Appendix

COMPLEXITY ANALYSIS

Algorithm	Time complexity	Space complexity
Eq. (4)-Eq. (7)	$O(n^{\lvert V(\tau)\rvert})$	$O(n^{\lvert V_C(\tau)\rvert})$
Disc	$O(n^{\lvert V(\tau)\cup_{\tau'\in anc(\tau)} V_C(\tau')\rvert})$	$O(1)$
Scope	$O(n^{\lvert V(\tau)\cup_{\tau'\in anc(\tau)} V_C^1(\tau')\rvert})$	$O(m)$

Table 5: Comparing the complexity of three algorithms

We summarize the time and space complexity of the three multi-join algorithms discussed in this paper in Table 5. Here, the complexity is for computing a single tree node τ, and the complexity for computing a tree or all trees in an equation is the largest complexity of all tree nodes. $anc(\tau)$ are the ancestors of τ. The complexity of Eq. (4)-Eq. (7) and the space complexity of all algorithms follow the previous discussions in Section 6. In Algorithm 3, we call the algorithm for τ only when the V_C of the ancestors and itself are matched. There are $O(n^{\sum_{\tau'\in anc(\tau)\cup\tau} |V_C(\tau')|})$ matches of the V_C of ancestors and itself. For each match, we need to compute matches for the vertices in $V(\tau) \setminus V_C(\tau)$ whose time complexity is $O(n^{|V(\tau)\setminus V_C(\tau)|})$. Combining the two parts, the time complexity of Disc is $O(n^{|V(\tau)\cup_{\tau'\in anc(\tau)} V_C(\tau')|})$. Similarly, the time complexity of Scope is $O(n^{|V(\tau)\cup_{\tau'\in anc(\tau)} V_C^1(\tau')|})$. Since $\cup_{\tau'\in anc(\tau)} V_C^1(\tau') \subset \cup_{\tau'\in anc(\tau)} V_C(\tau')$, Scope has a smaller time complexity than Disc. Besides, by sharing the V_C^1 of τ and ancestors, we can achieve the optimal time complexity $O(n^{|V(\tau)|})$.

ADDITIONAL EXPERIMENTS

	WS	RM	BY	CA	RG	SG	SY	WW	WH	CC	LJ	SO
EVOKE	0.05	0.07	0.2	0.27	0.59	1.3	4.2	12	18	27	58	-
DISC	61	61	68	62	71	84	-	-	-	-	-	-
SCOPE	0.02	0.03	0.06	0.09	0.21	0.42	1.6	3.9	5.5	9.3	18	40

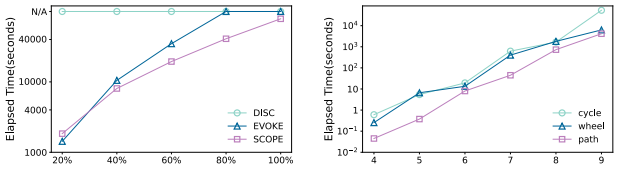
Table 6: The peak memory usage(GB) of 5-node queries

Memory usage. Table 6 shows the peak memory usage of SCOPE, DISC and EVOKE using 5-node queries. DISC has the highest memory usage and it runs out of memory on 6 graphs. EVOKE has a relatively high memory cost and it runs out of memory on the SO graph, although its code is carefully optimized. Our SCOPE has the smallest space cost. For the 407 6-node queries and 50 7-node queries, SCOPE's memory costs are 0.94GB, 1.42GB, respectively, on the SG graph.

		time (seconds)				
k	# queries	TD	equation	symmetry	V_C, attr order	total
5	58	0.045	0.015	0.002	0.067	0.129
6	407	10.449	0.193	0.029	0.728	11.399
7	50	52.258	0.156	0.019	0.425	52.859

Table 7: The preprocessing time of 3 query sets

Preprocessing time. Table 7 shows the preprocessing time of SCOPE. We generate a query plan for each query graph, which consists of an equation and tree decompositions for each pattern in the equation(Algorithm 1), compute *SymR*s, and choose V_C and attribute order for each tree node. It is much shorter than the total time in Fig. 13 and Fig. 14. Note that the plans only depend on the query, so we can also pre-compute and reuse the plans for different data graphs to eliminate this preprocessing time. We also note that our planning is much more efficient than DISC. DISC takes 56 seconds for 5-node queries, 1,255 seconds for 6-node queries, and 13,533 seconds for 7-node queries. We observe that the time for tree decomposition dominates when $k = 6$ or 7. This is because we use a brute-force approach to enumerate and check all tree decompositions. To scale to larger query graphs, we can reduce this time by applying existing tools [32] to find one decomposition with minimum width instead of checking all decompositions.



(a) varying data size, SO (b) varying query size

Figure 17: Scalability test

Scalability Test. Fig. 17(a) shows the running time of three approaches with respect to the number of edges in the SO graph. We randomly divided the edges of SO into 5 equally sized groups and created 5 graphs for the cases of 20%, 40%, 60%, 80%, and 100% edges. The $i+1$-th graph contains the i-th graph. DISC runs out of memory in all cases, and EVOKE runs out of memory when we use 80% or all edges. SCOPE scales better than EVOKE when increasing the data size. Fig. 17(b) shows the running time of SCOPE in three types of queries with sizes 4-9. For cycles, we randomly select o since all nodes are in the same orbit. For paths, we select the end node as o. Wheels are graphs made of a k-1 cycle and one center node connecting to other k-1 nodes. we select the center node as o. While existing approaches only support (unlabeled) queries with 6 nodes in practice, SCOPE supports queries with 9 nodes. Due to the hard nature of the subgraph counting problem, the running time grows exponentially with respect to the query size.

model	GCN	GraphSage	GAT	MoNet	GatedGCN
Baseline	73.67±0.22	70.92±0.123	78.95±0.12	85.78±0.04	85.61±0.05
$\mathcal{F}$-MPNN	79.66±1.65	85.74±0.37	84.87±0.22	86.54±0.02	85.72±0.38
$\mathcal{F}^+$-MPNN	**84.17±1.25**	**86.20±0.04**	**86.10±0.33**	**86.56±0.03**	**86.03±0.04**

Table 8: Accuracy for the PATTERN dataset

Applying subgraph counts to vertex classification. We study the effect of augmenting node features with local subgraph counts for the vertex classification task. The objective is to predict for every node whether it belongs to a fixed pattern P using PATTERN dataset [28] which has 14,000 graphs. Table 8 shows the results. Baseline uses a random value from $\{0, 1, 2\}$ as a feature. $\mathcal{F}$-MPNN [10] adds 3-5 clique counts, and we further add 464 non-zero counts taken from all 5/6-node patterns that $\mathcal{F}$-MPNN does not use, denoted as $\mathcal{F}^+$-MPNN. $\mathcal{F}^+$-MPNN has a much higher accuracy, outperforming Baseline and $\mathcal{F}$-MPNN in all cases.