lemma 1.

Lemma 1. *If ϵ is an eccentricity of a vertex u in the query graph G_q, then the size of the candidate region in data graph can never exceed the size of the $\mathcal{N}_\epsilon(v)$ of a vertex $v \in \psi(u)$.*

Proof. Since the query graph G_q is connected, and ϵ is the eccentricity of a vertex $u \in G_q$, all vertices of the query graph must be at maximum ϵ distance from u. Thus, if $v \in G_d$ is a possible match of u in G_q, then all other matches of the vertices of G_q must be at maximum ϵ distance from v. Consequently, matches of all vertices of G_q must be in $\mathcal{N}_\epsilon(v)$ of v. $\square$

Algorithm 3 presents the region exploration process formally. First, for each $v \in \psi(\hat{u})$, we determine ϵ-neighbourhood $\mathcal{N}_\epsilon(v)$ in the data graph G_d, where ϵ is the eccentricity of the pivot node $\hat{u}$. The ϵ-neighbourhood becomes the candidate region $\mathcal{R}$, in which a subgraph isomorphic image of the query graph G_q may exist. If size of $\mathcal{R}$ is less than the size of G_q, then a case of false positive exists and rest of the steps are skipped. Otherwise, only those vertices of the candidate region $\mathcal{R}$ are considered in $\hat{\mathcal{R}}$ that have same label as the label of a vertex $u \in G_q$ (steps 5-12). Thereafter, we further refine the region $\hat{\mathcal{R}}$ using the concept of *label*, *degree*, *maximum neighbourhood degree*, and *neighbours' labels*. After refining $\hat{\mathcal{R}}$, for every vertex $u \in G_q$, we get the set of all candidate matches $\psi(u)$ of u in $\hat{\mathcal{R}}$. The refined candidate region $\mathcal{C}(\mathcal{R})$ is obtained by collecting all candidate matches $\psi(u)$.

Algorithm 2: PivotVertexSelection(G_d, G_q)

```
Input  : $G_d$: data graph, $G_q$: query graph
Output: The pivot vertex $\hat{u}$ and its candidate matches $\psi(\hat{u})$
1  $\mathcal{L}$  // List of ordered pair $< u_i, |\psi(u_i)| >$
   where $|\psi(u_i)|$ is the number of
   candidate vertices of vertex $u_i \in G_q$.
2  $i = 0$
3  foreach vertex $u_i \in V(G_q)$ do
4      foreach vertex $v \in V(G_d)$ do
5          if $L_{V(G_q)}(u_i) = L_{V(G_d)}(v)$ &
           $deg_{V(G_q)}(u_i) \leq deg_{V(G_d)}(v)$ &
           $\overline{\mathcal{N}_{V(G_q)}deg(u_i)} \leq \overline{\mathcal{N}_{V(G_d)}deg(v)}$ then
6              $l_1 \leftarrow L_{V(G_q)}(\mathcal{N}(u_i))$
7              $l_2 \leftarrow L_{V(G_d)}(\mathcal{N}(v))$
8              if $l_1 \subseteq l_2$ then
9                  $\psi(u_i) \leftarrow \psi(u_i) \cup \{v\}$
10             end
11         end
12     end
13     $\mathcal{L}[i] \leftarrow < u_i, |\psi(u_i)| >$
14     $i \leftarrow i + 1$
15 end
16 sort the list $\mathcal{L}$ on increasing order of $|\psi(u_i)|$
17 $\mathcal{L}_3 \leftarrow$ firstThreeOrderPairs($\mathcal{L}$)
18 $\hat{u} \leftarrow \underset{<u,|\psi(u)|>\in\mathcal{L}_3}{\arg\min} \{|\psi(u)| \times Eccen(u)\}$
19 return $\hat{u}$, $\psi(\hat{u})$
```

Figure 3 presents the identified candidate regions from the data graph given in figure 2. The identified regions are labeled as $\mathcal{R}_1$ and $\mathcal{R}_2$ in this figure. For region $\mathcal{R}_1$, the refined candidate region is $\mathcal{C}(\mathcal{R}_1) = \{\psi(u_2), \psi(u_0), \psi(u_1), \psi(u_3), \psi(u_4), \psi(u_5), \psi(u_6)\}$, and elements of each $\psi(u_i)$ are as given below:

- $\psi(u_2)$ = $\{v_2\}$
- $\psi(u_0)$ = $\{v_0\}$
- $\psi(u_1)$ = $\{v_1, v_6, v_7\}$
- $\psi(u_3)$ = $\{v_3, v_{22}\}$
- $\psi(u_4)$ = $\{v_4\}$
- $\psi(u_5)$ = $\{v_5\}$
- $\psi(u_6)$ = $\{v_1, v_6, v_7\}$

Starting with $\{< u_2, v_2 >\}$ as an initial value of $\mathcal{M}$, we call $SubgraphSearch(G_q, G_d, \mathcal{C}(\mathcal{R}_1), \mathcal{M})$ to explore a possible embedding. Similarly, we can explore another candidate region $\mathcal{R}_2$. After exploring all candidate regions, we get all existing embeddings of the query graph G_q in the data graph G_d. The concept of embedding is formally defined in the following sub-section.

C. Subgraph Enumeration

After candidate region exploration and filtering, we have only those subgraphs that may be isomorphic to the query

Algorithm 3: RegionExploration(G_d, G_q, v)

```
Input  : $G_d$: data graph, $G_q$: query graph, $v$: a vertex
         of $\psi(\hat{u})$
Output: Minimized region $\mathcal{C}(\mathcal{R})$ corresponding to
         vertex $v$.
1  $\mathcal{R} \leftarrow \mathcal{N}_\epsilon(v)$
2  if $|V(\mathcal{R})| < |V(G_q)|$ then
3      return $\emptyset$
4  end
5  $\hat{\mathcal{R}} \leftarrow \emptyset$
6  foreach $v \in V(\mathcal{R})$ do
7      foreach $u \in V(G_q)$ do
8          if $L_{V(G_d)}(v) = L_{V(G_q)}(u)$ then
9              $\hat{\mathcal{R}} \leftarrow \hat{\mathcal{R}} \cup \{v\}$
10         end
11     end
12 end
13 $\mathcal{C}(\mathcal{R}) \leftarrow \emptyset$
14 foreach $u \in V(G_q)$ do
15     $\psi(u) \leftarrow \emptyset$
16     foreach $v \in \hat{\mathcal{R}}$ do
17         if $L_{V(G_q)}(u) = L_{V(G_d)}(v)$ &
            $\overline{deg_{V(G_q)}(u)} \leq \overline{deg_{V(G_d)}(v)}$ &
            $\overline{\mathcal{N}_{V(G_q)}deg(u)} \leq \overline{\mathcal{N}_{V(G_d)}deg(v)}$ then
18             $l_u \leftarrow L_{V(G_q)}(\mathcal{N}(u))$
19             $l_v \leftarrow L_{V(G_d)}(\mathcal{N}(v))$
20             if $l_u \subseteq l_v$ then
21                 $\psi(u) \leftarrow \psi(u) \cup \{v\}$
22             end
23         end
24     end
25     $\mathcal{C}(\mathcal{R}) \leftarrow \mathcal{C}(\mathcal{R}) \cup \psi(u)$
26 end
27 return $\mathcal{C}(\mathcal{R})$
```

graph, and hence only embeddings need to be identified. To find possible embeddings, we have used `SubgraphSearch` function, which is a depth-first-search function of the Ullmann's method [6]. It enumerates all embeddings of query graph G_q after verifying the adjacency relationships among the vertices of the given query graph G_q and their respective matches in the candidate region $\mathcal{R}$. The concept of embedding can be defined formally as follows:

Definition 8. `Embedding`*: If $f : G_q \rightarrow G_d$ is a subgraph isomorphic map from query graph G_q to data graph G_d, then an embedding $\mathcal{M}$ is the set of all ordered pairs $< u, v >$ where $u \in V(G_q)$, $v \in V(G_d)$, and $f(u) = v$.*

In Algorithm 4, we first take a vertex $u \in G_q$ and the corresponding $\psi(u)$ from $\mathcal{C}(\mathcal{R})$. For each vertex $v \in \psi(u)$, if u and v satisfy the *joining* condition specified in algorithm 5, then we add pair $< u, v >$ in embedding $\mathcal{M}$. In order to check whether vertices u and v can be joined or not, we have used algorithm 5, which returns *true* if u and v can be

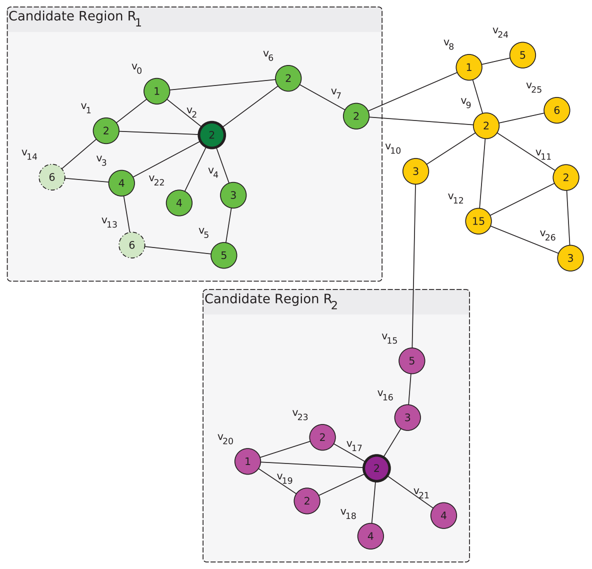

Figure 3: A visualization of the identified regions in the data graph given in figure 2

joined, otherwise *false*. After adding $< u, v >$ in embedding $\mathcal{M}$, algorithm 4 uses recursion to accomplish matching till it finds correct match for every vertex of the query graph. When matching fails, the algorithm restores the preceding state by removing the matches. In figure 3, the query graph G_q has four matches in the candidate region $\mathcal{R}_1$ that are listed below.

- $\{< u_2, v_2 >, < u_0, v_0 >, < u_1, v_1 >, < u_3, v_3 >, < u_4, v_4 >, < u_5, v_5 >, < u_6, v_6 >\}$
- $\{< u_2, v_2 >, < u_0, v_0 >, < u_1, v_6 >, < u_3, v_3 >, < u_4, v_4 >, < u_5, v_5 >, < u_6, v_1 >\}$
- $\{< u_2, v_2 >, < u_0, v_0 >, < u_1, v_1 >, < u_3, v_{22} >, < u_4, v_4 >, < u_5, v_5 >, < u_6, v_6 >\}$
- $\{< u_2, v_2 >, < u_0, v_0 >, < u_1, v_6 >, < u_3, v_{22} >, < u_4, v_4 >, < u_5, v_5 >, < u_6, v_1 >\}$

Algorithm 4: SubgraphSearch(G_q, G_d, $\mathcal{C}(\mathcal{R})$, $\mathcal{M}$)

```
Input  : $G_q$: query graph, $G_d$: data graph, $\mathcal{C}(\mathcal{R})$: list
         of vertex matches of the query graph vertices
         in candidate region $\mathcal{R}$, $\mathcal{M}$: list of embeddings
Output: List of all embeddings of the query graph $G_q$
         in $\mathcal{R}$
1  if $|\mathcal{M}| = |V(G_q)|$ then
2      Print($\mathcal{M}$)
3  end
4  $u \leftarrow$ Select an unmatched vertex from $V(G_q)$
5  $\psi(u) \leftarrow$ {find all possible candidate match of u from
     $\mathcal{C}(\mathcal{R})$}
6  foreach $v \in \psi(u)$ do
7      if $IsJoinable(u, v, \mathcal{M}, G_q, G_d) = True$ then
8          $\mathcal{M} \leftarrow \mathcal{M} \cup \{< u, v >\}$
9          SubgraphSearch($G_q$, $G_d$, $\mathcal{C}(\mathcal{R})$, $\mathcal{M}$)
10         Delete$< u, v >$ from $\mathcal{M}$
11     end
12 end
```

```
Algorithm 5: IsJoinable($u, v, \mathcal{M}, G_q, G_d$)
Input  : Query vertex $u \in V(G_q)$, candidate vertex
         $v \in \psi(u)$, $\mathcal{M}$: embedding, $G_q$: query graph,
         $G_d$: data graph
Output: Return True if $< u, v >$ joinable, otherwise
         return False
1 foreach $< u', v' > \in \mathcal{M}$ do
2     if $(u, u') \in E(G_q)$ & $(v, v') \notin E(G_d)$ then
3         return False
4     end
5 end
6 return True
```

V. Experimenttal Setup and Result

In this section, we present an experimental assessment of our proposed subgraph isomorphism search method SubISO. We also present a comparative analysis of SubISO with Turbo$_{\mathrm{ISO}}$, which is a well-known state-of-the-art method for subgraph isomorphism finding. Further details about the dataset, query set, performance evaluation, and comparative analysis are presented in the following sub-sections.

A. Dataset and Query set

To assess the performance of the proposed subgraph isomorphism search method SubISO, we have considered Human data graph, which is a simple connected labelled graph. In this graph, vertices represent proteins and edges represent protein-protein interactions. Table I presents the statistics of this data graph. Though original Human data graph used in Turbo$_{\mathrm{ISO}}$ has multiple vertex labels, we converted it into single vertex label graph, as used in [3]. We used eight Human query sets from [3], but we converted each query graph into simple undirected with single vertex label graph to match the structure of the queries used in Turbo$_{\mathrm{ISO}}$. Each query set contains 100 undirected connected labeled graphs, and named as q_{10s}, q_{15s}, q_{20s}, q_{25s}, q_{10d}, q_{15d}, q_{20d}, q_{25d}, where two-digit number in the subscript represents the number of vertices in each query graph, s represents *sparse* graph and d represents *dense* graph. For example, q_{10s} query file has 100 simple undirected labelled graphs in which each graph has 10 vertices and each graph of this file is of type sparse.

Table I: Statistics of the Human data graph

Graph Entity	Value
Number of Vertices	4675
Number of Edges	86282
Number of distinct label	90
Average vertex degree	36.82

B. Evaluation Results

We performed all experiments on a PC having Intel Core i5-6600 processor along with 4GB RAM. We implemented SubISO in 'C' programming language using igraph library [19] and compiled using GCC compiler on Unix environment. We evaluated SubISO in terms of the sum of CPU elapsed time in milliseconds (ms) to process all 100 queries of a query set, which is the sum of the total times of obtaining candidate regions and finding and printing the query graphs isomorphism (embedding) in data graph for the query set. Table II presents the evaluation results of SubISO in terms of elapsed times. This table also presents the number of embeddings obtained against each query set. It can be observed from this table that SubISO not only completed its execution to uncover the embeddings of all eight query sets in a reasonable time, but it also finds enough number of embeddings for each query graph in the query set.

Table II: Performance of SubISO against test query sets over Human data graph

S.No.	Query set	Total number of embeddings	Sum of elapsed time (ms)
1	q_{10s}	51799	4405.58886
2	q_{15s}	26809	14753.82910
3	q_{20s}	17932	12120.17090
4	q_{25s}	15738	12350.64453
5	q_{10d}	30374	3351.95068
6	q_{15d}	19064	10212.44824
7	q_{20d}	11838	10279.40234
8	q_{25d}	13880	16026.04004

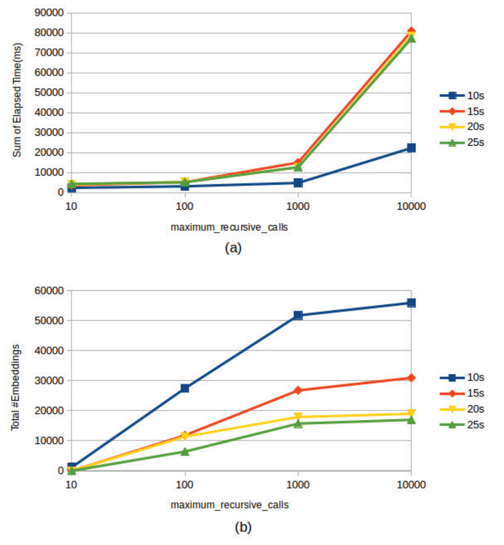

Figure 4: Performance evaluation results of SubISO over sparse query sets

We have also analyzed the performance of SubISO over sparse and dense query sets separately to find the relationship among the number of recursive calls, elapsed time and number of obtained embeddings. Figure 4 presents the performance of SubISO over the sparse query sets. Figure 4(a) presents a plot of the sum of elapsed times to find the embeddings of sparse query graphs of the query sets q_{10s}, q_{15s}, q_{20s}, and q_{25s} against the maximum number of recursive calls of SubgraphSearch function. Whereas, figure 4(b) presents a plot of the total number of embeddings of sparse query

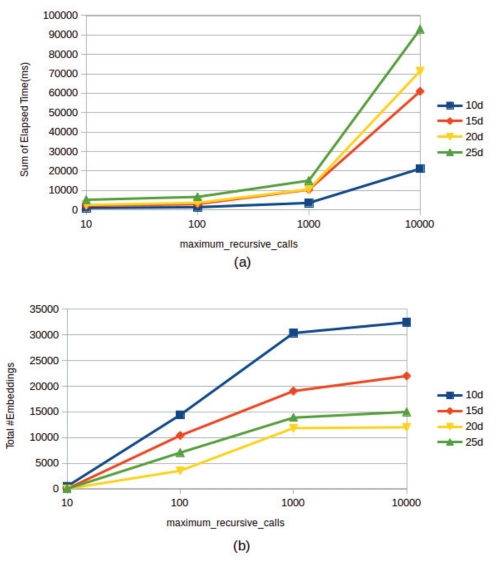

Figure 5: Performance evaluation results of SubISO over dense query sets

graphs against the maximum number of recursive calls of `SubgraphSearch` function on the same query sets. It can be observed from figure 4 that by increasing the maximum number of recursive calls of `SubgraphSearch` function beyond 1000, the elapsed time to find embeddings are increased exponentially, but there is a small increase in the number of embeddings. It shows that after a certain value of the maximum number of recursive calls SubISO takes a large number of recursive calls, increasing elapsed times exponentially to find a very small number of embeddings. Therefore, by fixing the maximum number of recursive calls of `SubgraphSearch` function, elapsed times can be reduced drastically without much missing too many embeddings. By fixing the maximum number of recursive calls will not only reduce the elapsed time, but we would be able to find a large number of embeddings (possibly not all) for those query sets for which existing algorithms are unable to find embeddings. In the same line, figure 5 presents the performance evaluation results of SubISO over dense query sets. Figure 5(a) presents a plot of the elapsed time against maximum recursive calls of `SubgraphSearch` function on the query sets q_{10d}, q_{15d}, q_{20d} and q_{25d}. Whereas, figure 5(b) presents a plot of the total number of embeddings of dense query graphs against the maximum number of recursive calls of `SubgraphSearch` function on the same query sets. It can be observed from figure 5 that SubISO shows similar behavior over the dense query sets as that of the sparse query sets.

C. Comparative Analysis

In this section, we present a comparative analysis of our proposed SubISO with one of the famous subgraph isomorphism finding methods Turbo$_{\text{ISO}}$, in terms of elapsed time. We have considered the same dataset and query set discussed in the previous sections. To this end, we run the executable codes of Turbo$_{\text{ISO}}$ provided by the authors on the same machine discussed above, but with the Windows platform. In case Turbo$_{\text{ISO}}$ was unable to complete the process in three hours, we considered it a very large elapsed time and represented with `INF`.

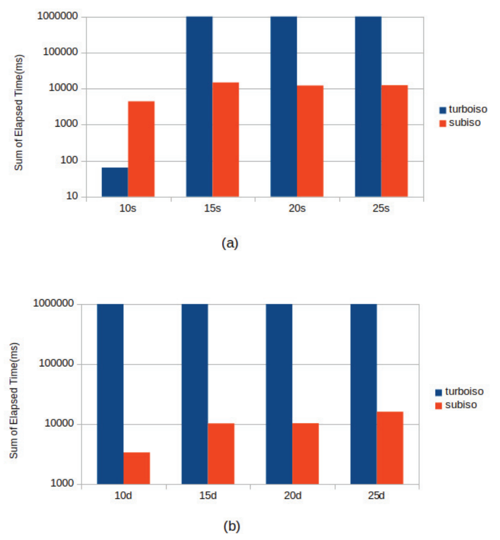

Figure 6: Comparison results of SubISO and Turbo$_{\text{ISO}}$ using all eight query sets on `Human` data graph

Figure 6 presents a comparative performance of both SubISO and Turbo$_{\text{ISO}}$ using all eight query sets over `Human` data graph. For comparison, we set the maximum recursive calls of `SubgraphSearch` function to 1000 and stopped SubISO after finding first 1000 embeddings per query graph. It can be observed from figure 6 that Turbo$_{\text{ISO}}$ takes `INF` execution time to find embeddings and SubISO significantly performs better for all query sets, except the query set q_{10s}.

Table III: Enumeration of individual queries from all query sets on which Turbo$_{\text{ISO}}$ couldn't complete execution

S.No.	Query set	Graph id
1	q_{15s}	75, 81
2	q_{20s}	32, 67, 98
3	q_{25s}	4, 5, 9, 24, 26, 27, 35, 41, 74, 88
4	q_{10d}	33
5	q_{15d}	9, 41, 42, 54, 57, 78, 88
6	q_{20d}	49, 55, 64, 65, 82
7	q_{25d}	0,2,7,26,27,50,58,65,73,94

Table III presents the list of query graphs from each query set on which Turbo$_{\text{ISO}}$ showed exponential behavior. We identified these query graph by running the Turbo$_{\text{ISO}}$ on the individual query graph of each query set. For query graphs listed in table III, Turbo$_{\text{ISO}}$ could not complete its execution in given time limit of three hours, and accordingly, it was unable to return any embeddings for these query graphs. It can be observed from table III that with increasing size of query graphs in the query sets, the number of query graphs on which

$\text{Turbo}_{\text{ISO}}$ showed exponential behavior increases. One of the reasons behind the exponential behavior of $\text{Turbo}_{\text{ISO}}$ on these query graphs may be the excessive number of recursive calls of `SubgraphSearch` function.

VI. Conclusion and Future Work

In this paper, we have proposed a scalable and novel method, SubISO, for subgraph isomorphism search in the large data graph. We have introduced the application of eccentricity and isomorphic invariants to find pivot vertex of query graph for subgraph matching. We have also introduced a new objective function that aims to minimize both number and size of candidate regions. We have considered a complex data graph obtained from `Human` data set for experimental evaluation of the proposed method, and illustrate how elapsed time and the total number of embeddings can be controlled by restricting the maximum number of recursive calls of the `SubgraphSearch` function. The proposed SubISO finds a good number of solutions in a reasonable time for those query graphs for which previously proposed methods are unable to complete their execution. Evaluation of SubISO on other benchmark datasets and its performance comparison with other contemporary methods for subgraph isomorphism search constitutes one of our future directions of research.

VII. Acknowledgment

This research work is an outcome of the R&D work undertaken project under the Visvesvaraya Ph.D. Scheme of Ministry of Electronics & IT, Government of India, being implemented by Digital India Corporation.

References

[1] M. R. Garey and D. S. Johnson, "Computers and intractability: a guide to the theory of np-completeness. 1979," *San Francisco, LA: Freeman*, vol. 58, 1979.

[2] C. Nabti and H. Seba, "Querying massive graph data: A compress and search approach," *Future Generation Computer Systems*, vol. 74, pp. 63–75, 2017.

[3] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient subgraph matching by postponing cartesian products," in *Proceedings of the 2016 International Conference on Management of Data*. ACM, 2016, pp. 1199–1214.

[4] W.-S. Han, J. Lee, and J.-H. Lee, "Turbo iso: towards ultrafast and robust subgraph isomorphism search in large graph databases," in *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data*. ACM, 2013, pp. 337–348.

[5] Z. Sun, H. Wang, H. Wang, B. Shao, and J. Li, "Efficient subgraph matching on billion node graphs," *Proceedings of the VLDB Endowment*, vol. 5, no. 9, pp. 788–799, 2012.

[6] J. R. Ullmann, "An algorithm for subgraph isomorphism," *Journal of the ACM (JACM)*, vol. 23, no. 1, pp. 31–42, 1976.

[7] L. P. Cordella, P. Foggia, C. Sansone, and M. Vento, "A (sub) graph isomorphism algorithm for matching large graphs," *IEEE transactions on pattern analysis and machine intelligence*, vol. 26, no. 10, pp. 1367–1372, 2004.

[8] H. Shang, Y. Zhang, X. Lin, and J. X. Yu, "Taming verification hardness: an efficient algorithm for testing subgraph isomorphism," *Proceedings of the VLDB Endowment*, vol. 1, no. 1, pp. 364–375, 2008.

[9] H. He and A. K. Singh, "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*. ACM, 2008, pp. 405–418.

[10] S. Zhang, S. Li, and J. Yang, "Gaddi: distance index based subgraph matching in biological networks," in *Proceedings of the 12th International Conference on Extending Database Technology: Advances in Database Technology*. ACM, 2009, pp. 192–203.

[11] P. Zhao and J. Han, "On graph query optimization in large networks," *Proceedings of the VLDB Endowment*, vol. 3, no. 1-2, pp. 340–351, 2010.

[12] K. Zhu, Y. Zhang, X. Lin, G. Zhu, and W. Wang, "Nova: A novel and efficient framework for finding subgraph isomorphism mappings in large graphs," in *Database Systems for Advanced Applications*. Springer, 2010, pp. 140–154.

[13] P. Peng, L. Zou, L. Chen, X. Lin, and D. Zhao, "Answering subgraph queries over massive disk resident graphs," *World Wide Web*, vol. 19, no. 3, pp. 417–448, 2016.

[14] X. Ren and J. Wang, "Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs," *Proceedings of the VLDB Endowment*, vol. 8, no. 5, pp. 617–628, 2015.

[15] V. Carletti, P. Foggia, A. Saggese, and M. Vento, "Challenging the time complexity of exact subgraph isomorphism for huge and dense graphs with vf3," *IEEE transactions on pattern analysis and machine intelligence*, 2017.

[16] J. Lee, W.-S. Han, R. Kasperovics, and J.-H. Lee, "An in-depth comparison of subgraph isomorphism algorithms in graph databases," in *Proceedings of the VLDB Endowment*, vol. 6, no. 2. VLDB Endowment, 2012, pp. 133–144.

[17] C. R. Rivero and H. M. Jamil, "Efficient and scalable labeled subgraph matching using sgmatch," *Knowledge and Information Systems*, vol. 51, no. 1, pp. 61–87, 2017.

[18] Archiveteam, "Friendster social network dataset: Friends," https://archive.org/details/friendster-dataset-201107, April 2011.

[19] G. Csardi and T. Nepusz, "The igraph software package for complex network research," *InterJournal, Complex Systems*, vol. 1695, no. 5, pp. 1–9, 2006.