Continuous Subgraph Matching via Cost-Model-based Dynamic Vertex Dominance Embeddings

Abstract

In many real-world applications such as social network analysis, knowledge graph discovery, biological network analytics, and so on, graph data management has become increasingly important and has drawn much attention from the database community. While many graphs (e.g., Twitter, Wikipedia, etc.) are usually involving over time, it is of great importance to study the *continuous subgraph matching* (CSM) problem, a fundamental, yet challenging, graph operator, which continuously monitors subgraph matching results over dynamic graphs with a stream of edge updates. To efficiently tackle the CSM problem, we carefully design a general CSM processing framework, based on novel *DynamIc Vertex DomINance Embedding* (DIVINE), which maps vertex neighborhoods into an embedding space to enable efficient subgraph matching and incremental maintenance under dynamic updates. Inspired by low pruning power for high-degree vertices, we propose a new *degree grouping* technique to decompose high-degree star patterns into groups of lower-degree star substructures, and devise *degree-aware star substructure synopses* (DAS^3) over embeddings of star substructure groups. We develop efficient algorithms to incrementally maintain dynamic graphs and answer CSM queries by traversing DAS^3 synopses and applying our designed *vertex dominance* and *range pruning strategies*. Through extensive experiments, we confirm the efficiency of our proposed DIVINE approach over both real and synthetic graphs.

CCS Concepts

• **Data Models and Languages → Graphs, social networks, web data, and semantic web**.

Keywords

Vertex Dominance Embedding, Continuous Subgraph Matching

ACM Reference Format:

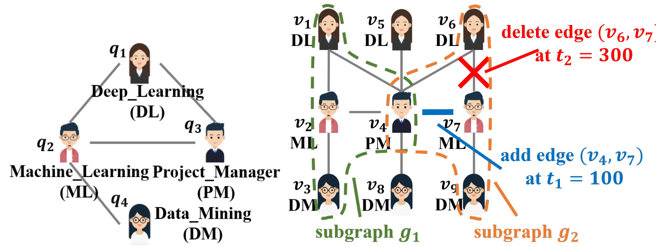
. 2018. Continuous Subgraph Matching via Cost-Model-based Dynamic Vertex Dominance Embeddings. In *Proceedings of Make sure to enter the correct conference title from your rights confirmation email (Conference acronym 'XX)*. ACM, New York, NY, USA, 18 pages. https://doi.org/XXXXXXX.XXXXXXX

1 Introduction

For the past few decades, the *subgraph matching* problem has been extensively studied as a fundamental operator in the graph data management [13, 38, 46, 61, 67, 72] for many real-world applications

Conference acronym 'XX, Woodstock, NY

ACM ISBN 978-1-4503-XXXX-X/2018/06
https://doi.org/XXXXXXX.XXXXXXX



(a) query graph q (b) dynamic collaboration network G_D

Figure 1: An example of the subgraph matching in dynamic collaboration networks G_D.

such as social network analysis, knowledge graph discovery, and pattern matching in biological networks. Given a large-scale data graph G, a subgraph matching query finds all the subgraphs of G that are isomorphic to a given query graph q.

While many previous works [8–10, 26, 30, 35, 53, 58] usually considered the subgraph matching over static graphs, real-world graph data are often dynamic and changing over time. For example, in social networks, friend relationships among users may be subject to changes (e.g., adding or breaking up with friends); similarly, in collaboration networks, people may start to collaborate on some project and then suddenly stop the collaboration for years. In these scenarios, it is important, yet challenging, to conduct the subgraph matching over such a large-scale, dynamic data graph G_D, upon updates (e.g., edge insertions/deletions). In other words, we need to continuously monitor subgraphs (e.g., user communities or collaborative teams) in G_D that follow query graph patterns q in real-world applications (such as online advertising to communities on social networks or finding collaboration teams from bibliographic networks).

Below, we give an example of the subgraph matching over dynamic collaboration networks in the expert team search application.

Example 1. ***(Monitoring Project Teams in Dynamic Collaboration Networks [19])*** *Due to numerous requests from departments/companies for recruiting (similar) project teams, a job search advisor may want to monitor the talent market (especially teams of experts) in dynamically changing collaboration networks.*

Consider a toy example of a collaboration network G_D in Figure 1. Figure 1(a) shows a desirable expert team pattern q, which is represented by a graph of 4 experts q_1~q_4, where each expert vertex q_i is associated with one's skill/role keyword (e.g., q_1 with keyword "Deep_Learning" and q_3 with "Project_Manager"), and each edge indicates that two experts had the collaboration before (e.g., edge (q_1, q_3)).

Figure 1(b) provides a dynamic collaboration network G_D, where collaboration edges among experts are inserted or deleted over time. For example, a new edge (v_4, v_7) is inserted at timestamp $t_1 = 100$, implying that two experts v_4 and v_7 start to collaborate with each other in a team. Similarly, at timestamp $t_2 = 300$, an existing edge (v_6, v_7) is removed from the graph G_D, which may indicate that experts v_6 and v_7 have not collaborated with each other for a certain period of time (e.g., two years).

In this example, the job search advisor can register a continuous subgraph matching query over G_D to continuously identify subgraphs *of G_D that match with the query graph pattern q. At timestamp t_1, subgraphs g_1 and g_2 are the subgraph matching answers, whereas at timestamp t_2, subgraph g_1 is the only answer (since the deletion of edge (v_6, v_7) invalidates the subgraph g_2).* ■

Continuous subgraph matching over dynamic graph has many other real applications. For example, in the application of anomaly detection for online shopping [49], the interactions (e.g., purchases, views, likes, etc.) among users/products over time form a dynamic social/transaction graph, and it is important to detect some abnormal events or fraudulent activities (i.e., query graph patterns) over such a dynamically changing graph.

Inspired by the examples above, we formulate a *continuous subgraph matching* (CSM) query over a dynamic graph, which continuously monitors subgraph matching results, upon updates to the data graph.

Prior Works: Due to the NP completeness of the subgraph isomorphism [15, 21, 22], continuous subgraph matching is not tractable. Several exact algorithms over dynamic graphs have been proposed [14, 20, 33, 34, 37, 40, 43], which can be classified into 3 categories: *recomputation-based* [20], *direct-incremental* [37], and *index-based incremental algorithms* [14, 33, 34, 40, 43]. The recomputation-based algorithms re-compute all the subgraph matching answers at each timestamp from scratch, the direct-incremental algorithms incrementally calculate new matching results from dynamic graphs upon updates, and index-based incremental ones incrementally maintain matching results via auxiliary indexes built over query answers.

Our Contributions: Our work falls into the second category of the direct-incremental algorithm. Different from existing works that directly use structural information (e.g., vertex label and neighbors' label set) to filter out vertex candidates, we design a novel and effective *DynamIc Vertex DomINance Embedding* (DIVINE) approach for candidate vertex/subgraph retrieval. Specifically, we propose a dynamic vertex dominance embedding technique, which transforms our CSM problem to a *dominating region search* problem in the embedding space with no false dismissals, and allows the generated vertex embeddings to be incrementally maintained in dynamic graphs. To enhance the pruning power of our embeddings for high-degree vertices, we propose a new *degree grouping* approach for vertex embeddings over basic subgraph patterns in different degree groups (i.e., groups of star substructures). We also devise a cost model to further guide the vertex embedding generation to effectively achieve high pruning power. Finally, we develop algorithms to incrementally maintain dynamic graphs and answer CSM queries efficiently.

Specifically, we make the following main contributions:

(1) We formally define the *continuous subgraph matching* (CSM) problem in Section 2, and propose a general framework for CSM query answering in Section 3.
(2) We carefully design incremental vertex dominance embeddings to facilitate CSM processing in Section 4.
(3) We devise a degree grouping technique to enhance the pruning power of vertex embeddings, and construct *degree-aware star substructure synopses* (DAS^3) to support the continuous subgraph matching in Section 5.
(4) We design effective synopsis pruning strategies, and develop an efficient algorithm, named *DynamIc Vertex DomINance Embedding* (DIVINE), for CSM processing in Section 6.

Table 1: Symbols and Descriptions

Symbol	Description
G	a static graph
G_D	a dynamic graph
ΔG_t	a set of graph updates at timestamp t
G_t	a snapshot graph of G_D at timestamp t
$e+$ (or $e-$)	an insertion (or deletion) of edge e
q	a query graph
Q	a set of query graph patterns
x_i	a SPUR vector of vertex v_i
y_i	a SPAN vector of vertex v_i's 1-hop neighbors
$o(v_i)$ (or $o(g_{v_i}), o(s_{v_i})$)	a vertex dominance embedding vector

(5) We present novel cost-model-guided vertex embeddings to further improve the CSM pruning power in Section 7.
(6) We demonstrate through extensive experiments the efficiency and effectiveness of our DIVINE approach over real/synthetic graphs in Section 8.

Section 9 reviews previous works on dynamic graph management and graph embeddings. Finally, Section 10 concludes this paper.

2 Problem Definition

Table 1 depicts the commonly used symbols and their descriptions in this paper.

2.1 Static Graph Model

We give the model of a static undirected, vertex-labeled graph G.

Definition 1. ***(Static Graph,** G**)*** *A static graph, G, is denoted as a triple $(V(G), E(G), L(G))$, where $V(G)$ is a set of vertices v_i, $E(G)$ is a set of edges $e_{i,j} = (v_i, v_j)$ between vertices v_i and v_j, and $L(G)$ is a label function from each vertex $v_i \in V(G)$ to a label $l(v_i)$.* ■

The static graph model (as given in Definition 1) has been widely used in various real-life applications to reflect relationships between different entities, such as social networks [1], Semantic Web [29], transportation networks [50], biological networks [36], citation networks [70], and so on.

The Graph Isomorphism: Next, we define the classic graph isomorphism problem between two graphs.

Definition 2. ***(Graph Isomorphism)*** *Given two graphs G_1 and G_2, graph G_1 is isomorphic to graph G_2 (denoted as $G_1 \equiv G_2$), if there exists a bijection mapping function $M : V(G_1) \Rightarrow V(G_2)$, such that: i) $\forall v_i \in V(G_1)$, we have $L(v_i) = L(M(v_i))$, and; ii) $\forall e_{i,j} \in E(G_1)$, edge $e_{M(v_i),M(v_j)} = (M(v_i), M(v_j)) \in E(G_2)$ holds.* ■

In Definition 2, the graph isomorphism problem checks whether the graphs G_1 and G_2 exactly match each other.

The Subgraph Matching Problem: The subgraph isomorphism (or subgraph matching) problem is defined as follows.

Definition 3. ***(Subgraph Matching)*** *Given graphs G and g, a subgraph matching problem identifies subgraphs g' of G (i.e., $g' \subseteq G$) such that g and g' are isomorphic.* ■

Note that, the subgraph matching problem has been proven to be NP-complete [21].

2.2 Dynamic Graph Model

Real-world graphs are often continuously evolving and updated. Examples of such dynamic graphs include social networks with new or broken friend relationships among users, bibliographical networks with new/discontinued collaboration relationships, and so on. In this subsection, we provide the data model for dynamic graphs with update operations (i.e., edge insertion/deletion) below.

Definition 4. ***(Dynamic Graph,** G_D**)** A dynamic graph, G_D, consists of an initial graph G_0 (as given by Definition 1) and a sequence of graph update operations $\Delta\mathcal{G} = \{\Delta G_1, \Delta G_2, \ldots, \Delta G_t, \ldots\}$ on G. Here, ΔG_t is a graph update operator, in the form of $(e+, t)$ or $(e-, t)$, which indicates either an insertion (+) or a deletion (−) of an edge e at timestamp t, respectively.* ■

Snapshot Graph, G_t**:** From Definition 4, after applying to initial graph G_0 all the graph update operations up to the current timestamp t (i.e., ΔG_1, ΔG_2, . . . , and ΔG_t), we can obtain a snapshot of the dynamic graph G_D, denoted as G_t.

Discussions on the Edge Insertion: For the edge insertion $(e+, t)$ in Definition 4, there are three cases:

- both ending vertices of e exist in the graph snapshot G_{t-1};
- one ending vertex of e exists in G_{t-1} and the other one is a new vertex, and;
- both ending vertices of e are new vertices.

We will later discuss how to deal with these three cases of edge insertions above for dynamic graph maintenance and incremental query answering.

2.3 Continuous Subgraph Matching Queries

In this subsection, we define a *continuous subgraph matching* (CSM) query in a large dynamic graph G_D, which continuously monitors subgraph matching answers for a set, Q, of query graph patterns.

Definition 5. ***(Continuous Subgraph Matching, CSM)** Given a dynamic graph G_D, a set, Q, of registered query graph patterns q, and a timestamp t, a continuous subgraph match (CSM) query maintains a subgraph matching answer set $A(q, t)$ for each query graph pattern $q \in Q$, such that any subgraph $g \in A(q, t)$ in G_D is isomorphic to the query graph q (denoted as $g \equiv q$).*

Alternatively, for each $q \in Q$, the CSM problem incrementally computes an update, $\Delta A(q, t)$, to the answer set $A(q, t-1)$, upon the change ΔG_t to G_{t-1}. ■

In Definition 5, the CSM query continuously maintains a subgraph matching answer set $A(q, t)$ (w.r.t. each query graph pattern $q \in Q$) over dynamic graph G_D (upon incremental updates) for a long period of time. The CSM problem is quite useful in real applications such as cyber-attack event detection and credit card fraud monitoring.

2.4 Challenges

The subgraph matching problem in a data graph has been proven to be NP-complete [21]. Thus, it is even more challenging to design efficient and effective techniques for answering subgraph matching queries in the scenario of dynamic graphs. Due to frequent updates to the data graph, it is non-trivial how to effectively maintain a large-scale dynamic graph that can support efficiently obtaining initial subgraph matching results, and continuously monitor CSM answer sets, upon fast graph updates.

To tackle the CSM problem, in this paper, we will propose a novel and effective vertex dominance embedding technique over dynamic graphs, which transforms the subgraph matching to a dominance search problem in a graph-node-embedding space. Our proposed embedding technique can guarantee the subgraph matching with no false dismissals. Furthermore, we also design effective pruning, indexing, and incremental maintenance mechanisms to facilitate an efficient and scalable continuous subgraph matching algorithm.

Algorithm 1: The DIVINE-based Framework for Continuous Subgraph Matching

Input: i) a dynamic data graph G_D; ii) a set, Q, of registered query graph patterns, and; iii) current timestamp t

Output: the subgraph matching answer set, $A(q, t)$, for each registered query graph $q \in Q$

// **Graph Maintenance Phase**

1 generate a vertex embedding $o(v_i)$ for each vertex $v_i \in G_0$

2 build m synopses, Syn_j, over vertex dominance embeddings with degree grouping

3 **for** *each graph update* $\Delta G_t \in \Delta\mathcal{G}$ **do**

4   obtain G_t by applying the graph update ΔG_t over G_{t-1}

5   maintain synopses Syn_j by updating the corresponding vertex embeddings

// **CSM Query Answering Phase**

// Initial Subgraph Matching Answer Set Generation

6 **for** *each new query graph* $q \in Q$ **do**

7   compute a query embedding vector $o(q_i)$ of each vertex $q_i \in V(q)$

8   find candidate vertex sets $q_i.cand\_set$ for query vertices q_i by accessing synopses Syn_j

9   enumerate candidate subgraphs g from candidate vertices in $q_i.cand\_set$

10   obtain an initial subgraph matching answer set, $A(q, t)$, of matching subgraphs g ($\equiv q$)

// Continuous Subgraph Matching Answer Set Monitoring

11 **for** *each graph update* $\Delta G_t \in \Delta\mathcal{G}$ **do**

12   **for** *each query graph* $q \in Q$ **do**

13   update candidate vertex sets $q_i.cand\_set$ by checking the updated vertices in ΔG_t

14   compute answer changes $\Delta A(q, t)$ from the new candidate sets $q_i.cand\_set$

15   apply changes $\Delta A(q, t)$ to $A(q, t-1)$ and obtain the latest subgraph answers in $A(q, t)$

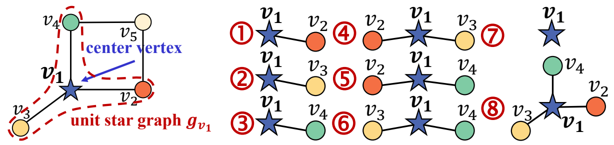
3 Continuous Subgraph Matching Framework

Algorithm 1 illustrates a general framework for continuous subgraph matching based on DynamIc Vertex DomINance Embedding (DIVINE), which consists of two phases, *graph maintenance* and *CSM query answering phases*.

Specifically, in the graph maintenance phase, we first pre-process initial graph G_0, by constructing m graph synopses, Syn_j, over vertex dominance embeddings $o(v_i)$ with m degree groups (discussed later in Section 5), respectively (lines 1-2). Then, for each graph update operation $\Delta G_t \in \Delta\mathcal{G}$, we incrementally maintain the dynamic graph G_t, vertex embeddings, and synopses Syn_j (lines 3-5).

In the CSM query answering phase, for each newly registered query graph q, we obtain the initial query answer set, $A(q, t)$, over the current snapshot G_t of dynamic graph G_D (lines 6-10). Then, we monitor the updates in query answer sets (w.r.t. Q) over continuously changing dynamic graph G_t (lines 11-15).

In particular, for initial answer set generation, we first compute query vertex embeddings $o(q_i)$ from vertices $q_i \in V(q)$ in each query graph q (lines 6-7). Then, for each query vertex q_i, we find and store candidate vertices in a set $q_i.cand\_set$ by accessing Syn_j (line 8). After that, we enumerate candidate subgraphs g by assembling candidate vertices in $q_i.cand\_set$ (line 9), and refine/return matching subgraphs g in an initial subgraph matching answer set $A(q, t)$ (line 10).



(a) unit star subgraph g_{v_1} (b) 8 possible star substructures s_{v_1}

Figure 2: Illustration of unit star subgraph and substructures.

Next, to monitor CSM query answers, for each graph update operation $\Delta G_t \in \Delta\mathcal{G}$ and each query graph $q \in Q$, we check the updated vertices in ΔG_t, update candidate vertex sets $q_i.cand\_set$, and compute the changes $\Delta A(q, t)$ in $A(q, t-1)$ (lines 11-14). Finally, we obtain the CSM query answer set $A(q, t)$ by applying changes $\Delta A(q, t)$ to $A(q, t-1)$ (line 15).

4 Dynamic Vertex Dominance Embeddings

In this section, we will present our dynamic vertex dominance embeddings, which can be incrementally maintained, preserve dominance relationships between basic subgraph patterns (i.e., unit star subgraphs), and support efficient CSM over dynamic graphs.

4.1 Preliminaries and Terminologies

We first introduce two terms of basic subgraph patterns (used for our proposed vertex dominance embedding), that is, *unit star subgraphs* and *star substructures*:

- **Unit Star Subgraph** g_{v_i}**:** A unit star subgraph g_{v_i} is defined as a (star) subgraph in G_t containing a center vertex v_i and its one-hop neighbors.
- **Star Substructure** s_{v_i}**:** A star substructure s_{v_i} is defined as a (star) subgraph of the unit star subgraph g_{v_i} in G_t (i.e., $s_{v_i} \subseteq g_{v_i}$), which shares the same center vertex v_i.

As an example in Figure 2(a), the unit star subgraph g_{v_1} in data graph G_t contains a center vertex v_1 and its 1-hop neighbors v_2, v_3, and v_4. Figure 2(b) shows 8 $(= 2^3)$ possible star substructures s_{v_1} $(\subseteq g_{v_1})$, which are centered at vertex v_1 and with different combinations of v_1's 1-hop neighbors.

Note that, intuitively, star substructures s_{v_i} are potential query star patterns (i.e., vertex $q_i \in V(q)$ and its 1-hop neighbors) in the query graph q of the CSM problem.

4.2 Vertex Dominance Embedding

To tackle the CSM problem, our goal is to avoid costly graph comparisons by transforming them into a search problem over vertex embeddings in a vector space. Thus, we design an embedding method to generate a vector, $o(v_i)$, for each vertex $v_i \in V(G_t)$ based on the labels of vertex v_i and its 1-hop neighbors (i.e., unit star subgraph or star substructure). Specifically, the embedding vector $o(v_i)$ consists of two portions, a *Seeded PseUdo Random* (SPUR) vector x_i and a *SPUR Aggregated Neighbor* (SPAN) vector y_i.

Seeded PseUdo Random (SPUR) Vector, $\boldsymbol{x_i}$**:** In dynamic graph G_D, each vertex v_i is associated with a label $l(v_i)$, which can be encoded by a nonnegative integer. By treating the vertex label $l(v_i)$ as the seed, we can generate a *Seeded PseUdo Random* (SPUR) vector, x_i, of arity d via a pseudo-random number generator $f_1(\cdot)$, that is,

$$x_i = f_1(l(v_i)). \tag{1}$$

We denote $x_i[j]$ as the j-th element in SPUR vector x_i, where $1 \le j \le d$ and $x_i[j] > 0$.

Note that, since our randomized SPUR vector x_i is generated based on the seed (i.e., label $l(v_i)$ of each vertex v_i), vertices with the same label will have the same SPUR vector.

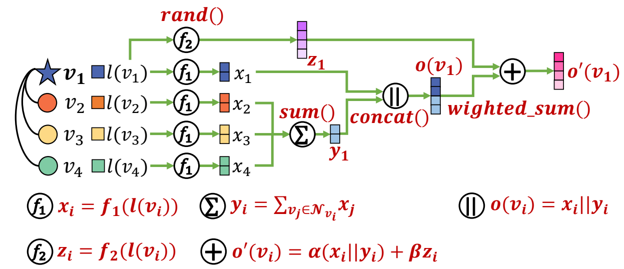


Figure 3: Illustration of vertex dominance embedding $o(v_1)$ **and the optimized vertex dominance embedding** $o'(v_1)$**.**

In the example of Figure 3, vertices $v_1 \sim v_4$ are associated with labels $l(v_1) \sim l(v_4)$, respectively. We generate their corresponding SPUR vectors, $x_1 \sim x_4$, via a randomized function $f_1(\cdot)$ in Eq. (1).

SPUR Aggregated Neighbor (SPAN) Vector, $\boldsymbol{y_i}$**:** For each vertex v_i, we also aggregate structural information from its 1-hop neighbors, $v_j \in \mathcal{N}_{v_i}$, in a star subgraph pattern (i.e., unit star subgraph g_{v_i} or star substructure s_{v_i}), and sum up their corresponding SPUR vectors x_j to obtain a *SPur Aggregated Neighbor* (SPAN) vector y_i:

$$y_i = \sum_{\forall v_j \in \mathcal{N}_{v_i}} x_j, \tag{2}$$

where $\mathcal{N}_{v_i}$ is a set of 1-hop neighbors of v_i, and x_j are the SPUR vectors of 1-hop neighbors of v_i.

Taking Figure 3 as an example, the SPAN vector of the vertex v_1 is calculated by $y_1 = x_2 + x_3 + x_4$.

Vertex Dominance Embedding, $\boldsymbol{o(v_i)}$**:** We define the *vertex dominance embedding* vector, $o(v_i)$, of each vertex v_i, by concatenating its SPUR vector x_i and SPAN vector y_i:

$$o(v_i) = x_i \mid\mid y_i, \tag{3}$$

where $||$ is the concatenation operation of two vectors, and x_i and y_i are given in Eqs. (1) and (2), respectively.

Intuitively, vertex dominance embedding $o(v_i)$ encodes label features of a star subgraph pattern (i.e., g_{v_i} or s_{v_i}) containing center vertex v_i and its 1-hop neighbors. To distinguish vertex dominance embeddings $o(v_i)$ from different star subsgraph patterns, we also use notations $o(g_{v_i})$ or $o(s_{v_i})$.

As shown in Figure 3, the vertex dominance embedding of vertex v_1 can be computed by $o(v_1) = x_1 \mid\mid y_1$.

4.3 Properties of Vertex Dominance Embedding

The Dominance Property of Vertex Embeddings: From Eq. (3), the vertex embedding $o(v_i)$ is a combination of SPUR and SPAN vectors. **Given any unit star subgraph** $\boldsymbol{g_{v_i}}$ **and its star substructure** $\boldsymbol{s_{v_i}}$ **(i.e.,** $\boldsymbol{s_{v_i} \subseteq g_{v_i}}$**), their vertex dominance embeddings always satisfy the *dominance* [11] (or equality) condition:** $\boldsymbol{o(s_{v_i})[j] \le o(g_{v_i})[j]}$**, for all dimensions** $\boldsymbol{1 \le j \le 2d}$, denoted as $o(s_{v_i}) \preceq o(g_{v_i})$ (including the case that $o(s_{v_i}) = o(g_{v_i})$).

The reason for the property above is as follows. Since star patterns s_{v_i} and g_{v_i} have the same center vertex v_i, the vertex embeddings, $o(s_{v_i})$ and $o(g_{v_i})$, for s_{v_i} and g_{v_i}, respectively, must share the same SPUR vector x_i. Moreover, since $s_{v_i} \subseteq g_{v_i}$ holds, 1-hop neighbors of vertex v_i in s_{v_i} and g_{v_i} must satisfy the condition that $\mathcal{N}_{v_i}(s_{v_i}) \subseteq \mathcal{N}_{v_i}(g_{v_i})$. As the SPAN vectors are defined as the summed SPUR aggregates of 1-hop neighbors in $\mathcal{N}_{v_i}(s_{v_i})$ and $\mathcal{N}_{v_i}(g_{v_i})$, respectively, we have the dominance relationship between SPAN vectors $y_i(s_{v_i})$ and $y_i(g_{v_i})$ (i.e., $y_i(s_{v_i}) \preceq y_i(g_{v_i})$).

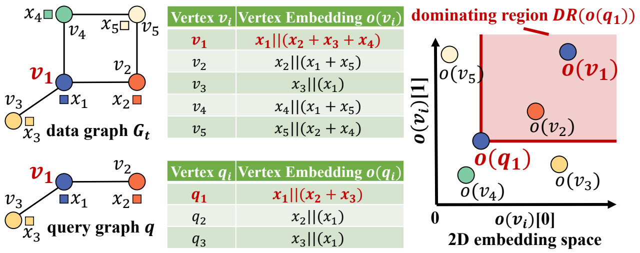


Figure 4: An example of vertex dominance embeddings in subgraph matching ($||$ is the concatenation of two vectors).

Lemma 4.1. *(The Dominance Property of Vertex Embeddings) Given a unit star subgraph g_{v_i} centered at vertex v_i and any of its star substructures s_{v_i} (i.e., $s_{v_i} \subseteq g_{v_i}$), their vertex embeddings satisfy the dominance condition that: $o(s_{v_i}) \preceq o(g_{v_i})$ (including $o(s_{v_i}) = o(g_{v_i})$) in the embedding space.*

Proof. Please refer to Appendix A for the detailed proof. □

The Usage of the Vertex Embedding Property: Note that, a star substructure s_{v_i} ($\subseteq g_{v_i}$) can be a potential query unit star subgraph s_{q_i} from the query graph q (containing a center query vertex q_i and its 1-hop neighbors) for continuous subgraph matching.

During the subgraph matching, given a query embedding vector $o(q_i)$ of query vertex q_i, we can identify those candidate matching data vertices v_i in dynamic graph G_D with embeddings $o(v_i)$ dominated by $o(q_i)$. This way, we can convert the continuous subgraph matching problem into a dominance search problem in the embedding space.

Example 2. *Figure 4 shows an example of our vertex dominance embedding for subgraph matching between a snapshot graph G_t and a query graph q. Based on Eq. (3), we can obtain the embedding for each vertex in G_t or q. For example, vertices v_1 and q_1 have the embeddings $o(v_1) = x_1||(x_2+x_3+x_4)$ and $o(q_1) = x_1||(x_2+x_3)$, respectively.*

Since vertex q_1 in q matches with vertex v_1 in (a subgraph of) G_t, we can see that $o(q_1)$ is dominating $o(v_1)$ in a 2D embedding space (based on the property of vertex dominance embedding), which implies that g_{q_1} is potentially a subgraph of (i.e., matching with) g_{v_1}. Moreover, although $o(v_2)$ is dominated by $o(q_1)$, g_{v_2} does not match g_{q_1}. Thus, in this case, vertex v_2 is a false positive during the subgraph candidate retrieval. On the other hand, since $o(q_1)$ is not dominating $o(v_3)$ in the 2D embedding space, query vertex q_1 cannot match with vertex v_3 in graph G_t. ■

Ease of Incremental Updates for Vertex Dominance Embeddings: In the dynamic graph G_D, our proposed vertex dominance embedding $o(v_i)$ can be incrementally maintained, upon graph update operations in ΔG_t. Specifically, any edge insertion $(e+, t)$ or deletion $(e-, t)$ operation (as given in Definition 4) will affect the embeddings of two ending vertices, v_i and v_j, of edge $e = (v_i, v_j)$. We consider incremental update of the embedding vector $o(v_i)$ for vertex v_i as follows.

- When v_i is a newly inserted vertex, we obtain its embedding vector $o(v_i)$ from scratch (i.e., $o(v_i) = x_i||x_j$), and;
- When v_i is an existing vertex in dynamic graph G_D, for an insertion (or deletion) operator of edge $e = (v_i, v_j)$, we update the SPAN vector, y_i, in $o(v_i)$ with $(y_i + x_j)$ (or $(y_i - x_j)$), where x_j is the SPUR vector of vertex v_j.

The case of updating the embedding of vertex v_j is similar and thus omitted.

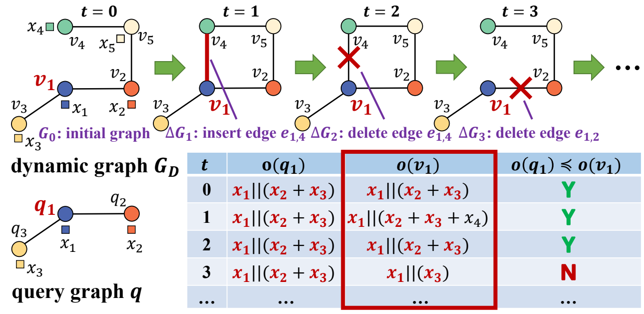


Figure 5: An example of vertex embeddings for CSM.

Note that, for each graph update (i.e., edge insertion $(e+, t)$ or deletion $(e-, t)$), the time complexity of incrementally updating vertex dominance embedding vectors is given by $O(d)$, where d is the dimension of SPUR/SPAN vectors.

Example 3. *Figure 5 illustrates an example of using our vertex dominance embeddings to conduct the subgraph matching over dynamic graph G_D over time t from 0 to 3, where q is a given query graph. Consider two vertices v_1 and q_1 in graphs G_D and q, respectively. At timestamp $t = 0$, their embeddings satisfy the dominance relationship, that is, $o(q_1) \preceq o(v_1)$, thus, vertex v_1 is a candidate matching with q_1.*

At timestamp $t = 1$, a new edge $e_{1,4}$ is inserted (i.e., ΔG_1), and the vertex embedding $o(v_1)$ changes from $x_1||(x_2+x_3)$ to $x_1||(x_2+x_3+x_4)$ (by including new neighbor v_4's SPUR vector x_4). Since $o(q_1) \preceq o(v_1)$ still holds, v_1 remains a candidate matching with q_1.

At timestamp $t = 2$, since edge $e_{1,4}$ is deleted (i.e., ΔG_2), the vertex embedding $o(v_1)$ changes back to $x_1||(x_2+x_3)$. At timestamp $t = 3$, edge $e_{1,2}$ is deleted (i.e., ΔG_3), and $o(v_1)$ is updated with $x_1||(x_3)$ (by removing the expired neighbor v_2's SPUR vector x_2). In this case, $o(q_1) \preceq o(v_1)$ does not hold, and vertex v_1 fails to match with q_1. ■

4.4 Embedding Optimization with Base Vector z_i

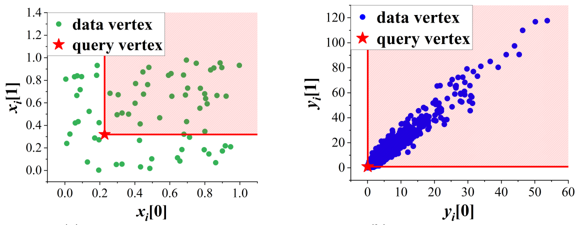
Figure 6 shows the distributions of 2D SPUR and SPAN vectors, x_i and y_i, respectively, in vertex dominance embedding $o(v_i)$ over Yeast graph [58] (with 3,112 vertices and 12,519 edges). We can see that, in Figure 6(a), the SPUR vectors x_i generated by a seeded randomized function are distributed uniformly in the embedding space, whereas the SPAN vectors (i.e., the summed SPUR vectors) y_i are more clustered along the reverse diagonal line in Figure 6(b).

As mentioned in Section 4.3, given a query vertex q_i, we can use its embedding vector $o(q_i)$ as a query point to find the dominated embedding vectors $o(v_i)$ ($= x_i \,||\, y_i$) in the 4D embedding space. Due to the scattered SPUR/SPAN vectors in the embedding space, it is very likely that some false alarms of embedding vectors $o(v_i)$ are included as candidate matching vertices (i.e., dominated by $o(q_i)$).

In order to further enhance the pruning power, our goal is to update our vertex dominance embedding vector from $o(v_i)$ to $o'(v_i)$, such that the number of candidate vertices dominated by $o(q_i)$ is reduced (i.e., as small as possible). Specifically, we propose to use an optimized embedding $o'(v_i)$, given by:

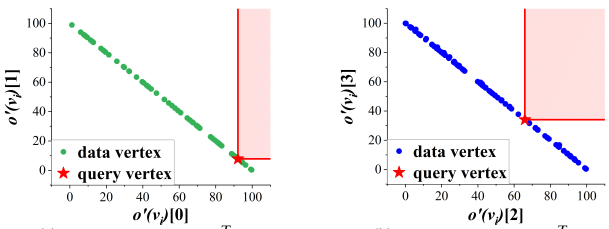
$$o'(v_i) = \alpha(x_i||y_i) + \beta z_i, \tag{4}$$

where α and β are positive constants ($\alpha \ll \beta$), and $z_i = f_2(l(v_i))$ is a random *base vector* of size $(2d)$, generated by a pseudo-random function $f_2(\cdot)$ with $l(v_i)$ as the seed. Please refer to Figure 3 on how to compute this optimized vertex dominance embedding.



(a) SPUR vectors x_i (b) SPAN vectors y_i

Figure 6: An example of SPUR/SPAN vector distributions in vertex dominance embedding $o(v_i)$.



(a) $\alpha x_i + \beta(z_i[0], z_i[1])^T$ (b) $\alpha y_i + \beta(z_i[2], z_i[3])^T$

Figure 7: An example of $o'(v_i)$ distributions with L_1-norm diagonal-line base vector z_i ($\alpha = 0.01, \beta = 100$).

Note that, since the base vector z_i is generated only based on the (seed) label $l(v_i)$ of center vertex v_i, different star subgraphs with the same center vertex label $l(v_i)$ will result in the same base vector z_i. Therefore, the new (optimized) vertex embedding vector $o'(v_i)$ in Eq. (4) still follows the dominance relationship, that is, $o'(q_i) \preceq o'(v_i)$ holds, if a query star pattern s_{q_i} is a subgraph of unit star subgraph g_{v_i} (i.e., $s_{q_i} \subseteq g_{v_i}$ and $l(q_i) \equiv l(v_i)$).

Lemma 4.2. *(The Dominance Property of the Optimized Vertex Dominance Embeddings) Given a unit star subgraph g_{v_i} centered at vertex v_i and any of its star substructures s_{v_i} (i.e., $s_{v_i} \subseteq g_{v_i}$), their optimized vertex dominance embeddings (via base vector z_i) satisfy the condition that: $o'(s_{v_i}) \preceq o'(g_{v_i})$ (including $o'(s_{v_i}) = o'(g_{v_i})$) in the embedding space.*

Proof. Please refer to Appendix A for the detailed proof. □

Discussions on How to Design a Base Vector, z_i: To improve the pruning power, we would like to make the distribution of the updated vertex embeddings in Eq. (4) more dispersed along diagonal line (or plane), so that fewer false alarms (dominated by $o'(q_i)$) can be obtained during continuous subgraph matching.

In this paper, we propose to design the base vector z_i as a random vector distributed on the unit diagonal line/hyperplane (L_1-norm) in the first quadrant. That is, we can normalize z_i to $\frac{z_i}{||z_i||_1}$, where $||\cdot||_1$ is L_1-norm. Moreover, since constant α is far smaller than β, $\alpha(x_i||y_i)$ can be considered as noise added to the scaled base vector βz_i. Thus, vertex embedding vectors $o'(v_i)$ ($= \alpha(x_i||y_i) + \beta z_i$) in Eq. (4) are still distributed close to a diagonal line (or hyperplane) in the first quadrant.

We would like to leave interesting topics of using other base vectors z_i (e.g., L_2-norm) as our future work.

Figure 7 shows the distributions of the updated 2D SPUR and SPAN vectors (from $o'(v_i)$), by adding a base vector z_i, where z_i is given by L_1-norm. Since we have $\alpha \ll \beta$ in Eq. (4), after adding $\alpha(x_i||y_i)$ to the scaled base vector βz_i, the previous embedding vector $o(v_i)$ (as shown in Figure 6) is re-located close to diagonal lines in Figure 7. This way, we can reduce the number of vertex false alarms dominated by $o(q_i)$ in both SPUR and SPAN spaces, and in turn enhance the CSM pruning power.

5 Degree-Aware Star Substructure Synopses

We observed that high-degree center vertices of star substructures tend to have vertex dominance embeddings with low pruning power during CSM processing. Therefore, in this section, we will classify star substructures into different degree groups, and build synopses over their embeddings under degree groups. This way, CSM can be conducted over a synopsis corresponding to a group of lower degrees, which can further improve the pruning power.

5.1 Vertex Embedding via Degree Grouping

Rationale behind the Degree Grouping: During the subgraph matching over a dynamic graph G_t, those vertices $v_i \in V(G_t)$ with high degrees $deg(v_i)$ are less likely to be pruned in the embedding space. This is because, for high degrees, the SPAN vectors in $o'(v_i)$ tend to have large values on all dimensions. Thus, $o'(v_i)$ also tends to be dominated by query embedding vector $o'(q_i)$, and treated as candidate vertices for the refinement (i.e., v_i cannot be pruned).

Therefore, to enhance the pruning power for high-degree vertices, in this subsection, we propose a novel and effective *degree grouping* technique, which divides all possible star substructures s_{v_i} of each vertex v_i in G_D (that may match with a query star pattern s_{q_i}) into different groups (corresponding to different degree intervals of center vertices v_i in star substructures s_{v_i}). Intuitively, during the CSM query processing, the embedding vector $o'(q_i)$ of query star pattern s_{q_i} only needs to be compared with those of star substructures s_{v_i} with similar degrees in a group, rather than all star substructures (including those with high degrees), which can greatly improve the pruning power.

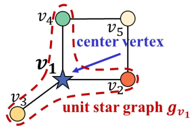
Equi-Frequency Degree Grouping: One straightforward way to perform the degree grouping is to let each single degree value be a degree group. However, since vertex degrees in real-world graphs usually follow the power-law distribution [2, 5, 6, 16, 45], only a small fraction of vertices have high degrees. In other words, only a few unit star subgraphs g_{v_i} have star substructures s_{v_i} of very high degrees, which is not space- and time-efficient to maintain in a dynamic graph G_t.

Inspired by this, in this paper, we propose to maintain vertex embeddings for star substructures with m degree groups (of different degree intervals). We assume that the degree statistics of the initial graph G_0 are similar to those in G_t.

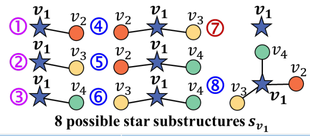
This way, we obtain the *probability mass function* of vertex degrees in G_0 is $pmf(\delta) = \frac{freq(deg(v_i) \geq \delta)}{|V(G_0)|}$, where $freq(deg(v_i) \geq \delta)$ is the number of vertices $v_i \in V(G_0)$ whose degrees are greater than or equal to a degree value δ. Our goal is to divide the degree interval, $(0, deg_{max}]$, into m degree groups $\mathcal{B} = \{B_1, B_2, \cdots, B_m\}$, such that each *degree bucket* B_j ($= (\delta_{j-1}, \delta_j]$) contains the same (or similar) mass in the distribution of $pmf(\delta)$ (for $\delta \in B_j$).

Example 4. *Figure 8 illustrates the degree grouping process for star substructures with center vertex v_1, where 7 possible star substructures $s_{v_1}^i$ (excluding Case ⑦, the isolated vertex v_1) are partitioned into two groups with 2 degree intervals, $B_1 = (0, 1]$ and $B_2 = (1, 3]$, respectively. For example, in Case ①, center vertex v_1 in star substructure $s_{v_1}^1$ has degree 1, which thus falls into the first degree bucket $B_1 = (0, 1]$.*

A query star pattern s_{q_1} with center vertex q_1 of degree 1 will be compared with star substructures in degree bucket $B_1 = (0, 1]$ only. In contrast, for a query star pattern s_{q_1} with degree equal to 2 or 3, we only need to search the second degree bucket B_2 (with interval $(1, 3]$).



Vertex v_i	SPUR Vector x_i
v_1	74
v_2	21
v_3	94
v_4	58
v_5	45



Degree Group / Synopsis	Star Substructure $s^i_{v_1}$	$deg(v_1)$	Vertex Embedding $o'(v_1) = \alpha(\text{SPUR } x_1 \| \text{SPAN } y_1) + \beta z_1$	MBRs in DAS³ Synopsis
$B_1 = (0, 1]$ (Synopsis Syn_1)	① $s^1_{v_1}$	1	$\alpha(x_1 \| x_2) + \beta z_1 = [36.74, 64.21]$	$v_1.\text{MBR}_1 = [(36.74, 36.74); (64.21, 64.94)]$
	② $s^2_{v_1}$	1	$\alpha(x_1 \| x_3) + \beta z_1 = [36.74, 64.94]$	
	③ $s^3_{v_1}$	1	$\alpha(x_1 \| x_4) + \beta z_1 = [36.74, 64.58]$	
$B_2 = (1, 3]$ (Synopsis Syn_2)	④ $s^4_{v_1}$	2	$\alpha(x_1 \| (x_2 + x_3)) + \beta z_1 = [36.74, 65.15]$	$v_1.\text{MBR}_2 = [(36.74, 36.74); (64.79, 65.52)]$
	⑤ $s^5_{v_1}$	2	$\alpha(x_1 \| (x_2 + x_4)) + \beta z_1 = [36. 74, 64.79]$	
	⑥ $s^6_{v_1}$	2	$\alpha(x_1 \| (x_3 + x_4)) + \beta z_1 = [36.74, 65.52]$	
	⑧ $s^8_{v_1}$	3	$\alpha(x_1 \| (x_2 + x_3 + x_4)) + \beta z_1 = [36.74, 65.73]$	$v_1.\text{MBR}_3 = [(36.74, 36.74); (65.73, 65.73)]$

Figure 8: An example of degree grouping and degree-aware star substructure synopses ($\alpha = 0.01, \beta = 100, z_1 = (0.36, 0.64)$).

5.2 The Construction of Degree-Aware Star Substructure Synopses (DAS³)

Based on the degree grouping $\mathcal{B} = \{B_1, B_2, \cdots, B_m\}$, we will construct m *degree-aware star substructure synopses* (DAS³), Syn_j (for $1 \le j \le m$), which are essentially m grid files [24], respectively. In each grid synopsis Syn_j, we store data vertices in different cells according to their (aggregated) vertex embeddings of star substructures in the corresponding degree group B_j.

Data Structure of DAS³ Synopses: Specifically, each DAS³ synopsis Syn_j partitions the embedding data space into cells, C, of equal size. Each grid cell C is associated with a list, $C.list$, of vertices v_i. Each vertex $v_i \in C.list$ contains the following aggregates:

- an *embedding upper bound* vector, $v_i.UB_\delta$, for all star substructures s_{v_i} with center vertex degrees $\delta \in (\delta_{j-1}, \delta_j]$, and;
- a list of MBRs, $v_i.MBR_\delta$ (for $\delta \in (\delta_{j-1}, \delta_j]$), that minimally bound all embedding vectors $o'(v_i)$ of star substructures s_{v_i} with degrees equal to δ.

To facilitate the access of cells in DAS³ synopsis Syn_j efficiently, we sort all cells in descending order of their keys $C.key$. Here, for each cell C, we use $C.UB$ to denote its maximal corner point (i.e., taking the maximum value of the cell interval on each dimension). The key, $C.key$, is defined as $||C.UB||_2$, where $||Z||_2 = \sum_{i=1}^{d} Z[i]^2$ for a d-dimensional vector Z. Note that, if a query vertex embedding $o(q_i)$ dominates $C.UB$, then the key $||o(q_i)||_2$ of query vertex embedding $o(q_i)$ must be smaller than or equal to the cell key $C.key$. Thus, with such a sorted list of cells, we can efficiently access all cells that are fully or partially dominated by a given query vertex embedding $o(q_i)$.

DAS³ Synopsis Construction: Each DAS³ synopsis Syn_j corresponds to a degree group B_j ($1 \le j \le m$) with degree interval $(\delta_{j-1}, \delta_j]$. We build the DAS³ synopsis Syn_j as follows. For each vertex v_i with degree $deg(v_i) > \delta_{j-1}$ in G_t and for each degree value δ ($\in (\delta_{j-1}, \delta_j]$), we obtain all its star substructures, s_{v_i}, with degrees of center vertex v_i equal to δ, compute their embedding vectors $o'(s_{v_i})$, and use a *minimum bounding rectangle* (MBR), $v_i.MBR_\delta$, to minimally bound these embedding vectors.

Let $v_i.UB_\delta$ be the maximum corner point of the MBR $v_i.MBR_\delta$ (by taking the upper bound of the MBR on each dimension). Then, for vertex v_i, we insert vertex v_i into the vertex list, $C.list$, of a cell $C \in \mathcal{I}_j$, into which corner point $v_i.UB_{ub\_\delta}$ falls, where degree upper bound $ub\_\delta = \min\{deg(v_i), \delta_j\}$. Intuitively, this corner point $v_i.UB_{ub\_\delta}$ is the one most likely to be dominated by a query vertex embedding $o'(q_i)$ in this degree group B_j.

Moreover, we also associate corner point $v_i.UB_{ub\_\delta}$ with the list of MBRs, $v_i.MBR_\delta$ (for each $\delta \in (\delta_{j-1}, ub\_\delta]$), in synopsis Syn_j, which can be used for further pruning for a specific degree in interval $(\delta_{j-1}, \delta_j]$.

Discussions on MBR Computations and Incremental Maintenance: Note that, the MBRs, $v_i.MBR_\delta$, mentioned above minimally bound embedding vectors $o'(v_i)$ for all star substructures s_{v_i}. Since all possible star structures of v_i share the same SPUR vector x_i in $o'(v_i)$, we can directly assign the lower and upper bounds of the k-th dimension in the MBR (i.e., $v_i.MBR_\delta[2k]$ and $v_i.MBR_\delta[2k+1]$, for $0 \le k < d$) as follows:

$$v_i.MBR_\delta[2k] = v_i.MBR_\delta[2k+1] = o'(v_i)[k] = \alpha x_i[k] + \beta z_i[k]. \quad (5)$$

For the SPAN vector part in the MBR, since there are an exponential number of possible star structures (i.e., $2^{deg(v_i)}$), it is not efficient to enumerate all star substructures, compute their embedding vectors, and obtain the MBRs. Therefore, in this paper, we propose a sorting-based method to obtain $v_i.MBR_\delta$ without enumerating all star substructures with degree δ.

Specifically, for each vertex v_i, we maintain d sorted lists, each of which, $v_i.spur\_list_k$ (for $0 \le k < d$), contains the k-th elements, $x_j[k]$, of SPUR vectors x_j (obtained from v_i's 1-hop neighbors $v_j \in \mathcal{N}(v_i)$) in ascending order.

For a degree δ in the degree group $B_j = (\delta_{j-1}, \delta_j]$, we obtain lower and upper bounds of MBR $v_i.MBR_\delta$, that is, $v_i.MBR_\delta[2k]$ and $v_i.MBR_\delta[2k+1]$, respectively, as follows:

$$v_i.MBR_\delta[d + 2k] = \sum_{deg \in [1, \delta]} v_i.spur\_list_k[deg], \quad (6)$$

$$v_i.MBR_\delta[d + 2k + 1] = \sum_{deg \in [deg(v_i) - \delta + 1, deg(v_i)]} v_i.spur\_list_k[deg], \quad (7)$$

where $deg(v_i)$ is the degree of vertex v_i in G_t (or the size of the sorted list $v_i.spur\_list_k$).

Intuitively, Eqs. (6) and (7) give the lower/upper bounds of the MBR $v_i.MBR_\delta$ on the $(d + k)$-th dimension, by summing up the first δ (smallest) and the last δ (largest) values, respectively, in the sorted list $v_i.spur\_list_k$.

The Time Complexity of the MBR Maintenance: The time complexity of sorting the k-th dimension of SPUR vectors (for $1 \le k \le d$) is given by $O(d \cdot deg(v_i) \cdot log(deg(v_i)))$, which is much less than that of enumerating all star substructures (i.e., $O(d \cdot 2^{deg(v_i)})$).

For the maintenance of a sorted list $v_i.spur\_list_k$ in dynamic graph G_t, upon edge insertion or deletion (or insertion/deletion of a 1-hop neighbor v_j), we can use a binary search to locate where we need to insert into (or remove from) $v_i.spur\_list_k$ elements $y_j[k]$ in the SPUR vector y_j. Therefore, the time complexity of incrementally maintaining d sorted lists $v_i.spur\_list_k$ is given by $O(d \cdot log_2(deg(v_i)))$.

Example 5. *In the example of Figure 8, we construct 2 DAS³ synopses, Syn_1 and Syn_2, over embeddings $o'(v_1)$ of star substructures $s^i_{v_1}$ falling into degree groups $B_1 = (0, 1]$ and $B_2 = (1, 3]$, respectively. For example, in grid synopsis Syn_1, we store vertex v_1 in a cell C, which contains embedding vectors $o'(v_1)$ of star structures $s^i_{v_1}$ (for $1 \le i \le 3$) under degree group B_1. Vertex v_1 is also associated with an MBR $v_1.MBR_1$ which minimally bounds embedding vectors $o'(v_1)$ of all star substructures $s^i_{v_1}$ in degree bucket B_1.*

6 CSM Query Processing

In this section, we will illustrate query processing algorithms based on vertex dominance embeddings in DAS^3 synopses to efficiently answer CSM queries.

6.1 Synopsis Pruning Strategies

Given a query vertex q_i in the query graph q and SAS^3 synopses Syn_j, we would like to obtain candidate vertices v_i from synopsis Syn_j which may match with the query vertex q_i, where $deg(q_i)$ falls into the degree group $B_j = (\delta_{j-1}, \delta_j]$.

In this subsection, we present two effective pruning methods over index Syn_j, named *embedding dominance pruning* and *MBR range pruning*, which are used to rule out false alarms of cells/vertices.

Embedding Dominance Pruning: We first provide the embedding dominance pruning method, which filters out those cells/vertices in synopsis Syn_j that are not dominated by a given query embedding vector $o'(q_i)$.

Lemma 6.1. **(Embedding Dominance Pruning)** *Given a query embedding vector $o'(q_i)$ of the query vertex q_i, any cell C or vertex v_i can be safely pruned, if $o'(q_i)$ does not dominate any portion of cell C or embedding upper bound vector $v_i.UB_\delta$.*

Proof. Please refer to Appendix A for the detailed proof. □

MBR Range Pruning: Next, we give an effective *MBR range pruning* method, which further utilizes the MBR ranges, $v_i.MBR_\delta$, of star substructure embeddings and prunes vertices v_i that do not fall into the MBRs.

Lemma 6.2. **(MBR Range Pruning)** *Given a query embedding $o'(q_i)$ of the query vertex q_i and a vertex v_i in a cell of DAS^3 synopsis Syn_j (for $deg(q_i) \in (\delta_{j-1}, \delta_j]$), vertex v_i can be safely pruned, if it holds that $o'(q_i) \notin v_i.MBR_{deg(q_i)}$.*

Proof. Please refer to Appendix A for the detailed proof. □

6.2 Initial CSM Answer Set Generation

Algorithm 2 illustrates the algorithm of generating the initial subgraph matching answer set, $A(q,t)$, over a snapshot G_t of a dynamic subgraph G_D, by traversing synopses over vertex dominance embeddings. Specifically, given a query graph q, we first obtain vertex dominance embeddings $o'(q_i)$ for each query vertex $q_i \in V(q)$ (line 1), and then traverse DAS^3 synopses, Syn_j, to retrieve vertex candidate sets $q_i.cand\_set$ (lines 2-8). Next, we generate a query plan Q, which is an ordered list of connected query vertices that can be used to join their corresponding candidate vertices (lines 9-10). Finally, we assemble candidate vertices into candidate subgraphs, and refine candidate subgraphs by the *left-deep join based method* [37, 53] by invoking function Refinement(·), and return the initial subgraph matching answers in $A(q,t)$ (lines 11-13).

The Synopsis Search: For each query vertex $q_i \in V(q)$, we first need to find a synopsis Syn_j that matches the degree group of q_i (i.e., $deg(q_i) \in (\delta_{j-1}, \delta_j]$ must hold; line 3). As mentioned in Section 5.2, cells, C, in the DAS^3 synopsis Syn_j are sorted in descending order of their keys $C.key$. We thus only need to search for those cells $C \in Syn_j$ satisfying the condition that $C.key \geq ||o'(q_i)||_2$ (line 4). Then, for each candidate vertex $v_i \in C.list$, we apply our proposed embedding dominance and MBR range pruning strategies (as discussed in Lemmas 6.1 and 6.2, respectively), and obtain the candidate vertex set, $q_i.cand\_set$, for the query vertex q_i (lines 5-8).

Algorithm 2: Initial CSM Answer Set Generation

Input: i) a snapshot graph G_t at current timestamp t; ii) a DAS^3 synopsis Syn_j on G_t, and; iii) a query graph q

Output: a set, $A(q,t)$, of initial subgraph matching results in G_t

1 compute vertex dominance embeddings $o'(q_i)$ for all query vertices $q_i \in V(q)$

`// traverse synopses` Syn_j `to obtain candidate vertices`

2 **for** *each query vertex* $q_i \in V(q)$ **do**

3   obtain a synopsis Syn_j such that $deg(q_i) \in (\delta_{j-1}, \delta_j]$

4   **for** *each cell* $C \in Syn_j$ *with* $C.key \geq ||o'(q_i)||_2$ **do**

5   **for** *each candidate vertex* $v_i \in C.list$ **do**

6   **if** $o'(q_i) \preceq v_i.UB_\delta$ **then**

7   **if** $o'(q_i) \in v_i.MBR_{deg(q_i)}$ **then**

8                     $q_i.cand\_set \leftarrow q_i.cand\_set \cup \{v_i\}$

                        `// Lemmas 6.1 and 6.2`

`// generate a query plan` Q

9 select a query vertex q_i with the smallest candidate set size $|q_i.cand\_set|$ as the first query vertex in an ordered list Q

10 iteratively select a neighbor q_k of $q_i \in Q$ with minimum $|q_k.cand\_set|$ and append q_k to Q, until all query vertices in $V(q)$ have been added to Q

`// assemble and refine candidate subgraphs`

11 $A(q,t) \leftarrow \emptyset$

12 invoke Refinement$(q, Q, G_t, A(q,t), \emptyset, 0)$ to obtain actual CSM query answers in $A(q,t)$

13 **return** $A(q,t)$;

Query Plan Generation: After obtaining candidate vertices, we generate a query plan Q for refinement. Intuitively, we would like to reduce the size of intermediate join results. Therefore, we first initialize Q with a query vertex $q_i \in V(q)$ having the minimum candidate set size $|q_i.cand\_set|$ (line 9), and then iteratively add to Q the one, q_k, that is connected with those selected query vertices in Q and having the smallest $|q_k.cand\_set|$ value (line 10).

Refinement: Algorithm 3 uses a *left-deep join based method* [37, 53] to assemble candidate vertices in $q_i.cand\_set$ into candidate subgraphs, and obtain the actual matching subgraph answer set $A(q,t)$ (i.e., lines 11-13 of Algorithm 2). Please refer to Appendix B for a detailed discussion.

After the recursive function Refinement(·) has been executed, the answer set A (or $A(q,t)$ in Algorithm 3) will contain a set of actual subgraph matching results.

Complexity Analysis: The time complexity of Algorithm 2 is given by: $O(|V(q)|+|E(q)|+|V(q)|\cdot(1-PP_C)K^d\cdot(1-PP_v)|C.list|+\prod_{i=0}^{|V(q)|-1}|q_i.cand\_set|+|V(q)|^2)$, where K^d is the number of cells, $|C.list|$ is the number of vertices in cell C, PP_C is the pruning power of the cells' key value and PP_v is the pruning power in each cell's point list. Please refer to Appendix C for a detailed discussion.

6.3 CSM Query Answering

Given a dynamic graph G_{t-1} at timestamp $(t-1)$, a registered query graph q, and a graph update operation ΔG_t at timestamp t, Algorithm 4 provides a CSM query answering algorithm, which maintains/obtains the latest subgraph matching answer set, $A(q,t)$. Specifically, if the graph update operation ΔG_t is an insertion of edge $e = (v_i, v_j)$, we will calculate an incremental update, $\Delta A(q,t)$, of new subgraph matching results (lines 1-11).

Algorithm 3: Refinement

Input: i) a query graph q; ii) a sorted list (query plan) Q; iii) a snapshot graph G_t; iv) matching results set A; v) a sorted list, M, of vertices matching with query vertices in Q, and; vi) the recursion depth n

Output: a set, A, of subgraph matching results

if $n = |Q|$ **then**
  $A \leftarrow A \cup \{M\}$
  return;
else
  $S_{cand} = \emptyset$
  if $n = 0$ **then**
    $S_{cand} \leftarrow S_{cand} \cup Q[n].cand\_set$
  else
    for *each candidate vertex* $u \in Q[n].cand\_set$ *and* $u \notin M$ **do**
      if *edges* $(M[i], u)$ *exist in* $E(G_t)$ *for all edges* $(Q[i], Q[n]) \in E(q)$ *(for* $0 \leq i < n$*)* **then**
        $S_{cand} \leftarrow S_{cand} \cup \{u\}$
for *each candidate vertex* $u \in S_{cand}$ **do**
  $M[n] = u$
  Refinement $(q, Q, G_t, A, M, n+1)$;

Edge Insertion: For the insertion of an edge $e = (v_i, v_j)$, we will first find matching edge(s) (q_i, q_j) in the query graph with the same labels (line 3). Then, we will check whether or not edges (v_i, v_j) and (q_i, q_j) match each other, by applying the *embedding dominance* and *MBR range pruning* strategies (described in Section 6.1; lines 4-5). If the answer is yes, then we will generate a new query plan Q' starting from query vertices q_i and q_j (note: the remaining ones are obtained from Q, similar to line 10 of Algorithm 2; line 7), initialize the first two matching vertices v_i and v_j in a sorted list M' (lines 8-9), and invoke the function Refinement$(\cdot)$ with parameters of new query plan Q', incremental answer set $\Delta A(q,t)$, initialized sorted list M', and recursion depth $n = 2$ (as two matching vertices have been found; line 10). After that, we can update the answer set $A(q,t)$ with $A(q, t-1) \cup \Delta A(q,t)$ (line 11).

Edge Deletion: When ΔG_t is an edge deletion operation $(e-, t)$, we can simply remove those existing subgraph matching answers $g \in A(q, t-1)$ which contain the deleted edge e, and obtain the updated answer set $A(q,t)$ (lines 12-16).

Finally, we return the latest CSM query answer set $A(q,t)$ at timestamp t (line 17).

Complexity Analysis: In Algorithm 4, for the edge insertion operation, we check all query edges, generate a new query plan Q', and refine new candidate subgraphs with the new edge (lines 1-11). Therefore, the worst-case time complexity is given by $O((|V(q)|^2 + \prod_{i=2}^{|V(q)|-1} |Q'[i].cand\_set|) \cdot |E(q)|)$.

For the edge deletion operation (lines 12-16), since we delete matching results from $A(q, t-1)$ that contain the deleted edge, by using the hash file to check the edge existence with $O(1)$ cost, the time complexity is given by $O(|A(q, t-1)|)$.

7 Cost-Model-Guided Embedding Optimization

In this section, we will design a novel cost model for evaluating the pruning power of our proposed vertex dominance embedding technique. Then, we will utilize this cost model to guide the vertex

Algorithm 4: The Query Answering Algorithm for Continuous Dynamic Subgraph Matching Query

Input: i) a dynamic graph G_{t-1} at previous timestamp $(t-1)$; ii) a graph update operation ΔG_t; iii) a registered query graph q, and; iv) matching result set $A(q, t-1)$ at timestamp $(t-1)$

Output: an updated set, $A(q,t)$, of subgraph matching results at timestamp t

// edge insertion operation $\Delta G_t = (e+, t)$
1 **if** $\Delta G_t = (e+, t)$ *for edge* $e = (v_i, v_j)$ **then**
2   $\Delta A(q,t) \leftarrow \emptyset$
3   **for** *each query edge* $(q_i, q_j) \in E(q)$ *with labels matching with that of* (v_i, v_j) **do**
    // check whether (v_i, v_j) matches (q_i, q_j)
4   **if** $o'(q_i) \leq o'(v_i)$ *and* $o'(q_j) \leq o'(v_j)$ **then**
5   **if** $o'(q_i) \in v_i.MBR_{deg(q_i)}$ *and* $o'(q_j) \in v_j.MBR_{deg(q_j)}$ **then**
          // Lemmas 6.1 and 6.2
6   generate a query plan Q' by using q_i and q_j as the first two query vertices and the remaining ones from Q
7         $M'[0] = v_i, M'[1] = v_j$
8   invoke Refinement$(q, Q', G_t, \Delta A(q,t), M', 2)$ to incrementally obtain new matching results in $\Delta A(q,t)$
10   $A(q,t) \leftarrow A(q, t-1) \cup \Delta A(q,t)$
11 **else**
   // edge deletion operation $\Delta G_t = (e-, t)$
12   $A(q,t) = A(q, t-1)$
13   **for** *each subgraph answer* $g \in A(q, t-1)$ **do**
14   **if** *edge* (v_i, v_j) *exists in* $E(g)$ **then**
15       $A(q,t) \leftarrow A(q,t) - \{g\}$
16 **return** $A(q,t)$

dominance embedding generation and achieve better performance of continuous subgraph matching.

7.1 Cost Model for Continuous Subgraph Matching via Dynamic Vertex Dominance Embeddings

In this subsection, we will propose a cost model to evaluate the performance of our continuous subgraph matching (or the pruning power of using vertex dominance embeddings).

Specifically, we estimate the number, $Cost_{CSM}$, of candidate vertices v_i (to retrieve and refine) whose embedding vectors $o'(v_i)$ are dominated by query embedding vector $o'(q_i)$ (given by Lemma 6.1). That is, we have:

$$Cost_{CSM} = \sum_{v_i \in V(G_t)} \prod_{j=1}^{2d} Pr\{o'(q_i)[j] \leq o'(v_i)[j]\}, \tag{8}$$

where $2d$ is the dimension of the embedding vector $o'(\cdot)$, and $Pr\{\cdot\}$ is a probability function.

Analysis of the Cost Model: We consider $o'(v_i)[j]$ as a random number generated from a random variable, with mean $\mu_{o'(v_i)[j]}$ and variance $\sigma^2_{o'(v_i)[j]}$. Moreover, $o'(q_i)[j]$ can be considered as a constant. We have the following equation:

$$\Pr\{o'(q_i)[j] \leq o'(v_i)[j]\}$$

$$= \Pr\left\{ \frac{(o'(q_i)[j] - o'(v_i)[j]) - (o'(q_i)[j] - \mu_{o'(v_i)[j]})}{\sigma^2_{o'(v_i)[j]}} \le \frac{-(o'(q_i)[j] - \mu_{o'(v_i)[j]})}{\sigma^2_{o'(v_i)[j]}} \right\}. \quad (9)$$

By applying *Central Limit Theorem* (CLT) [69] to Eq. (9), we have:

$$Pr\{o'(q_i)[j] \le o'(v_i)[j]\} \approx \Phi\left(\frac{\mu_{o'(v_i)[j]} - o'(q_i)[j]}{\sigma^2_{o'(v_i)[j]}}\right), \quad (10)$$

where $\Phi(\cdot)$ is the *cumulative density function* (cdf) of *standard normal distribution.*

We substitute Eq. (10) into Eq. (8) and obtain:

$$Cost_{CSM} = |V(G_t)| \cdot \prod_{j=1}^{2d} \Phi\left(\frac{\mu_{o'(v_i)[j]} - o'(q_i)[j]}{\sigma^2_{o'(v_i)[j]}}\right). \quad (11)$$

7.2 Cost-Model-Based Vertex Dominance Embedding Design

Eq. (11) estimates the (worst-case) query cost for one query vertex q_i during the continuous subgraph matching. Intuitively, our goal is to design vertex embeddings $o'(\cdot)$ that minimizes the cost model $Cost_{CSM}$ (given in Eq. (11)).

Note that, $\Phi(\cdot)$ in Eq. (11) is a monotonically increasing function. Therefore, if we can minimize the term $\frac{\mu_{o'(v_i)[j]} - o'(q_i)[j]}{\sigma^2_{o'(v_i)[j]}}$ in Eq. (11), then low query cost $Cost_{CSM}$ can be achieved. In other words, guided by our proposed cost model (to minimize the query cost $Cost_{CSM}$), **our target is to design/select a "good" distribution of vertex dominance embeddings $o'(v_i)$ with:**

(1) low mean $\mu_{o'(v_i)[j]}$, and;

(2) high variance $\sigma^2_{o'(v_i)[j]}$.

In this paper, unlike standard vertex dominance embeddings generated by *Uniform* random function $f_1(\cdot)$ (as discussed in Section 4.2), we choose to use a (seeded) *Zipf* random function to produce SPUR vectors x_i in vertex embeddings $o'(v_i)$ (or in turn generate SPAN vectors $y_i = \sum x_j$). The $Zipf$ distribution exactly follows our target of finding a random variable with low mean and high variance, which can achieve low query cost, as guided by our proposed cost model $Cost_{CSM}$ (given in Eq. (8)). Please refer to experimental comparison of $Zipf$ with $Uniform$ in Figure 9(c) of Section 8. We would like to leave the interesting topic of studying other low-mean/high-variance distributions as our future work.

Discussions on the Seeded Zipf Generator: We consider two distributions, $Uniform$ and $Zipf$, each of which is divided to b buckets with the same area. This way, we create a 1-to-1 mapping between buckets in a Uniform distribution and that in a $Zipf$ distribution. Given a seeded pseudo-random number, r, from the Uniform distribution, we can first find the bucket this Uniform random number falls into, obtain its corresponding bucket in the $Zipf$ distribution, and compute its proportional location r' in the $Zipf$ bucket. As a result, r' is the random number that follows the $Zipf$ distribution.

Integration of the Cost-Model-Based Vertex Embeddings into our Dynamic Subgraph Matching Framework: In light of our cost model above, we design a novel cost-model-based vertex dominance embedding, denoted as $o'_C(v_i)$, for each vertex v_i, that is:

$$o'_C(v_i) = \alpha(x'_i || y'_i) + \beta z_i, \quad (12)$$

Table 2: Statistics of real-world graph data sets.

Data Sets	$\|V(G)\|$	$\|E(G)\|$	$\|\sum\|$	$avg\_deg(G)$
Yeast (ye)	3,112	12,519	71	8.0
HPRD (hp)	9,460	34,998	307	7.4
DBLP (db)	317,080	1,049,866	15	6.6
Youtube (yt)	1,134,890	2,987,624	25	5.3
US Patents (up)	3,774,768	16,518,947	20	8.8

Table 3: Parameter settings.

Parameters	Values
the dimension, d, of the SPUR/SPAN vector	1 **2**, 3, 4
the ratio, β/α	10, 100, **1,000**, 10,000, 100,000
the number, m, of degree groups	1, 2, **3**, 4, 5
the number, K, of cell intervals in each dimension of Syn_j	1, 2, **5**, 8, 10
the number, $\|\Sigma\|$, of distinct labels	5, 10, **15**, 20, 25
the average degree, $avg\_deg(q)$, of the query graph q	2, **3**, 4
the size, $\|V(q)\|$, of the query graph q	5, 6, **8**, 10, 12
the average degree, $avg\_deg(G_D)$, of dynamic graph G_D	3, 4, **5**, 6, 7
the size, $\|V(G_D)\|$, of the dynamic data graph G_D	10K, 30K, **50K**, 80K, 100K, 500K, 1M

where $x'_i = f_Z(l(v_i))$ is the newly designed SPUR vector generated by a seeded $Zipf$ random generator $f_Z(\cdot)$, and $y'_i = \sum_{\forall v_j \in \mathcal{N}_{v_i}} x'_j$.

Note that, since the SPAN vector y'_i in Eq. (12) is given by $\sum_{\forall v_j \in \mathcal{N}_{v_i}} x'_j$, its distribution still follows the property of low mean and high variance to achieve low query cost.

8 Experimental Evaluation

8.1 Experimental Settings

To evaluate the performance of our DIVINE approach, we conduct experiments on a Ubuntu server equipped with an Intel Core i9-12900K CPU and 128GB memory. Our source code in C++ and real/synthetic graph data sets are available at URL: *https://anonymous.4open.science/r/DSM-B102/*.

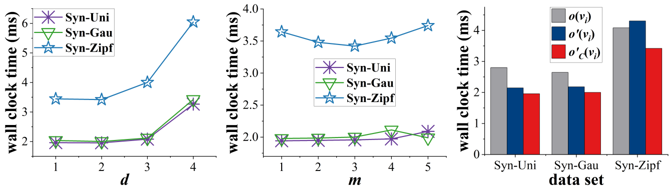
Real/Synthetic Graph Data Sets: We evaluate our DIVINE approach over both real and synthetic graphs.

Real-world graphs. We test five real-world graph data used by previous works [8, 9, 26, 27, 30, 39, 41, 51, 53, 58, 60, 73], which can be classified into three categories: i) biology networks (Yeast and HPRD); ii) bibliographical/social networks (DBLP and Youtube), and; iii) citation networks (US Patents). Statistics of these real graphs are summarized in Table 2.

Synthetic graphs. We generate synthetic graphs via NetworkX [25], and produce small-world graphs following the Newman-Watts-Strogatz model [68]. Parameter settings of synthetic graphs are depicted in Table 3. For each vertex v_i, we generate its label $l(v_i)$ by randomly picking up an integer within $[1, |\sum|]$, following the $Uniform$, $Gaussian$, or $Zipf$ distribution. Accordingly, we obtain 3 types of graphs, denoted as *Syn-Uni*, *Syn-Gau*, and *Syn-Zipf*, resp.

The insertion (or deletion) rate is defined as the ratio of the number of edge insertions (or deletions) to the total number of edges in the raw graph data. Following the literature [14, 40, 43, 59], we set the insertion rate to 10% by default (i.e., initial graph G_0 contains 90% edges, and the remaining 10% edges arrive as the insertion stream). Similarly, for the case of edge deletions, we set the default deletion rate to 10% (i.e., initial graph G_0 contains all edges, and deletion operations of 10% edges arrive in the stream). Below, we will report the CSM performance with edge insertion only, unless specified otherwise (see the edge-deletion case in Figure 13).

Query Graphs: Similar to previous works [3, 8, 9, 26, 27, 39, 51, 59, 60], for each graph G_D, we randomly extract/sample 100 connected subgraphs as query graphs, where parameters of query graphs q (e.g., $|V(q)|$ and $avg\_deg(q)$) are depicted in Table 3.



(a) DIVINE vs. d (b) DIVINE vs. m (c) DIVINE vs. embeddings

Figure 9: The DIVINE efficiency w.r.t parameters d, m, and embedding design.

Baseline Methods: We compare our DIVINE approach (using the cost-model-based vertex dominance embeddings $o'_C(v_i)$, as discussed in Section 7) with five representative baseline methods of continuous subgraph matching as follows: Graphflow (GF) [37], SJ-Tree (SJ) [14], TurboFlux (TF) [40], SymBi (Sym) [43], and IEDyn (IED) [33, 34]. We use the code of baseline methods from [59], which is implemented in C++ for a fair comparison.

Evaluation Metrics: In our experiments, we report the efficiency of our DIVINE approach and baseline methods, in terms of the *wall clock time*, which is the time cost of processing the CSM query over the entire graph update sequence (i.e., continuous subgraph matching answer set monitoring). In particular, this time cost includes the time of filtering, refinement, embedding updates, and synopsis updates for each CSM query. We also evaluate the *pruning power* of our *embedding dominance pruning* and *MBR range pruning strategies* (as mentioned in Section 6.1), which is defined as the percentage of vertices that can be ruled out by our pruning methods. For all the experiments, we take an average of each metric over 100 runs (w.r.t. 100 query graphs, resp.).

Table 3 depicts parameter settings in our experiments, where default parameter values are in bold. For each set of experiments, we vary the value of one parameter while setting other parameters to their default values.

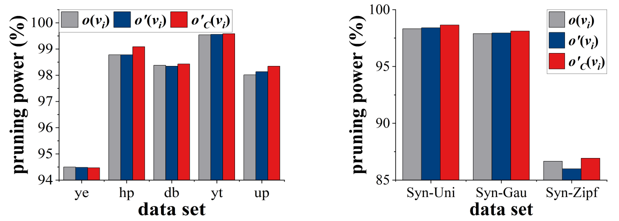
8.2 Parameter Tuning

We first tune parameters for our DIVINE approach in synthetic graph data sets.

The DIVINE Efficiency w.r.t. SPUR/SPAN Vector Dimension d: Figure 9(a) illustrates the DIVINE performance by varying the dimension, d, of the cost-model-based SPUR/SPAN vector (in $o'_C(v_i)$ of Eq. (12)) from 1 to 4, where other parameters are set to their default values. With higher embedding dimension d, the pruning power of our proposed pruning strategies in higher dimensional space increases. However, the access of synopses with larger d may also incur higher costs due to the "dimensionality curse" [7]. Thus, in this figure, for larger d, the wall clock time of DIVINE first decreases and then increases over all synthetic graphs. Nonetheless, the time cost remains low (i.e., 1.96 ~ 6.05 ms) for different d values.

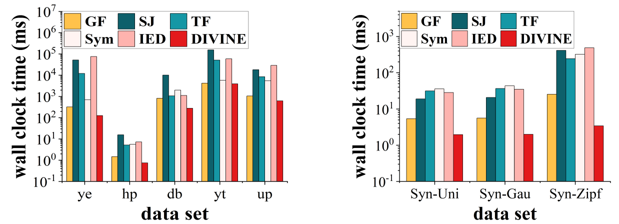
The DIVINE Efficiency w.r.t. # of Degree Groups, m: Figure 9(b) reports the performance of our DIVINE approach, by varying the number, m, of degree groups from 1 to 5, where other parameters are set by default. In this figure, the time costs over *Syn-Uni* and *Syn-Zipf* first decrease and then increase when m increases, and there are some fluctuations for *Syn-Gau* (e.g., for m = 4 or 5). For all m values, the time cost remains low (i.e., 1.94 ~ 3.74 ms).

The DIVINE Efficiency w.r.t. Vertex Dominance Embedding Design Strategies: Figure 9(c) tests the DIVINE performance with different designs of vertex dominance embeddings, $o(v_i)$ (in Eq. (3)),



(a) real-world graphs (b) synthetic graphs

Figure 10: The DIVINE pruning power w.r.t different design strategies of vertex dominance embeddings.



(a) real-world graphs (b) synthetic graphs

Figure 11: The DIVINE efficiency on real/synthetic graphs, compared with baseline methods.

$o'(v_i)$ (in Eq. (4)), and $o'_C(v_i)$ (in Eq. (12)), where default values are used for parameters. In the figure, we can see that, the optimized vertex embeddings $o'(v_i)$ (via the base vector z_i) incur smaller time cost than $o(v_i)$ in all cases, whereas the cost-model-based vertex embeddings $o'_C(v_i)$ consistently achieve the lowest time. For different vertex embeddings, the query cost remains low (i.e., 1.96~4.3ms).

Please refer to the tuning of other parameters (e.g., β/α and K) in Appendix D. In subsequent experiments, we will set parameters $d = 2$, $\beta/\alpha = 1,000$, $m = 3$, and $K = 5$, and use the best cost-model-based vertex embeddings $o'_C(v_i)$ (given by Eq. (12)).

8.3 The DIVINE Effectiveness Evaluation

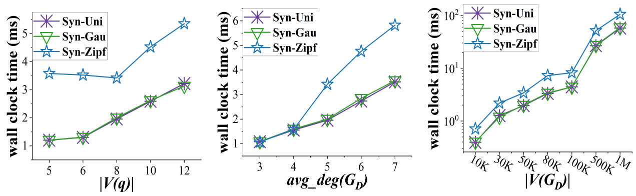
In this subsection, we report the pruning power of our proposed pruning strategies (as discussed in Section 6.1) for continuous subgraph matching over real/synthetic graphs.

The DIVINE Pruning Power on Real/Synthetic Graphs: Figure 10 shows the pruning power of our DIVINE approach over real/synthetic graphs with different embedding designs $o(v_i)$, $o'(v_i)$, and $o'_C(v_i)$, where all parameters are set by default. In figures, we can see that for all real/synthetic graphs, the pruning power of our proposed vertex embedding designs can reach as high as 94.47% ~ 99.58% for real-world graphs and 85.99% ~ 98.65% for synthetic graphs, which confirms the effectiveness of our embedding-based pruning strategies. Our cost-model-based vertex embedding strategy $o'_C(v_i)$ always achieves the highest pruning power (i.e., 86.93% ~ 99.58%).

8.4 The DIVINE Efficiency Evaluation

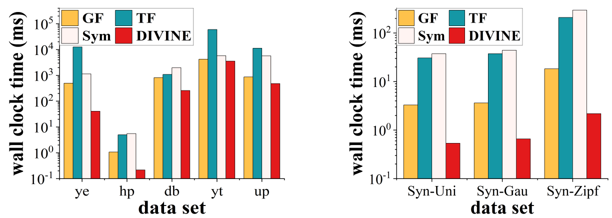
In this subsection, we report the efficiency of our DIVINE approaches over real/synthetic graphs.

The DIVINE Efficiency on Real/Synthetic Dynamic Graphs: Figure 11 compares the efficiency of our DIVINE approach with that of 5 state-of-the-art baseline methods over both real and synthetic graphs, where all parameters are set to default values. In the figure, we can see that our DIVINE approach outperforms baseline methods mostly by 1-2 orders of magnitude. Overall, for all graphs (even for



(a) DIVINE vs. $|V(q)|$ (b) DIVINE vs. $avg\_deg(G_D)$ (c) DIVINE vs. $|V(G_D)|$

Figure 12: The DIVINE efficiency w.r.t. parameters $|V(q)|$, $avg\_deg(G_D)$, and $|V(G_D)|$.



(a) real-world graphs (b) synthetic graphs

Figure 13: The DSM efficiency on edge-deletion-only real/synthetic graphs, compared with baseline methods.

up with 3.77M vertices and 1.65M edge insertions), the time cost of our DIVINE approach remains low (i.e., <3.91 *sec*).

To evaluate the robustness of our DIVINE approaches, in the sequel, we vary the parameter values on synthetic graphs (e.g., $|V(q)|$, $avg\_deg(G_D)$, and $|V(G_D)|$). To better illustrate the trends of curves, we omit the results of the baselines below. Moreover, we also evaluate the DIVINE efficiency in the edge-deletion case.

The DIVINE Efficiency w.r.t. Query Graph Size $|V(q)|$: Figure 12(a) illustrates our DIVINE performance, by varying the query graph size, $|V(q)|$, from 5 to 12, where default values are used for other parameters. When the number, $|V(q)|$, of vertices in query graph q increases, fewer candidate subgraphs are expected to match with larger query graph q. On the other hand, larger query graph size $|V(q)|$ will cause higher query costs in finding candidates for more query vertices, through synopsis traversal and refinement. Therefore, the query time is influenced by these two factors. Nevertheless, the time cost remains low for different query graph sizes (i.e., <5.36ms).

The DIVINE Efficiency w.r.t. Avg. Degree, $avg\_deg(G_D)$, of Dynamic Graph G_D: Figure 12(b) presents our DIVINE performance with different average degrees, avg_deg (G_D), of dynamic graph G_D, where $avg\_deg(G_D) = 3 \sim 7$, and default values are used for other parameters. Intuitively, higher degree $avg\_deg(G_D)$ in data graph G_D incurs lower pruning power and more candidate vertices. Thus, when $avg\_deg(G_D)$ becomes higher, the wall clock time also increases, especially for *Syn-Zipf* (due to its skewed vertex label distribution). Nevertheless, the DIVINE query time remains small for different $avg\_deg(G_D)$ values (i.e., 1.08 ∼ 5.82 *ms*).

The DIVINE Scalability Test w.r.t. Dynamic Graph Size $|V(G_D)|$: Figure 12(c) tests the scalability of our DIVINE approach with different dynamic graph sizes, $|V(G_D)|$, from 10K to 1M, where default values are assigned to other parameters. A larger dynamic graph incurs more matching candidate vertices (and, in turn, candidate subgraphs). From this figure, the time cost of DIVINE increases linearly with the increase of graph size $|V(G_D)|$, nonetheless, remains low (i.e., 0.4 ∼ 104.62 *ms*), which confirms the scalability of our DIVINE approach for large graph sizes.

The DIVINE Efficiency on Edge-Deletion-Only Dynamic Graphs: Since SJ and IED baselines do not support the edge deletion [59], Figure 13 only compares the efficiency of our DIVINE approach with that of GF, TF, and SYM over real/synthetic graphs, where all parameters are set to their default values. From the figure, our DIVINE approach can always outperform the three baseline methods. Overall, for all real/synthetic data sets (even for *up* with 1.65M edge deletions), the wall clock time of our DIVINE approach remains low (i.e., < 3.55 *sec*).

Please refer to more experimental results for other parameters (e.g., different insertion/deletion rates, $|\Sigma|$, and $avg\_deg(q)$) and offline pre-computation cost (e.g., DAS^3 synopsis time/space cost and pre-computation cost with various graph sizes) in Appendix D.

9 Related Work

Continuous Subgraph Matching: Existing exact dynamic continuous matching methods obtain incremental matching results in the following three ways: i) recomputing matching results at each timestamp [20]; ii) direct updating matching results over the data graph [37], and; iii) using auxiliary index over matching results to find updates [14, 33, 34, 40, 43]. In contrast, our work designs a novel and effective *vertex dominance embedding* technique for candidate vertex/subgraph retrieval, which can effectively prune vertex/subgraph candidates and improve query efficiency.

In addition, there are also some studies that consider specific query graph topologies (e.g., paths [47, 57], cycles [49], and cliques [44]) or approximate matching [12, 18, 20, 31, 54]. Different from these works, our paper focuses on exact subgraph matching query over dynamic graphs, given general query graphs (i.e., with arbitrary query graph structures).

Graph Embeddings: Traditional heuristic-based graph embedding methods [23, 48, 52, 62, 63] are designed for static graphs and generate graph or node embeddings for fixed graph structures, which cannot be directly used for a dynamic graph task. With the prosperity of *graph neural networks* (GNNs) [28, 32, 55, 56, 64–66], some previous works [4, 17, 42, 71] proposed to use GNNs to generate graph embeddings for the graph matching. However, these works either cannot guarantee the accuracy of tasks over unseen test graph data [4, 42] (due to the limitation of neural networks), or cannot efficiently and incrementally maintain embeddings in dynamic graphs with continuous updates [17, 71]. In contrast, our vertex dominance embedding technique does not use learning-based graph embedding, which can guarantee exact subgraph matching over dynamic graphs without false dismissals and enable incremental embedding maintenance over dynamic graphs upon updates.

10 Conclusion

In this paper, we propose a novel *DynamIc Vertex DomINance Embedding* (DIVINE)-based framework for efficiently processing Continuous Subgraph Matching (CSM) queries over a large-scale dynamic graph. We also provide an effective degree grouping technique and pruning strategies to facilitate our efficient algorithms of retrieving/maintaining CSM subgraph answers. Most importantly, we devise an effective cost model for guiding the design of dynamic vertex dominance embeddings and further enhance the pruning power of our CSM query processing. Extensive experiments on real/synthetic dynamic graphs demonstrate the good performance of our DIVINE approach.

References

[1] Ahmed Al-Baghdadi, Gokarna Sharma, and Xiang Lian. 2020. Efficient processing of group planning queries over spatial-social networks. *IEEE Transactions on Knowledge and Data Engineering* 34, 5 (2020), 2135–2147.

[2] Réka Albert and Albert-László Barabási. 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74, 1 (2002), 47.

[3] Blair Archibald, Fraser Dunlop, Ruth Hoffmann, Ciaran McCreesh, Patrick Prosser, and James Trimble. 2019. Sequential and parallel solution-biased search for subgraph algorithms. In *Proceedings of the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR)*. 20–38.

[4] Yunsheng Bai, Hao Ding, Song Bian, Ting Chen, Yizhou Sun, and Wei Wang. 2019. Simgnn: A neural network approach to fast graph similarity computation. In *Proceedings of the International Conference on Web Search and Data Mining (WSDM)*. 384–392.

[5] Albert-László Barabási and Réka Albert. 1999. Emergence of scaling in random networks. *Science* 286, 5439 (1999), 509–512.

[6] Albert-László Barabási and Eric Bonabeau. 2003. Scale-free networks. *Scientific American* 288, 5 (2003), 60–69.

[7] Stefan Berchtold, Daniel A. Keim, and Hans-Peter Kriegel. 1996. The X-tree: An Index Structure for High-Dimensional Data. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 28–39.

[8] Bibek Bhattarai, Hang Liu, and H Howie Huang. 2019. Ceci: Compact embedding cluster index for scalable subgraph matching. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1447–1462.

[9] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient subgraph matching by postponing cartesian products. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1199–1214.

[10] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinformatics* 14, 7 (2013), 1–13.

[11] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 421–430.

[12] Lei Chen and Changliang Wang. 2010. Continuous subgraph pattern search over certain and uncertain graph streams. *IEEE Transactions on Knowledge and Data Engineering* 22, 8 (2010), 1093–1109.

[13] Zaiben Chen, Heng Tao Shen, Xiaofang Zhou, and Jeffrey Xu Yu. 2009. Monitoring path nearest neighbor in road networks. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 591–602.

[14] Sutanay Choudhury, Lawrence Holder, George Chin, Khushbu Agarwal, and John Feo. 2015. A selectivity based approach to continuous pattern detection in streaming graphs. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 157–168.

[15] Luigi P Cordella, Pasquale Foggia, Carlo Sansone, and Mario Vento. 2004. A (sub) graph isomorphism algorithm for matching large graphs. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 26, 10 (2004), 1367–1372.

[16] Sergei N Dorogovtsev and José FF Mendes. 2003. *Evolution of networks: From biological nets to the Internet and WWW*. Oxford University Press.

[17] Chi Thang Duong, Trung Dung Hoang, Hongzhi Yin, Matthias Weidlich, Quoc Viet Hung Nguyen, and Karl Aberer. 2021. Efficient streaming subgraph isomorphism with graph neural networks. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, Vol. 14. 730–742.

[18] Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. 2010. Graph pattern matching: From intractable to polynomial time. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, Vol. 3. 264–275.

[19] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Expfinder: Finding experts by graph pattern matching. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 1316–1319.

[20] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Incremental graph pattern matching. *ACM Transactions on Database Systems* 38, 3 (2013), 1–47.

[21] Michael R. Garey and David S. Johnson. 1983. Computers and intractability: A guide to the theory of NP-completeness. *The Journal of Symbolic Logic* 48, 2 (1983), 498–500.

[22] Martin Grohe and Pascal Schweitzer. 2020. The graph isomorphism problem. *Commun. ACM* 63, 11 (2020), 128–134.

[23] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 855–864.

[24] Ralf Hartmut Güting. 1994. An introduction to spatial database systems. *The International Journal on Very Large Data Bases* 3 (1994), 357–399.

[25] Aric Hagberg and Drew Conway. 2020. Networkx: Network analysis with python. *URL: https://networkx. github. io* (2020).

[26] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1429–1446.

[27] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: towards ultrafast and robust subgraph isomorphism search in large graph databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 337–348.

[28] Yu Hao, Xin Cao, Yufan Sheng, Yixiang Fang, and Wei Wang. 2021. Ks-gnn: Keywords search over incomplete graphs via graphs neural network. In *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*. 1700–1712.

[29] Oktie Hassanzadeh, Anastasios Kementsietsidis, and Yannis Velegrakis. 2012. Data management issues on the semantic web. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 1204–1206.

[30] Huahai He and Ambuj K Singh. 2008. Graphs-at-a-time: query language and access methods for graph databases. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 405–418.

[31] Monika Rauch Henzinger, Thomas A Henzinger, and Peter W Kopke. 1995. Computing simulations on finite and infinite graphs. In *Proceedings of the IEEE Annual Foundations of Computer Science (AFCS)*. 453–462.

[32] Chenji Huang, Yixiang Fang, Xuemin Lin, Xin Cao, and Wenjie Zhang. 2022. Able: Meta-path prediction in heterogeneous information networks. *ACM Transactions on Knowledge Discovery from Data* 16, 4 (2022), 1–21.

[33] Muhammad Idris, Martín Ugarte, and Stijn Vansummeren. 2017. The dynamic Yannakakis algorithm: Compact and efficient query processing under updates. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1259–1274.

[34] Muhammad Idris, Martín Ugarte, Stijn Vansummeren, Hannes Voigt, and Wolfgang Lehner. 2020. General dynamic Yannakakis: conjunctive queries with theta joins under updates. *The International Journal on Very Large Data Bases* 29, 2 (2020), 619–653.

[35] Alpár Jüttner and Péter Madarasi. 2018. VF2++—An improved subgraph isomorphism algorithm. *Discrete Applied Mathematics* 242 (2018), 69–81.

[36] Xuan Kan, Zimu Li, Hejie Cui, Yue Yu, Ran Xu, Shaojun Yu, Zilong Zhang, Ying Guo, and Carl Yang. 2023. R-Mixup: Riemannian Mixup for Biological Networks. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 1073–1085.

[37] Chathura Kankanamge, Siddhartha Sahu, Amine Mhedbhi, Jeremy Chen, and Semih Salihoglu. 2017. Graphflow: An active graph database. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1695–1698.

[38] Guy Karlebach and Ron Shamir. 2008. Modelling and analysis of gene regulatory networks. *Nature Reviews Molecular Cell Biology* 9, 10 (2008), 770–780.

[39] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. 2017. Subgraph querying with parallel use of query rewritings and alternative algorithms. In *Proceedings of the International Conference on Extending Database Technology (EDBT)*. 25–36.

[40] Kyoungmin Kim, In Seo, Wook-Shin Han, Jeong-Hoon Lee, Sungpack Hong, Hassan Chafi, Hyungyu Shin, and Geonhwa Jeong. 2018. Turboflux: A fast continuous subgraph matching system for streaming graph data. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 411–426.

[41] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An in-depth comparison of subgraph isomorphism algorithms in graph databases. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 133–144.

[42] Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph matching networks for learning the similarity of graph structured objects. In *Proceedings of the International Conference on Machine Learning (ICML)*. 3835–3845.

[43] Seunghwan Min, Sung Gwan Park, Kunsoo Park, Dora Giammarresi, Giuseppe F Italiano, and Wook-Shin Han. 2021. Symmetric continuous subgraph matching with bidirectional dynamic programming. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 1298–1310.

[44] Jayanta Mondal and Amol Deshpande. 2016. Casqd: continuous detection of activity-based subgraph pattern queries on dynamic graphs. In *Proceedings of the International Conference on Distributed and Event-based Systems (ICDES)*. 226–237.

[45] Mark EJ Newman. 2005. Power laws, Pareto distributions and Zipf's law. *Contemporary Physics* 46, 5 (2005), 323–351.

[46] Abdelghny Orogat and Ahmed El-Roby. 2022. SmartBench: demonstrating automatic generation of comprehensive benchmarks for question answering over knowledge graphs. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 3662–3665.

[47] You Peng, Ying Zhang, Xuemin Lin, Wenjie Zhang, and Jingren Zhou. 2019. Hop-constrained s-t Simple Path Enumeration: Towards Bridging Theory and Practice. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, Vol. 13. 463–476.

[48] Bryan Perozzi, Rami Al-Rfou, and Steven Skiena. 2014. Deepwalk: Online learning of social representations. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 701–710.

[49] Xiafei Qiu, Wubin Cen, Zhengping Qian, You Peng, Ying Zhang, Xuemin Lin, and Jingren Zhou. 2018. Real-time constrained cycle detection in large dynamic graphs. *Proceedings of the International Conference on Very Large Data Bases (PVLDB)* 11, 12 (2018), 1876–1888.

[50] Niranjan Rai and Xiang Lian. 2023. Top-k Community Similarity Search Over Large-Scale Road Networks. *IEEE Transactions on Knowledge and Data Engineering* 35, 10 (2023), 10710–10721.

[51] Xuguang Ren and Junhu Wang. 2015. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 617–628.

[52] Leonardo FR Ribeiro, Pedro HP Saverese, and Daniel R Figueiredo. 2017. struc2vec: Learning node representations from structural identity. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 385–394.

[53] Haichuan Shang, Ying Zhang, Xuemin Lin, and Jeffrey Xu Yu. 2008. Taming verification hardness: an efficient algorithm for testing subgraph isomorphism. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 364–375.

[54] Chunyao Song, Tingjian Ge, Cindy Chen, and Jie Wang. 2014. Event pattern matching over graph streams. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, Vol. 8. 413–424.

[55] Li Sun, Zhenhao Huang, Zixi Wang, Feiyang Wang, Hao Peng, and S Yu Philip. 2024. Motif-aware Riemannian Graph Neural Network with Generative-Contrastive Learning. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 38. 9044–9052.

[56] Li Sun, Zhongbao Zhang, Junda Ye, Hao Peng, Jiawei Zhang, Sen Su, and S Yu Philip. 2022. A self-supervised mixed-curvature graph neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, Vol. 36. 4146–4155.

[57] Shixuan Sun, Yuhang Chen, Bingsheng He, and Bryan Hooi. 2021. Pathenum: Towards real-time hop-constrained st path enumeration. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1758–1770.

[58] Shixuan Sun and Qiong Luo. 2020. In-memory subgraph matching: An in-depth study. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1083–1098.

[59] Xibo Sun, Shixuan Sun, Qiong Luo, and Bingsheng He. 2022. An in-depth study of continuous subgraph matching. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*, Vol. 15. 1403–1416.

[60] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient Subgraph Matching on Billion Node Graphs. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 788–799.

[61] Damian Szklarczyk et al. 2015. STRING v10: protein–protein interaction networks, integrated over the tree of life. *Nucleic Acids Research* 43, D1 (2015), D447–D452.

[62] Jian Tang, Meng Qu, Mingzhe Wang, Ming Zhang, Jun Yan, and Qiaozhu Mei. 2015. Line: Large-scale information network embedding. In *Proceedings of the Web Conference (WWW)*. 1067–1077.

[63] Daixin Wang, Peng Cui, and Wenwu Zhu. 2016. Structural deep network embedding. In *Proceedings of the International Conference on Knowledge Discovery and Data Mining (SIGKDD)*. 1225–1234.

[64] Hanchen Wang, Defu Lian, Wanqi Liu, Dong Wen, Chen Chen, and Xiaoyang Wang. 2022. Powerful graph of graphs neural network for structured entity analysis. In *Proceedings of the Web Conference (WWW)*. 609–629.

[65] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, Xiangjian He, Yiguang Lin, and Xuemin Lin. 2021. Binarized graph neural network. In *Proceedings of the Web Conference (WWW)*. 825–848.

[66] Hanchen Wang, Defu Lian, Ying Zhang, Lu Qin, and Xuemin Lin. 2020. Gognn: Graph of graphs neural network for predicting structured entity interactions. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*. 1317–1323.

[67] Stanley Wasserman and Katherine Faust. 1994. Social network analysis: Methods and applications. (1994).

[68] Duncan J Watts and Steven H Strogatz. 1998. Collective dynamics of 'small-world'networks. *Nature* 393, 6684 (1998), 440–442.

[69] Wikipedia. 2023. Central Limit Theorem. *https://en.wikipedia.org/wiki/Central_limit_theorem* (2023).

[70] Carl Yang and Jiawei Han. 2023. Revisiting citation prediction with cluster-aware text-enhanced heterogeneous graph neural networks. In *Proceedings of the International Conference on Data Engineering (ICDE)*. 682–695.

[71] Yutong Ye, Xiang Lian, and Mingsong Chen. 2024. Efficient Exact Subgraph Matching via GNN-based Path Dominance Embedding. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 1628–1641.

[72] Yikai Zhang and Jeffrey Xu Yu. 2022. Relative Subboundedness of Contraction Hierarchy and Hierarchical 2-Hop Index in Dynamic Road Networks. In *Proceedings of the International Conference on Management of Data (SIGMOD)*. 1992–2005.

[73] Peixiang Zhao and Jiawei Han. 2010. On graph query optimization in large networks. In *Proceedings of the International Conference on Very Large Data Bases (PVLDB)*. 340–351.