Appendix

A Lemma Proofs

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. For a given unit star subgraph g_{v_i} centered at vertex v_i and any of its star substructures s_{v_i} (i.e., $s_{v_i} \subseteq g_{v_i}$), since they share the same center vertex v_i (i.e., with the same label $l(v_i)$), they must have the same SPUR vector x_i.

Moreover, since $s_{v_i} \subseteq g_{v_i}$ holds, we also have $\mathcal{N}_{v_i}(s_{v_i}) \subseteq \mathcal{N}_{v_i}(g_{v_i})$, where $\mathcal{N}_{v_i}(s_{v_i})$ (or $\mathcal{N}_{v_i}(g_{v_i})$) is a set of v_i's 1-hop neighbors in subgraph s_{v_i} (or g_{v_i}). Thus, for SPAN vectors $y_i(s_{v_i})$ and $y_i(g_{v_i})$ of s_{v_i} and g_{v_i}, respectively, it must hold that: $y_i(s_{v_i})[k] \leq y_i(g_{v_i})[k]$ for all dimensions k (since we have $y_i(s_{v_i})[k] = \sum_{\forall v_j \in \mathcal{N}_{v_i}(s_{v_i})} x_j[k] \leq \sum_{\forall v_j \in \mathcal{N}_{v_i}(g_{v_i})} x_j[k] = y_i(g_{v_i})[k]$). In other words, their SPAN vectors satisfy the condition that $y_i(s_{v_i}) \preceq y_i(g_{v_i})$. Therefore, for $s_{v_i} \subseteq g_{v_i}$, their vertex dominance embeddings satisfy the condition that: $o(s_{v_i}) = (x_i || y_i(s_{v_i})) \preceq (x_i || y_i(g_{v_i})) = o(g_{v_i})$ (including $o(s_{v_i}) = o(g_{v_i})$) in the embedding space, which completes the proof. □

Lemma 4.2. (The Dominance Property of the Optimized 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 optimized vertex 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. Given a unit star subgraph g_{v_i} centered at vertex v_i and any of its star substructures s_{v_i} (i.e., $s_{v_i} \subseteq g_{v_i}$), since they have the same center vertex v_i (with the same vertex label), their base vectors have the same values, i.e., $z_i(s_{v_i}) = z_i(g_{v_i})$, or equivalently $\beta z_i(s_{v_i}) = \beta z_i(g_{v_i})$ (for $\beta > 0$).

Due to the property of vertex dominance embeddings (as given in Lemma 4.1), we have $o(s_{v_i}) \preceq o(g_{v_i})$, or equivalently $\alpha o(s_{v_i}) \preceq \alpha o(g_{v_i})$ (for $\alpha > 0$).

Therefore, we can derive that $\alpha o(s_{v_i}) + \beta z_i(s_{v_i}) \preceq \alpha o(g_{v_i}) + \beta z_i(g_{v_i})$, or equivalently $o'(s_{v_i}) \preceq o'(g_{v_i})$ (including $o'(s_{v_i}) = o'(g_{v_i})$), which completes the proof. □

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. If a query vertex q_i in query graph q matches with a data vertex v_i in a subgraph of the data graph, then their vertex dominance embeddings must hold that $o'(q_i) \preceq o'(v_i)$. Therefore, if $o'(q_i)$ does not dominate the cell C, then $o'(q_i)$ cannot dominate any vertex $o'(v_i)$ inside cell C, and all vertices in cell C (or cell C) can be safely pruned.

Moreover, if $o'(q_i)$ does not dominate embedding upper bound vector $v_i.UB_\delta$, i.e., $v_i.UB_\delta \notin DR(o'(q_i))$, then $o'(q_i)$ does not dominate any star substructure s_{v_i} with center vertex v_i and the corresponding degree group. In other words, q_i does not match with v_i. Thus, vertex v_i can be safely pruned. □

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. The MBR $v_i.MBR_{deg(q_i)}$ minimally bounds vertex embeddings for all possible star substructures s_{v_i} with center vertex v_i and degree $deg(q_i)$. Thus, if query vertex $q_i \in V(q)$ and its 1-hop neighbors match with some star substructures with the same degree $deg(q_i)$, then its query embedding vector $o'(q_i)$ must fall into this MBR $v_i.MBR_{deg(q_i)}$. Therefore, if this condition does not hold, i.e., $o'(q_i) \notin v_i.MBR_{deg(q_i)}$, then q_i does not match with v_i, and v_i can be safely pruned, which completes the proof. □

B Details of Algorithm 3

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).

Specifically, we maintain a vertex vector M that will store vertices of the subgraph g matching with ordered query vertices in Q. To enumerate all matching subgraphs, each time we recursively expand partial matching results by including a new candidate vertex u as $M[n]$ that maps with the n-th query vertex $Q[n]$. If we find all vertices in M mapping with Q (i.e., recursion depth $n = |Q|$), then we can add M to the answer set A (lines 1-3). Otherwise, we will find a vertex candidate set S_{cand} from $Q[n].cand\_set$, such that each vertex u in S_{cand} has the same edge connections to that in M as that in Q (lines 5-11). Next, for each vertex $u \in S_{cand}$, we treat it as vertex $M[n]$ matching with $Q[n]$, and recursively call function Refinement(·) with more depth $n + 1$ (lines 13-17).

C Complexity Analysis of Algorithm 2

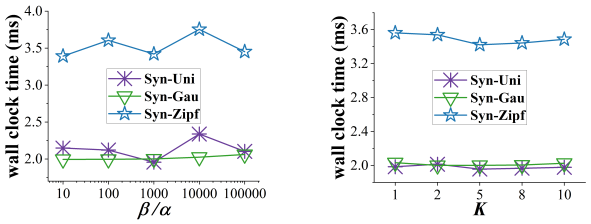
In Algorithm 2, since we need to access each query vertex q_i and its 1-hop neighbors $\mathcal{N}_{v_i}$, the time complexity of computing vertex dominance embeddings $o'(q_i)$ (line 1) is given by $O(|V(q)|+|E(q)|)$.

For the synopsis traversal (lines 2-8), assume that PP_C is the pruning power of the cells' key value and PP_v is the pruning power in each cell's point list. Thus, the synopsis traversal cost is $O(|V(q)| \cdot (1 - PP_C)K^d \cdot (1 - PP_v)|C.list|)$, where K^d is the number of cells and $|C.list|$ is the number of vertices in the cell C.

Next, for the greedy-based query plan generation (lines 9-10), we need to iteratively select a neighbor of vertices in the query plan Q, which requires $O(|V(q)|^2)$ cost.

Finally, we invoke the recursive function Refinement(·) to find actual subgraph matching results (lines 11-13), with the worst-case time complexity $O(\prod_{i=0}^{|V(q)|-1} |q_i.cand\_set|)$.

Therefore, the overall time complexity of Algorithm 2 is given by: $O(|V(q)| + |E(q)| + |V(q)| \cdot (1 - PP_C)K^d \cdot (1 - PP_v)|C.list| + \prod_{i=0}^{|V(q)|-1} |q_i.cand\_set| + |V(q)|^2)$.



(a) DIVINE vs. β/α (b) DIVINE vs. K

Figure 14: The DIVINE efficiency w.r.t β/α and K.

D Experimental Evaluation

Experimental Settings

We compare our DIVINE approaches (using the cost-model-based vertex dominance embeddings $o'_C(v_i)$, as discussed in Section 7) with five representative baseline methods of dynamic subgraph matching as follows:

(1) **Graphflow (GF)** [37] is a direct-incremental algorithm that enumerates updates of matching results without any auxiliary data structure.
(2) **SJ-Tree (SJ)** [14] is an index-based incremental method that evaluates the join query with binary joins using the index.
(3) **TurboFlux (TF)** [40] is an index-based incremental method that stores matches of paths in q without materialization and evaluates the query with the vertex-at-a-time method.
(4) **SymBi (Sym)** [43] is an index-based incremental method that prunes candidate vertex sets using all query edges.
(5) **IEDyn (IED)** [33, 34] is an index-based incremental method that supports acyclic queries and can achieve constant delay enumeration under the setting of graph homomorphism.

We used the code of baseline methods from [59], which is implemented in C++ for a fair comparison.

Parameter Tuning

The DIVINE Efficiency w.r.t. β/α Ratio: Figure 14(a) varies the ratio, β/α, from 10 to 100,000 for the optimized (or cost-model-based) vertex dominance embeddings $o'(v_i)$ (or $o'_C(v_i)$), where other parameters are set by default. In the figure, we can see that our DIVINE approaches are not very sensitive to the ratio β/α. For different β/α ratios, the query cost remains low (i.e., 1.96 ~ 3.75 *ms*).

The DIVINE Efficiency w.r.t. # of Cell Intervals on Each Dimension, K: Figure 14(b) evaluates the effect of the number, K, of cell intervals on each dimension on the DIVINE performance, where K varies from 1 to 10, and other parameters are set by default. When K becomes larger, more vertices in synopsis cells can be pruned, however, more cells need to be accessed. Therefore, for DIVINE, with the increase of K, the time cost first decreases and then increases. Nonetheless, for different K values, the query cost remains low (i.e., 1.96 ~ 3.54 *ms*).

The experimental results on real-world graphs are similar and thus omitted here.

The DIVINE Efficiency Evaluation

The DIVINE Efficiency on Edge-Insertion-Only Real/Synthetic Graphs: Figure 15 compares the efficiency of our DIVINE approach with that of 5 state-of-the-art baseline methods by varying the insertion ratio from 10% to 50% over both real and synthetic graphs, where all other parameters are set to default values. In Figures 15(a) and 15(f) with the default 10% insertion ratio, we can see that our DIVINE approach outperforms baseline methods mostly by 1-2 orders of magnitude. Overall, for all graphs (even for *up* with 3.77M vertices and 1.65M edge insertions), the time cost of our DIVINE approach remains low (i.e., <3.91 *sec*).

For other subfigures with different insertion ratios from 20% to 50%, we can see similar experimental results over both real and synthetic graphs. Although the total query time will increase as the number of inserted edges increases, our DIVINE approach can achieve performance that is up to 1-2 orders of magnitude than baseline methods. In summary, for all insertion ratios and graphs (even for *up* with 3.77M vertices and 8.26M edge insertions), the time cost of our DIVINE approach remains low (i.e., < 4.37 *sec*).

The DIVINE Efficiency on Edge-Deletion-Only Real/Synthetic Graphs: Since SJ and IED baselines do not support the edge deletion [59], Figure 16 only compares the efficiency of our DIVINE approach with that of GF, TF, and SYM over real/synthetic graphs, where the deletion ratio varies from 10% to 50% and all other parameters are set to their default values. From Figures 16(a) and 16(f) with the default 10% deletion ratio, 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*).

From other subfigures with different deletion ratios 20% ~ 50%, we can find similar experimental results over both real and synthetic graphs, where our DIVINE approach can always outperform the three baseline methods. For different edge deletion ratios and graph sizes (even for *up* with 3.77M vertices and 8.26M edge deletions), the time cost of our DIVINE approach remains low (i.e., < 4.62 *sec*).

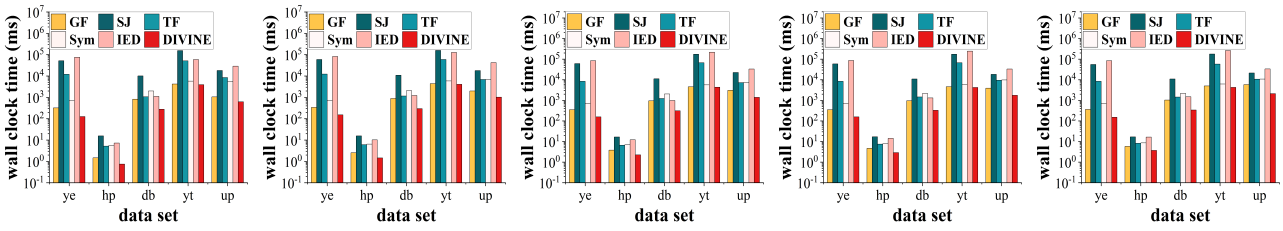
To evaluate the robustness of our DIVINE approaches, in the sequel, we vary the parameter values on synthetic graphs (e.g., $|\Sigma|$ and $avg\_deg(q)$). To better illustrate the trends of curves, we omit the results of the baselines below.

The DIVINE Efficiency w.r.t. # of Distinct Vertex Labels, $|\Sigma|$: Figure 17(a) shows the wall clock time of our DIVINE approach, where $|\Sigma|$ varies from 5 to 25, and other parameters are set to default values. When the number, $|\Sigma|$, of distinct vertex labels increases, the pruning power also increases (i.e., with fewer candidate vertices). Moreover, the query cost is also affected by vertex label distributions. Overall, the DIVINE query cost remains low for different $|\Sigma|$ values (i.e., 1.83 ~ 4.04 *ms*).

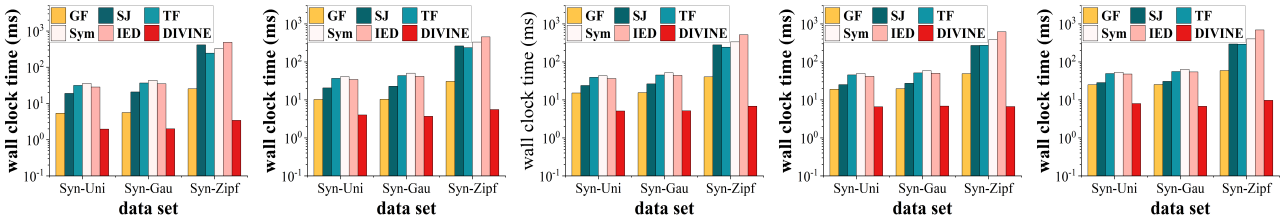
The DIVINE Efficiency w.r.t. Average Degree, $avg\_deg(q)$, of the Query Graph q: Figure 17(b) examines the DIVINE performance by varying the average degree, $avg\_deg(q)$, of the query graph q from 2 to 4, where other parameters are set to default values. Higher degree $avg\_deg(q)$ of q incurs higher pruning power of query vertices. Therefore, when $avg\_deg(q)$ increases, the DIVINE time cost decreases. For different $avg\_deg(q)$ values, the DIVINE query cost remains low (i.e., 1.58 ~ 7.68 *ms*).

DAS^3 Synopsis Initialization Cost

The DAS^3 Synopsis Initialization Cost on Real/Synthetic Graphs. We compare the DAS^3 synopsis initialization cost of our

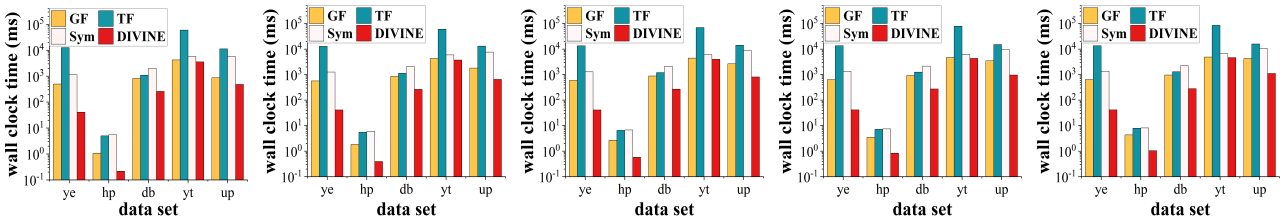


(a) real-world graphs (10%) (b) real-world graphs (20%) (c) real-world graphs (30%) (d) real-world graphs (40%) (e) real-world graphs (50%)

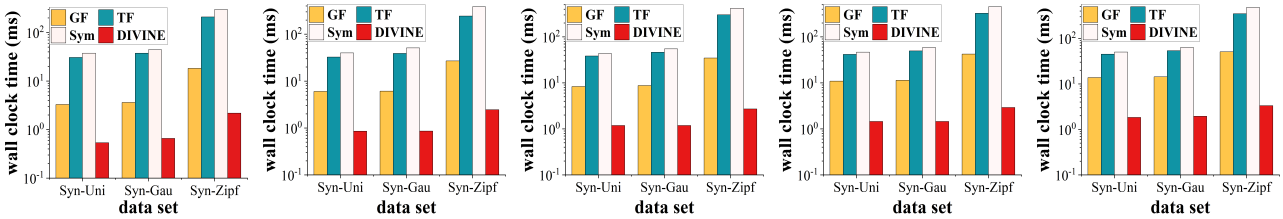


(f) synthetic graphs (10%) (g) synthetic graphs (20%) (h) synthetic graphs (30%) (i) synthetic graphs (40%) (j) synthetic graphs (50%)

Figure 15: The DIVINE efficiency on edge-insertion-only real/synthetic graphs, compared with baseline methods.

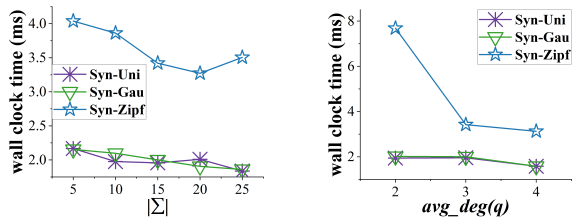


(a) real-world graphs (10%) (b) real-world graphs (20%) (c) real-world graphs (30%) (d) real-world graphs (40%) (e) real-world graphs (50%)



(f) synthetic graphs (10%) (g) synthetic graphs (20%) (h) synthetic graphs (30%) (i) synthetic graphs (40%) (j) synthetic graphs (50%)

Figure 16: The DIVINE efficiency on edge-deletion-only real/synthetic graphs, compared with baseline methods.

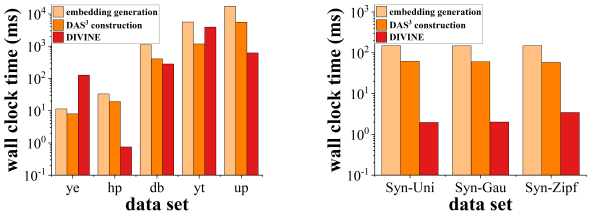


(a) DIVINE vs. $|\Sigma|$ (b) DIVINE vs. $avg\_deg(q)$

Figure 17: The DIVINE efficiency w.r.t. parameters $|\Sigma|$ and $avg\_deg(q)$.

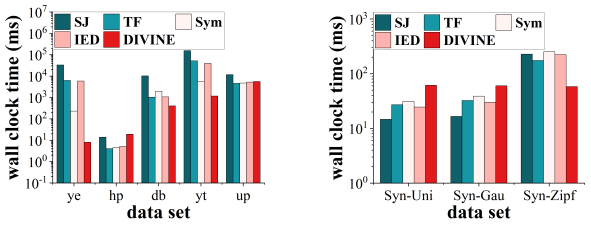
DIVINE approach (including time costs of vertex dominance embedding generation and DAS^3 construction over vertex dominance embeddings) with online query time over real/synthetic graphs, where parameters are set to default values. In Figure 18, for graph size from 3K to 3.77M, the overall offline pre-computation time varies from 19 *ms* ~ 23 *sec*. Specifically, the time costs of embedding generation and DAS^3 construction are 11 *ms* ~ 17 *sec*, 8 *ms* ~ 6 *sec*, respectively. On the other hand, since the online query time includes the time cost of DAS^3 and embedding update, the maintenance cost of our DAS^3 is low.

The DAS^3 Index Time/Space Cost on Real/Synthetic Graphs. Since GF directly enumerates matching answers over the original



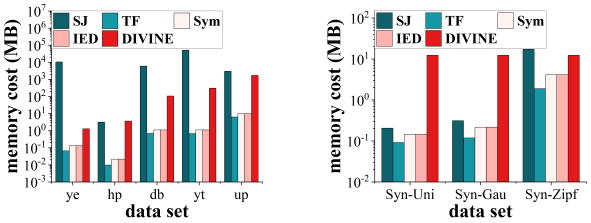
(a) real-world graphs (b) synthetic graphs

Figure 18: The comparison analysis of the DIVINE offline pre-computation and online query costs on real/synthetic graphs.



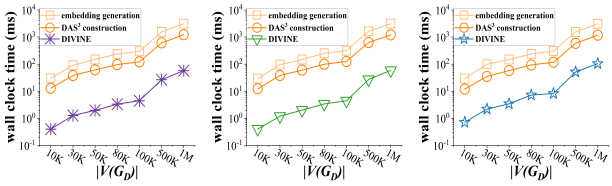
(a) real-world graphs (b) synthetic graphs

Figure 19: DAS^3 construction time on real/synthetic graphs, compared with baseline methods.



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

Figure 20: DAS^3 memory cost on real/synthetic graphs, compared with baseline methods.



(a) *Syn-Uni* (b) *Syn-Gau* (c) *Syn-Zipf*

Figure 21: The DIVINE pre-computation and query costs w.r.t data graph size $|V(G_D)|$.

data graph without any auxiliary data structure [37, 59], Figure 20 compares the storage cost of our DAS^3 with the other 4 baselines, where parameters are set to their default values. From Figure 19, we can see that the one-time construction cost of our DAS^3 synopses is comparable to that of baselines. Moreover, in Figure 20, except for SJ with exponential space cost [14, 59] over most graphs, our DAS^3 synopses need more memory cost for storing/organizing vertex dominance embeddings. However, different from baseline methods that construct an index over more and more incoming (registered) query graphs q (which may not be scalable), our DAS^3 construction is *one-time only* over initial data graph G_0. Thus, our DAS^3 synopses can be used to accelerate numerous online CSM query requests from users simultaneously with high throughput.

The DAS^3 Synopsis Initialization Cost and Online Query Costs w.r.t. Data Graph Size $|V(G_D)|$. Figure 21(a) evaluates the DAS^3 synopsis initialization cost of our DIVINE approach, including time costs of the vertex dominance embedding generation and DAS^3 construction over embeddings, compared with online DIVINE query time, over synthetic graphs *Syn-Uni*, where we vary the graph size $|V(G)|$ from $10K$ to $1M$ and other parameters are set to default values. Specifically, for graph sizes from $10K$ to $1M$, the time costs of the embedding generation and DAS^3 construction are 0.03~3.11 *sec* and 0.01~1.22 *sec*, respectively. The overall offline pre-computation time varies from 0.04 *sec* to 4.33 *sec*, and the dynamic subgraph matching query cost is much smaller (i.e., 0.4~57.49 *ms*).

In Figures 21(b) and 21(c), we can see similar experimental results over *Syn-Gau* and *Syn-Zipf* graphs, respectively, where the subgraph matching query cost is much smaller than the overall offline pre-computation time.