Graph Perturbation Analysis for Subgraph Counting

Hanhua Xiao
Singapore Management University
Singapore, Singapore
hhxiao.2020@phdcs.smu.edu.sg

Yuchen Li
Singapore Management University
Singapore, Singapore
yuchenli@smu.edu.sg

Kyriakos Mouratidis
Singapore Management University
Singapore, Singapore
kyriakos@smu.edu.sg

Abstract

Subgraph counting, which involves determining the frequency of a query graph within a data graph, has numerous applications such as query optimization, fraud detection, and evaluating the expressiveness of graph neural networks. Despite its importance, there has been no systematic study on the impact of adversarial graph perturbations on subgraph counts. In this work, we examine the kSub problem, which aims to identify k edge additions that maximize the count of a query graph. We prove that kSub is intractable due to its NP-hardness, even for constant approximation. To address this, we relax the problem into a top-k selection, termed topkSub. Depending on the structure of the relaxed query graph, we distinguish two possible processing scenarios (namely, the *connected* and the *disconnected* case), and design dedicated search space pruning strategies for each scenario. Additionally, we develop sampling techniques on the pruned search space to scale topkSub for handling large graphs. In the form of a case study, we demonstrate that topkSub effectively uncovers vulnerabilities in state-of-the-art GNN models for subgraph counting, providing a significant advantage over alternative graph perturbation methods. The efficiency analysis shows that our pruning techniques achieve substantial speedups for exact processing in both connected and disconnected cases, while our sampling methods reduce estimation errors by 1-2 orders of magnitude compared to state-of-the-art samplers.

CCS Concepts

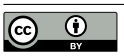
• **Information systems → Graph-based database models**.

Keywords

Graph perturbation; Subgraph counting

ACM Reference Format:
Hanhua Xiao, Yuchen Li, and Kyriakos Mouratidis. 2026. Graph Perturbation Analysis for Subgraph Counting. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V.1 (KDD '26), August 09–13, 2026, Jeju Island, Republic of Korea.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3770854.3780256

Resource Availability:
The source code of this paper has been made publicly available at https://doi.org/10.5281/zenodo.18060323.



KDD '26, Jeju Island, Republic of Korea

ACM ISBN 979-8-4007-2258-5/2026/08
https://doi.org/10.1145/3770854.3780256

1 Introduction

Graph perturbation analysis examines how structural changes, such as the addition or removal of nodes/edges, may impact key properties, like clustering coefficient, network diameter and centrality. That analysis provides insights into the resilience and efficiency of different systems. Real-world applications include power grids [46], where perturbation analysis helps prevent cascading failures and optimize energy flow. In transportation networks [7], it assists in managing disruptions and improving traffic flow. In knowledge graphs [64], it enables auditing the robustness of outstanding facts, helping avoid jumping to conclusions from incomplete context. Additionally, perturbation analysis has become increasingly relevant in studying the vulnerability of Graph Neural Networks (GNNs) [61, 65, 73, 74]. Specifically, *adversarial attacks* [28, 51], i.e., fake relationships in social networks, could be injected into the training data graph and the GNN models could be fooled by the topological changes from the adversarial perturbations. By analyzing how changes in a network's topology impact the accuracy of GNN models, researchers can develop more robust defenses to secure these models against manipulation and ensure their reliability in real-world applications.

Subgraph counting, the problem of enumerating isomorphic instances of a query graph Q within a data graph G [27, 30, 41, 62], is foundational to numerous applications. It plays a critical role in cardinality estimation for query optimization [43, 49, 67], where accurate subgraph counts directly influence the selection of efficient query execution plans. In fraud detection, recurring subgraph patterns may serve as indicators of anomalous behavior in financial transactions or social networks [20, 68]. Additionally, subgraph counting is a key tool for assessing the learning capabilities of GNNs, as their capacity to identify and distinguish substructures reflects the models' overall expressiveness [29, 63, 66, 71].

Despite the broad relevance of this problem, little attention has been paid to understanding how structural perturbations affect subgraph counts. Analyzing these perturbations can offer valuable insights across several domains. In query optimization, the query plan selection is highly relevant to cardinality estimators [26, 37]. These adversarial perturbations could reveal sensitivity of the estimators and their influence on query plan selection. In fraud detection, pattern counts serve as input features for downstream classifiers [4, 50]. Therefore, perturbation analysis can help assess the robustness of significant subgraph patterns, determining how changes in network structure (e.g., potential undocumented/unregulated fund transfers) impact pattern counts and, consequently, the performance of classifiers. Finally, graph perturbations can be employed as adversarial attacks to systematically evaluate GNN expressiveness [61, 65, 74], by measuring the accuracy of predicted subgraph counts in the presence of network perturbations in the training data.

In this paper, we study graph perturbation in the context of subgraph counting. To our knowledge, this is the first work on perturbation analysis in that context. As an initial focus, we consider the most common form of perturbation, i.e., edge additions [6, 10], as adversarial attacks. In particular, we study the kSub problem to identify k edge additions to the data graph G that maximize the number of instances of the query graph Q in G. We show that kSub is not only NP-hard, but also does not have a polynomial-time constant approximation, through a reduction from MAX-CLIQUE [21]. Due to the hardness result of kSub, we relax it to the topkSub problem, where we find the top-k edge additions that match a specific edge e_Q of the query graph Q, such that the number of instances of Q in G is maximized.

A naïve approach for topkSub enumerates all possible edge additions and calculates their respective perturbation value (i.e., the number of new instances of Q they induce in G). This becomes impractical for large graphs due to the vast search space of $O(|V_G|^2)$ possible edge additions. To avoid exhaustive search, an alternative is to find all instances of a *relaxed query graph* $Q^* = Q \setminus e_Q$ and then exclude all instances of Q to determine the perturbation value for each candidate edge addition. While this method solves topkSub, it performs redundant instance enumerations, because every instance of Q is also an instance of Q^*. Additionally, the instances of Q^* can be significantly more than those of Q, which exacerbates the inefficiency of that approach.

To address these challenges, we propose optimizations for both *connected* and *disconnected* relaxed queries. For *connected relaxed queries*, we reduce redundant computations by jointly enumerating instances of Q and Q^*, and use a novel *look-ahead* technique for early termination when no valid edge addition can match e_Q. For *disconnected relaxed queries*, we apply the *inclusion-exclusion* principle to prune the large search space of $O(|V_G|^2)$ possible edge additions. Furthermore, we design efficient samplers over the pruned search space for the connected and for the disconnected case, tailored at large graphs where our exact techniques may be expensive.

Our technical contributions are summarized as follows:

- To our best knowledge, this is the first study on graph perturbation for subgraph counting. We show that the combinatorial problem (kSub) of selecting k concurrent edge additions is intractable, thereby relaxing it to a top-k selection problem (topkSub).
- We categorize the relaxed queries Q^* into two distinct cases, i.e., connected and disconnected. We design optimizations tailored for each case to effectively prune the enumeration space. Moreover, we introduce effective samplers to scale topkSub for large graphs.
- We carefully design and implement baselines using state-of-the-art exact and sampling solutions for subgraph counting. We demonstrate that our advanced enumeration approaches significantly outperform the exact baselines. On the other hand, our sampling strategies achieve 1-2 orders of magnitude lower approximation error compared to the baseline samplers, all within the same 10-minute budget.
- We further conduct an application study to assess the robustness of GNNs built specifically for subgraph counting, using our perturbation analysis. The study shows that our methods can uncover strong adversarial attacks on GNNs, amplifying their prediction errors by multiple orders of magnitude.

2 Preliminaries

An undirected graph is denoted as $g = (V_g, E_g, L_g)$, where V_g and E_g are the vertex set and the edge set, respectively. Function L_g maps each vertex $v \in V_g$ to its label $L_g(v)$. In this work, we focus on undirected vertex-labeled graphs, but the proposed techniques generalize to handling edge labels as well as directed graphs. For a vertex v, $N(v)$ denotes the set of neighbors of v. Consider a *connected* query graph $Q = (V_Q, E_Q, L_Q)$ and a data graph $G = (V_G, E_G, L_G)$. We refer to the vertices and edges of Q as *query vertices/edges*, and to those of G as *data vertices/edges*. Next, we introduce key concepts for subgraph counting.

Definition 2.1 (Subgraph Matching). If a subgraph g of G is isomorphic to a graph Q, then g is an *instance* of Q in G.

Definition 2.2 (Matching Order). Given a query graph Q, a matching order φ is a query vertex sequence used for iterative vertex matching of data graph instances. We use $\varphi[i]$ to denote the i^{th} query vertex to be matched, and $\varphi(u)$ to denote the order of u in φ.

Definition 2.3 (Partial / Full Instances). A *partial* instance M (of Q in G) is a sequence of $|M| < |\varphi|$ specific data vertices that match (in terms of vertex labels and connectivity) the query Q up to the $|M|$-th position in the matching order φ. If $|M| = |\varphi|$, M is a (full) instance. For a query vertex u, we use $M(u)$ to denote the data vertex in M that u maps to (i.e., the $|\varphi(u)|$-th data vertex in M). $\mathcal{M}(Q, G)$ denotes the set of all (full) instances of Q in G. We drop G and use $\mathcal{M}(Q)$ when the context is clear.

Definition 2.4 (Backward/Forward Neighbor). Given a matching order φ, the backward neighbors $N^{\varphi}_{<}(u)$ of a query vertex u are the neighbors of u in Q that appear before u in φ. On the contrary, the forward neighbors $N^{\varphi}_{>}(u)$ of u are the neighbors of u in Q that appear after u in φ.

Definition 2.5 (Candidate Sets and Candidate Graph). For each query vertex u, the global candidate set $C(u)$ is the set of data vertices v that have the same label, i.e., $L_G(v) = L_Q(u)$. Given a query edge $e_Q = (u, u')$ and a candidate vertex $v \in C(u)$, the set $C(u'|u, v)$ represents the neighbors of v in $C(u')$. In other words, $C(u'|u, v)$ includes all the candidate matches of u' in the data graph with respect to query edge $e_Q = (u, u')$ when data vertex v plays the role of u. We use $C(Q, G)$ to represent the candidate graph of Q in G. It contains the global candidate sets for all $u \in V_Q$ as well as any data edge (v, v') that matches a query edge $e_Q = (u, u')$, i.e., $L_G(v) = L_Q(u)$ and $L_G(v') = L_Q(u')$.

Definition 2.5 is a simplified version of the candidate graph to streamline our explanation. Candidate graph construction is orthogonal to the techniques proposed in this work. In our implementation, we use state-of-the-art methods to construct candidate graphs to effectively prune candidates from the candidate graph [5, 9, 25, 52, 67].

Example 2.6. Figure 1 presents a running example of a query graph Q and a data graph G. Assume that the matching order is $\varphi = (u_4, u_1, u_3, u_2, u_5, u_6)$, the backward neighbors of u_3 are $\{u_1, u_4\}$ since they come before u_3 in φ and are neighbors of u_3 in Q. To derive the candidate graph $C(Q, G)$, we first construct a global candidate set $C(u)$ for each query vertex. For instance, $C(u_1) =$

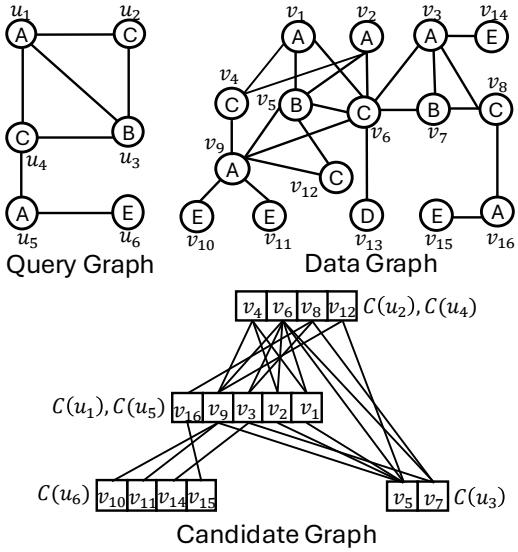


Figure 1: Example of Q, G, and $C(Q, G)$.

$\{v_1, v_2, v_3, v_9, v_{16}\}$ as $\forall v \in C(u_1), L_G(v) = L_Q(u_1) = A$. Next, for every query edge $e_Q = (u, u')$ (e.g., edge (u_1, u_3)), we link those members of $C(u)$ and $C(u')$ that are adjacent in the data graph (e.g., the data edge between $v_1 \in C(u_1)$ and $v_5 \in C(u_3)$ is retained in the candidate graph). The instances of Q in G can be retrieved by traversing $C(Q, G)$ according to φ (e.g., using depth-first search); discovered partial instances are incrementally expanded and, if they successfully reach length $|\varphi|$, are reported/added to set $\mathcal{M}(Q, G)$. In our example, $\mathcal{M}(Q, G) = \{\{(u_4, v_6), (u_1, v_3), (u_3, v_7), (u_2, v_8), (u_5, v_9), (u_6, v_{10})\}, \{(u_4, v_6), (u_1, v_3), (u_3, v_7), (u_2, v_8), (u_5, v_9), (u_6, v_{11})\}, \{(u_4, v_8), (u_1, v_3), (u_3, v_7), (u_2, v_6), (u_5, v_{16}), (u_6, v_{15})\}, \{(u_4, v_6), (u_1, v_9), (u_3, v_5), (u_2, v_{12}), (u_5, v_3), (u_6, v_{14})\}\}$.

3 Perturbation Analysis for Subgraph Counting

In this paper, we focus on edge additions [6, 10] for perturbation analysis. Specifically, we aim to identify perturbations of k edge additions to the data graph G that maximize the count of subgraph instances for the query graph Q. We leave other forms of perturbation as future work. We formally define the problem of k-Perturbation for Subgraph Counting (kSub) as follows.

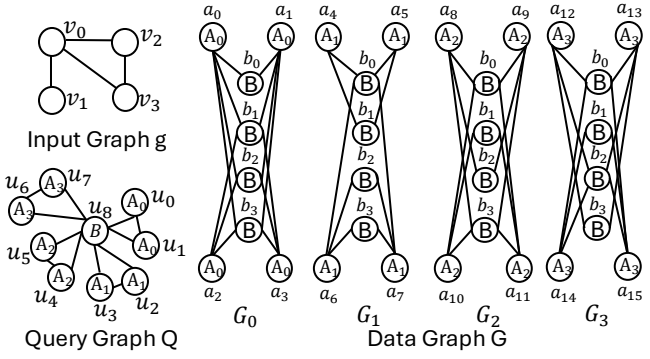


Figure 2: kSub instance corresponding to MC problem on g.

Definition 3.1 (kSub). Given a data graph G and a query graph Q, find a set of k edge additions for G as $\Delta E = \{\Delta e_1, \cdots, \Delta e_k\}$ to maximize the *perturbation value* $|\Delta\mathcal{M}(\Delta E)|$, where $\Delta\mathcal{M}(\Delta E) = \mathcal{M}(Q, G \cup \Delta E) \setminus \mathcal{M}(Q, G)$ is the set of new instances after adding ΔE to the data graph G.

Theorem 3.2. *The kSub problem is NP-hard.*

Proof. We demonstrate the NP-hardness of the kSub problem through a polynomial-time reduction from the NP-hard MAX-CLIQUE (MC) problem [21]. Given a graph g with n vertices, MC identifies the subset $\mathcal{S}$ of g's vertices with maximum cardinality, such that every pair of vertices in $\mathcal{S}$ are adjacent. From an instance of MC on g, we shall formulate the corresponding instance of kSub, starting with the construction of the query graph Q and the data graph G. We will use the 4-vertex graph g in Figure 2 as an example.

First, we construct Q. For each $v_i \in V_g$, we create a pair of nodes u_{2i} and u_{2i+1} with label A_i (where $A_i \neq A_j, \forall j \neq i$). Then, we create a center node u_{2n} with label B. For each pair u_{2i}, u_{2i+1}, we add edges (u_{2i}, u_{2i+1}), (u_{2i}, u_{2n}) and (u_{2i+1}, u_{2n}). That is, every u_{2i}, u_{2i+1} pair forms a triangle with the center node u_{2n}.

Next, we construct G. It contains n components, i.e., $G = \cup_{0 \leq i < n} G_i$ where each G_i corresponds to v_i of g. First, we create n center nodes with label B, denoted as $b_1, b_2, \cdots, b_n$, which are shared among all G_i components. Then, for each G_i, we create 2 pairs of nodes with label A_i denoted as $[a_{4i}, a_{4i+1}]$ and $[a_{4i+2}, a_{4i+3}]$. Next, we build wedges for these two pairs. Specifically, we add edges (a_{4i}, b_j), (a_{4i+1}, b_j) to G_i, where $j \in \{x \mid v_x \in N(v_i) \text{ or } x = i\}$. Similarly, we add (a_{4i+2}, b_p) and (a_{4i+3}, b_p) to G_i, where $p \in \{x \mid x \neq i\}$. The complexity of edge construction is $O(n^2)$. Combining the construction of Q and G yields a time complexity $O(n^2)$ indicating that this is a polynomial-time reduction.

We will show that solving kSub on Q and G for $k = n$ is equivalent to solving MC on g. In order for at least one matching instance of Q to exist in G, there should be at least one triangle in every G_i. Since $k = n$, this implies that exactly one edge should be added to each G_i. Let B_i denote the set of label-B nodes that participate in a triangle in G_i after the edge addition. The number of matching instances of Q in the (updated) G is $|\bigcap_{0 \leq j < n} B_j|$, which kSub aims to maximize. A crucial deduction is that the edge added in G_i should be either (a_{4i}, a_{4i+1}) or (a_{4i+2}, a_{4i+3}) (because the set B_i induced in either of these cases is a strict superset of the B_i induced if we link one member of $[a_{4i}, a_{4i+1}]$ with one member of $[a_{4i+2}, a_{4i+3}]$).

To complete the proof, we show that every possible clique $\mathcal{S}$ in g has a one-to-one correspondence with a set of $k = n$ viable edge additions in G, and that $|\mathcal{S}| = |\bigcap_{0 \leq j < n} B_j|$ after these additions. Since kSub maximizes $|\bigcap_{0 \leq j < n} B_j|$, it also identifies the maximum clique. Specifically, a given clique $\mathcal{S}$ corresponds to (I) adding edge (a_{4i}, a_{4i+1}) to each G_i where $v_i \in \mathcal{S}$, and (II) adding edge (a_{4i+2}, a_{4i+3}) to each G_i where $v_i \notin \mathcal{S}$. In the (I) case, label-B node b_i will definitely be in $\bigcap_{0 \leq j < n} B_j$, because it is in B_i and also in every B_j for $j \neq i$ (if $v_j \notin \mathcal{S}$, edge (a_{4j+2}, a_{4j+3}) forms a triangle with b_i, and if $v_j \in \mathcal{S}$ then $v_i \in N(v_j)$ and thus edge (a_{4j}, a_{4j+1}) forms a triangle with b_i). In the (II) case, label-B node b_i is not in B_i and hence not in $\bigcap_{0 \leq j < n} B_j$. In other words, $b_i \in \bigcap_{0 \leq j < n} B_j$ iff $v_i \in \mathcal{S}$ and $|\mathcal{S}| = |\bigcap_{0 \leq j < n} B_j|$. □

Furthermore, given the inapproximability of MC [72], kSub in our reduction asserts that:

Corollary 3.3. *The kSub problem does not have a polynomial-time ($n^{1-\epsilon}$)-approximation, for any $\epsilon > 0$, unless P = NP.*

The hardness of kSub is mainly due to the combinatorial nature of selecting k edges for addition to G, out of $O\left(\binom{|V_G|^2}{k}\right)$ possible perturbations. Furthermore, each added edge can be matched to multiple query edges in multiple matching instances. As such, we relax the problem by investigating a given query edge $e_Q \in E_Q$ and find the top-k edge additions that match e_Q and induce the largest number of new instances of Q in G. One could investigate all query edges in E_Q in isolation, and use the aggregated top-k results as a heuristic solution for kSub. We formally introduce the problem of top-k-Perturbation for Subgraph Counting (topkSub) as follows.

Definition 3.4 (topkSub). Given a data graph G, a query graph Q, and a query edge $e_Q \in E_Q$, find the top-k edge additions for G as $\Delta E = \{\Delta e_1, \cdots, \Delta e_k\}$ such that $|\Delta\mathcal{M}(\Delta e^*)| \geq |\Delta\mathcal{M}(\Delta e)|$, where edges $\Delta e^* \in \Delta E$ and $\Delta e \notin \Delta E$ are both matched to e_Q for all instances in $\Delta\mathcal{M}(\Delta e^*)$ and $\Delta\mathcal{M}(\Delta e)$, respectively.

Solution Overview. Solving topkSub remains extremely challenging. A naïve approach would be to compute the perturbation value $|\Delta\mathcal{M}(\Delta e)|$ for every possible edge addition Δe. However, this is impractical due to the vast search space of $O(|V_G|^2)$ potential edge additions. To avoid an exhaustive search, we observe that for any edge addition Δe, $|\Delta\mathcal{M}(\Delta e)|$ can be quantified by counting the number of instances in $\mathcal{M}(Q^*) = \mathcal{M}(Q^*, G)$ that do not appear in $\mathcal{M}(Q) = \mathcal{M}(Q, G)$, where $Q^* = Q \setminus e_Q$ is called the *relaxed query graph*. These instances of Q^* are missing only Δe to complete the match with e_Q and form full instances of Q. Hence, a viable strategy is to use a subgraph matching technique to compute $\mathcal{M}(Q^*)$ and $\mathcal{M}(Q)$. Then, by scanning through all instances in $\mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$, the top-$k$ edge additions with the highest perturbation values can be identified. We call this improved strategy RelaxEnum.

On the downside, RelaxEnum results in redundant enumeration, because every instance of Q is also an instance of Q^*. Furthermore, the number of instances of Q^* can be substantially larger than those of Q, making the enumeration process inefficient. In the following, we examine two types of relaxed queries and propose optimizations to reduce unnecessary enumerations. In particular, Section 5 focuses on the case where Q^* is *connected*, while Section 6 addresses the scenario where Q^* is *disconnected*. To aid our discussion, we establish a convention for labeling the vertices of edge e_Q.

Definition 3.5 (Head/Tail Vertex). Given an edge $e_Q = (u_h, u_t)$ to be removed from Q to form Q^*, we assume that u_h (head vertex) precedes u_t (tail vertex) in the matching order φ, i.e., $\varphi(u_h) < \varphi(u_t)$.

4 Related Work

Subgraph Counting. As subgraph counting evolved over the years, a multitude of algorithms and methods were developed that address the problem in different ways and for distinct purposes, categorized into exact and approximate. Many pioneering studies [27, 30, 41, 62] count the exact frequency of a subgraph within a data graph by backtracking enumeration. For approximate counting, there are three general approaches. Analytics-based methods [3, 42, 45, 48] compute and aggregate the counts of small queries to estimate the count of a larger query. Sampling-based techniques [33, 39, 47, 49, 67] estimate the count by traversing only a sample of the possible instances from the search space. Learning-based approaches [59, 71] utilize graph neural networks (GNNs) [24, 34, 57] to estimate the count. To our best knowledge, this work provides the first systematic perturbation analysis for subgraph counting, which functions as an adversarial attack on approximation methods and evaluates their robustness.

Subgraph Matching. Instead of just counting, subgraph matching reports all the instances of the query graph in the data graph. Current mainstream approaches [5, 25, 32, 52] follow the *filter-order-enumerate* procedure. That is, they first filter vertices for each candidate set to build a candidate graph. Then, they determine the matching order by a cost model. Finally, they enumerate the candidate graph using backtracking along the matching order to obtain match instances. Meanwhile, there has been a line of study on incremental subgraph matching by finding $\Delta\mathcal{M}$ in response to updates ΔE to G by minimizing redundant enumeration [16–18]. More recently, subgraph matching has also been considered for dynamic data graphs, termed continuous subgraph matching (CSM). Existing CSM approaches [38] model the query as a natural join of all query edges and evaluate the changes incurred by the updates based on the delta-rule [2, 36]. That is distinct from our problem, since we seek to *identify* the perturbations that maximize the delta match rather than applying a *given* set of graph updates.

Robustness and Graph Perturbation. Network robustness refers to the ability of a network to remain functional when it is attacked or partially damaged [8, 12, 13, 15, 19]. Common robustness measures, be them connectivity-based [35, 56, 58] or spectrum-based [11, 14, 54], evaluate how drastically the global properties of the network are affected by graph perturbations (typically, removal of edges or nodes [13, 14, 31]). On the other hand, edge addition [8] or rewiring [12] are considered as a defense mechanism to failures, aiming to preserve global network characteristics. Other than robustness, perturbation has also been used for data privacy/protection [55], i.e., adding a noise graph (according to a probability distribution) to the data graph before its release/publishing. All aforementioned studies of graph perturbation consider global graph properties and assess the data graph in its entirety. Conversely, our problem regards a query-specific property rather than the general characteristics of the data graph. Perturbation analysis has also been studied to simulate adversarial attacks that deceive GNN models and reveal their limitations [65, 73, 74]. Since subgraph counting is a critical metric for evaluating the learning capacity and expressiveness of GNNs [59, 69, 71], our proposed analysis focuses on identifying specific graph perturbations that can effectively identify weaknesses in the model's ability to generalize and accurately capture complex subgraph patterns.

5 Perturbation by Connected Relaxed Query

In this section, we address topkSub in the case where the relaxed query $Q^* = Q \setminus (u_h, u_t)$ is *connected*. We first propose a technique (JointEnum) to avoid the redundant enumeration of both $\mathcal{M}(Q^*)$ and $\mathcal{M}(Q)$ by focusing only on the difference set $\mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$. Next, we present a strategy (LookAheadEnum) to further prune the search space by introducing a *look-ahead set* that terminates early the enumeration of partial instances incapable of forming a

full instance in $\mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$. Both types of pruning are effective and lead to exact TOPKSUB answers. However, for responsiveness in large problem instances, we additionally propose a sampling technique that builds on the previous two and efficiently approximates the solution to TOPKSUB.

Joint Enumeration Process. We base the joint enumeration process (JointEnum) on the existing depth-first search (DFS) paradigm for subgraph matching [5, 25, 52]. For each matching position i, DFS maintains a current partial instance M and identifies a set of candidate data vertices S that could match the next query vertex u, according to a predefined matching order φ. Set S is formed by intersecting the neighbors of already matched vertices with the backward neighbors of u. JointEnum uses a common DFS tree for both $\mathcal{M}(Q^*)$ and $\mathcal{M}(Q)$, since their individual DFS trees would be *identical* until the tail vertex u_t needs to be matched. Importantly, we *exclude any matches* where the data edge (v_h, v_t) already exists in G. The reason is that common matches of $\mathcal{M}(Q^*)$ and $\mathcal{M}(Q)$ are useless for the retrieval of the difference set $\mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$.

Algorithm 1: Joint Enumeration (JointEnum)

Input: head vertex u_h, tail vertex u_t, candidate graph $C(Q^*, G)$, matching order φ, matching position i, current instance M.
Output: perturbation values $|\Delta\mathcal{M}(\Delta e)|$ for any Δe.

1 $u \leftarrow \varphi[i]$;
2 **if** $i = 0$ **then** $S \leftarrow C(u)$;
3 **if** $i = |\varphi|$ **then**
4 $\quad \Delta e = (M(u_h), M(u_t))$;
5 $\quad |\Delta\mathcal{M}(\Delta e)|$++;
6 $\quad$ **return**;
7 **if** $u = u_t$ **then** $S \leftarrow \cap_{u' \in N^{\varphi}_{<}(u)} C(u \mid u', M(u')) \setminus C(u \mid u_h, M(u_h))$;
8 **else** $S \leftarrow \cap_{u' \in N^{\varphi}_{<}(u)} C(u \mid u', M(u'))$;
9 **foreach** $v \in S$ **do**
10 $\quad$ **if** $v \notin M$ **then**
11 $\quad\quad M' \leftarrow M \cup (u, v)$;
12 $\quad\quad$ JointEnum $(u_h, u_t, C(Q^*, G), \varphi, i+1, M')$;

JointEnum is detailed in Algorithm 1. We first generate the candidate graph for the connected relaxed query $C(Q^*, G)$ and the matching order φ, following the methods described in a recent survey on subgraph matching [70]. The enumeration process begins with the first query node $u = \varphi[0]$, initializing the set of data vertices to be matched to u as the global candidates $C(u)$ from the candidate graph. As elaborated previously, when matching the tail vertex u_t, we exclude from set S all neighbors of data vertices matched to the head vertex u_h to avoid redundant enumeration (Line 7). We otherwise follow the standard DFS paradigm to determine set S and to recursively traverse the search tree to discover instances in $\mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$ (Lines 8-12). Once a full match M is found, we increment the perturbation value $|\Delta\mathcal{M}(\Delta e)|$ for the edge $\Delta e = (M(u_h), M(u_t))$. An example is presented in Appendix A.1 to demonstrate JointEnum. By the end of the enumeration process, we can identify the top-k edges with the highest perturbation values.

Look-Ahead Set. Although JointEnum avoids the redundant enumeration of instances of Q, the remaining search space still contains numerous partial instances in $\mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$ to consider. To prune partial instances, we introduce the *look-ahead set* for the tail vertex u_t, i.e., we proactively compute the candidate set for u_t using matched vertices in a partial instance M and early terminate the enumeration of M once we find that the candidate set for u_t is empty. During the enumeration process, we maintain an up-to-date look-ahead set S_t to track the current candidates for the tail vertex u_t. When a data vertex v is matched to a backward neighbor u of u_t, we proactively update the candidates for u_t by refining the look-ahead set to $S_t \cap C(u_t|u, v)$. Similarly, when a data vertex v is matched to the head vertex u_h, we prune the candidates for u_t by updating the set to $S_t \setminus C(u_t|u_h, v)$. The updated look-ahead set is recursively passed into the enumeration process. If at any point the look-ahead set S_t becomes empty, we terminate the current partial match and backtrack in the enumeration.

Algorithm 2: Look Ahead Enumeration (LookAheadEnum)

Input: head vertex u_h, tail vertex u_t, candidate graph $C(Q^*, G)$, matching order φ, matching position i, current instance M, look-ahead set S_t.
Output: perturbation values $|\Delta\mathcal{M}(\Delta e)|$ for any Δe.

1 $u \leftarrow \varphi[i]$;
2 **if** $i = 0$ **then**
3 $\quad S \leftarrow C(u)$;
4 $\quad S_t \leftarrow C(u_t)$;
5 **if** $i = |\varphi|$ **then**
6 $\quad \Delta e = (M(u_h), M(u_t))$;
7 $\quad |\Delta\mathcal{M}(\Delta e)|$++;
8 $\quad$ **return**;
9 **if** $u = u_t$ **then** $S \leftarrow S_t$;
10 **else** $S \leftarrow \cap_{u' \in N^{\varphi}_{<}(u)} C(u \mid u', M(u'))$;
11 **foreach** $v \in S$ **do**
12 $\quad$ **if** $v \notin M$ **then**
13 $\quad\quad M' \leftarrow M \cup (u, v)$;
14 $\quad\quad$ **if** $u = u_h$ **then**
15 $\quad\quad\quad S'_t \leftarrow S_t \setminus C(u_t|u_h, v)$;
16 $\quad\quad$ **else if** $u \in N^{\varphi}_{<}(u_t)$ **then**
17 $\quad\quad\quad S'_t \leftarrow S_t \cap C(u_t|u, v)$;
18 $\quad\quad$ **if** $S'_t \neq \emptyset$ **then**
19 $\quad\quad\quad$ LookAheadEnum $(u_h, u_t, C(Q^*, G), \varphi, i+1, M', S'_t)$;

The enhanced enumeration process, termed LookAheadEnum, is detailed in Algorithm 2. To begin, we initialize S to the candidate set of the first query vertex $u = \varphi[0]$, and the look-ahead set S_t to the (global) candidate set of u_t. In Line 9, we skip the computation for the candidate set S of u_t (unlike in Line 7 of Algorithm 1) because the look-ahead set S_t has already been computed before reaching u_t. For each vertex v matched to the current query vertex u, we not only extend the partial match to M', but also update the look-ahead set if $u = u_h$ or $u \in N^{\varphi}_{<}(u_t)$ (Lines 13-17). The enumeration process proceeds with M' only if the updated look-ahead set is non-empty.

It is important to note that LookAheadEnum pushes forward the computation of candidates for u_t as the search progresses, whereas JointEnum computes the candidates for u_t only when the enumeration reaches u_t. This approach allows for early termination at the cost of additional memory required to maintain the look-ahead sets. However, the overhead of maintaining these look-ahead sets is bounded by $O(|\varphi| \cdot |V_G|)$. This is because the DFS paradigm maintains a single partial match M at any given time, with each matched data vertex (up to $|\varphi|$ vertices) having a corresponding look-ahead set. Each look-ahead set is limited by the size of the global candidate set $C(u_t)$ for the tail vertex, which is at most $|V_G|$.

Hybrid Sampling. Inspired by existing random walk (RW) sampling approaches for subgraph counting [33, 39, 67], we draw RW samples from the enumeration space that remains after LookAheadEnum's pruning, and estimate $|\Delta\mathcal{M}(\Delta e)|$ for every valid edge addition Δe discovered through sampling. Specifically, instead of

matching all candidate nodes in S to the query node u at matching position i (Lines 11-19 of LookAheadEnum), we randomly select a single node v from S to match u. Once a partial/full match M is sampled, the perturbation value $|\Delta\mathcal{M}(\Delta e)|$ is estimated as: $X_M = \mathbb{I}\{\Delta e \in M\} \cdot \prod_{i=0}^{|\varphi|} d_i$, where $\mathbb{I}\{\Delta e \in M\}$ is an indicator random variable that equals 1 if M contains Δe, and d_i represents the size of the candidate set S at matching position i while sampling M.

Theorem 5.1. *X_M is an unbiased estimator for $|\Delta\mathcal{M}(\Delta e)|$ for any perturbation Δe, given that M is a random walk sample from the search space of LookAheadEnum.*

Proof. We can decompose any instance $M \in \mathcal{M}(Q^*) \setminus \mathcal{M}(Q)$ into two components M_1 and M_2, where M_1 is in $\mathcal{M}(Q_t^*) \setminus \mathcal{M}(Q_t)$ with $Q_t^* = Q_t \setminus \Delta e$, and M_2 is an instance of $Q \setminus Q_t$, thereby:

$$|\Delta\mathcal{M}(\Delta e)| = \sum_{M \in \mathcal{M}(Q^*) \setminus \mathcal{M}(Q)} \mathbb{I}\{\Delta e \in M\}$$
$$= \sum_{\substack{M_1 \in \mathcal{M}(Q_t^*) \setminus \mathcal{M}(Q_t) \\ M_2 \in \mathcal{M}(Q \setminus Q_t)}} \frac{\mathbb{I}\{\Delta e \in M_1\} \times \mathbb{I}\{M_1 \cup M_2 \in \mathcal{M}(Q^*)\}}{\prod_i d_i} \prod_i d_i$$

Since $\frac{1}{\prod_i d_i}$ is the sampling probability of instance $M = M_1 \cup M_2$, the estimator is unbiased as it resembles the Horvitz-Thompson (HT) estimator [33]. □

With finite samples, RW systematically underestimates because complete subgraphs are rarely observed; the bias worsens as query size grows [33, 67]. To address the underestimation issue, we propose a hybrid sampling approach that leverages both the accuracy of enumeration and the efficiency of sampling. Specifically, we begin by identifying the *densest subgraph* Q_D^* within the relaxed query Q^*. The high density of Q_D^* suggests that enumerating its instances can be highly efficient, as its edge constraints effectively prune invalid partial matches. Thus, we first enumerate all instances of Q_D^*, and subsequently sample the remaining nodes for the instances of $Q^* \setminus Q_D^*$. The hybrid estimator is unbiased because it combines exact enumeration with inverse-probability weighting. We enumerate all embeddings of the dense subgraph Q_D^* exactly (no sampling error), and for each enumerated prefix we sample the remaining vertices of $Q^* \setminus Q_D^*$ by making uniform choices from the feasible candidate set at each step. Each sampled completion is then weighted by the inverse of its sampling probability (HT estimator), so the expected weighted count for each prefix equals its true number of valid completions (including the candidate edge). Since every full match extends exactly one enumerated prefix and all prefixes are covered, summing these unbiased per-prefix estimates yields an unbiased estimate overall. The Look-Ahead Set only prunes prefixes with no feasible completion and thus does not change the inclusion probability of any valid match.

6 Perturbation By Disconnected Relaxed Query

In this section, we address the case where the relaxed query $Q^* = Q \setminus (u_h, u_t)$ is *disconnected*. Since the removal of e_Q from Q renders Q^* disconnected, Q^* can be decomposed into two disjoint components, Q_h and Q_t, where $u_h \in V_{Q_h}$ and $u_t \in V_{Q_t}$. Although JointEnum from Section 5 could still be used, a disconnected Q^* leads to a prohibitively large search space, i.e., the Cartesian product of the individual search spaces for Q_h and Q_t, and a total complexity of $O(Cost(Q_h) \times Cost(Q_t))$, where $Cost(g)$ represents the cost of finding all instances of a query graph g in G.

To mitigate this overhead, we propose a more efficient enumeration strategy, named ProbeEnum. We pick either of the component graphs first, say Q_t, to enumerate and record all its instances (i.e., set $\mathcal{M}(Q_t)$). Next, we search for instances of Q_h in G, and for every instance M_h discovered as we go, we probe the stored instances of Q_t to identify matches $M_t \in \mathcal{M}(Q_t)$ such that $M_h \cup M_t$ forms an instance of Q^*. Since Q^* is disconnected, it suffices to verify that M_h and M_t do not share any common data vertex. Once such an instance $M_h \cup M_t$ is obtained, we increment the perturbation value of $\Delta e = (M_h(u_h), M_t(u_t))$. After enumerating all instances of Q^*, we can exactly determine the top-k edge additions for TopkSub.

ProbeEnum reduces complexity to $O(\text{Cost}(Q_h) + \text{Cost}(Q_t) + |\mathcal{M}(Q_h)| \times |\mathcal{M}(Q_t)|)$, as we enumerate the instances of Q_h and Q_t only once. The $O(|\mathcal{M}(Q_h)| \times |\mathcal{M}(Q_t)|)$ term accounts for probing. In the following, we first introduce an efficient pruning strategy that uses the *inclusion-exclusion principle* to focus on enumerating the most promising edge additions, thereby reducing the quadratic probing costs. Next, for large graphs where exact processing is expensive, we propose a *weighted perturbation sampling* add-on, which prioritizes sampling more promising edge additions and then employs the RW approach to estimate the perturbation values of sampled edge additions so as to bypass the probing process.

Inclusion-Exclusion Enumeration. To reduce the quadratic probing cost, we apply the inclusion-exclusion principle to derive upper and lower bounds for the perturbation value $|\Delta\mathcal{M}(\Delta e)|$ of any considered perturbation $\Delta e = (v_h, v_t)$ that matches (u_h, u_t). This allows us to disregard edges whose upper bound is no greater than any of the top-k lower bounds of valid perturbations found so far, because such edges cannot be part of the TopkSub result.

Let set $\mathcal{M}(g|v)$ represent all instances of a graph g in G that contain the data vertex v. We derive the *inclusion* upper bound for any valid perturbation (v_h, v_t) as $UB(v_h, v_t) = |\mathcal{M}(Q_h|v_h)| \cdot |\mathcal{M}(Q_t|v_t)|$. Furthermore, we derive the *exclusion* lower bound as: $LB(v_h, v_t) = UB(v_h, v_t) - \sum_{v' \in V_G} |\mathcal{M}(Q_h|v_h, v')| \cdot |\mathcal{M}(Q_t|v_t, v')|$, where $\mathcal{M}(g|v, v')$ denotes the instances of a graph g in G that contain both the data vertices v and v'.

The upper bound $UB(v_h, v_t)$ represents the maximum possible number of instances of $Q^* = Q_h \cup Q_t$ for a given perturbation (v_h, v_t), assuming that there is no overlap (i.e., no common data vertex) between any pair of Q_h and Q_t instances. It is an upper bound because the actual $|\Delta\mathcal{M}(\Delta e)|$ value discounts pairs of overlapping Q_h and Q_t instances, as they form invalid instances of Q^*. On the other hand, the lower bound $LB(v_h, v_t)$ subtracts from $UB(v_h, v_t)$ an *overestimate* of the number of overlapping pairs (Q_h, Q_t); to see this, the term $-\sum_{v' \in V_G} |\mathcal{M}(Q_h|v_h, v')| \cdot |\mathcal{M}(Q_t|v_t, v')|$ discounts multiple times (instead of just once) a pair of Q_h and Q_t instances that share more than one vertices v'.

Based on the bounds, we devise the *inclusion-exclusion* enumeration (IEEnum) for the TopkSub problem with the following steps:

(1) We first compute $|\mathcal{M}(Q_h|v_h)|$, $|\mathcal{M}(Q_t|v_t)|$, $|\mathcal{M}(Q_h|v_h, v')|$ and $|\mathcal{M}(Q_t|v_t, v')|$ for all $v_h \in C(u_h)$, $v_t \in C(u_t)$ and $v' \in V_G$, by enumerating all Q_h and Q_t instances only *once*, and record these counts in 4 hash tables for efficient look-up. Note that we keep the counts rather than storing the actual instances.

(2) We sort vertices $v_h \in C(u_h)$ and $v_t \in C(u_t)$ on descending $|\mathcal{M}(Q_h|v_h)|$ and $|\mathcal{M}(Q_t|v_t)|$, respectively. In a nested loop fashion, we go down the two lists, forming a new (v_h, v_t) pair in each iteration. Assume that the outer loop corresponds to $C(u_h)$ and the inner loop to $C(u_t)$. Throughout the process, we maintain the top-k lower bound values of pairs (i.e., valid perturbations) seen so far. If $UB(v_h, v_t)$ for the current pair is no greater than any of the top-k lower bounds, the pair is ignored (and the inner loop safely breaks, because any pairing of the current position in the $C(u_h)$ list with any position past v_t in the $C(u_t)$ list can only have an even lower upper bound). Otherwise, we record (v_h, v_t) as a candidate perturbation, compute $LB(v_h, v_t)$, and update accordingly the top-k lower bounds.

(3) Finally, we enumerate all remaining candidate perturbations Δe to compute the actual $|\Delta\mathcal{M}(\Delta e)|$ and report the top-k perturbations exactly.

Weighted Perturbation Sampling. Since Q^* is disconnected, we can first *uniformly* sample an edge $\Delta e = (v_h, v_t)$ from $C(u_h) \times C(u_t)$, and then apply the sampling strategy from Section 5, as the remaining query vertices are connected, effectively reducing the sampling space. However, since the sampling space of (u_h, u_t) is quadratic, i.e., $|C(u_h)| \times |C(u_t)|$, a large number of samples is required to capture the top-k perturbations. Furthermore, we must account for the high variance associated with the RW estimator.

Based on our empirical analysis, we find that the upper bound values serve as good proxies for the true perturbation values, especially for pruning candidates outside of the actual top-k perturbations (see Section 7.3 for the empirical results). Therefore, we first draw RW samples with respect to Q_h and Q_t to estimate the $|\mathcal{M}(Q_h|v_h)|$ and $|\mathcal{M}(Q_t|v_t)|$ counts, respectively. Note that the estimators for the statistics are unbiased. Next, we sample an edge $\Delta e = (v_h, v_t)$ with a probability proportional to its upper bound value $UB(v_h, v_t)$. Once (v_h, v_t) is selected, we use RW samplers to complete an instance M of Q^* by sampling the remaining nodes. By adapting Theorem 5.1, this allows us to develop an unbiased estimator for $|\Delta\mathcal{M}(\Delta e)|$ based on M. After gathering enough samples, we can accurately estimate the top-k perturbations for TopkSub using the law of large numbers.

7 Experiments

In this section, we present the empirical results of our perturbation analysis for subgraph counting. The results (efficiency and accuracy) of processing *connected* relaxed queries and *disconnected* relaxed queries are discussed in Section 7.2 and Section 7.3, respectively. Our application study assessing the robustness of a state-of-the-art GNN model for subgraph counting is presented in Section 7.4.

7.1 Experiment Setup

Datasets. We employ 8 real datasets that are widely used in existing subgraph counting studies [5, 9, 25, 52]. The characteristics of these datasets are presented in Table 1. Among these datasets, there are two labeled data graphs, i.e., Human and WordNet. For other graphs, we follow the approach of previous studies [5, 9, 25, 52, 67] to generate random vertex labels.

Query Generation. We follow the query generation process in [5, 9, 25, 52, 67] which randomly extracts subgraphs from the input data graph. We use query graphs of 4, 8, or 16 vertices. Each query set contains 200 connected query graphs with the same number of vertices. We process TopkSub for each query edge across all query graphs. The proportion of connected relaxed queries (CRQ) and disconnected relaxed queries (DRQ) for all query sizes is also presented in Table 1 as the relative ratio of DRQ to CRQ. We evaluate the following methods for both types of relaxed queries.

Table 1: The statistics of the datasets and queries.

Dataset	$\|V_G\|$	$\|E_G\|$	$\|L_G\|$	$\|V_{Q^*}\|$=4	$\|V_{Q^*}\|$=8	$\|V_{Q^*}\|$=16
DBLP	317,080	1,049,866	15	0.16	0.12	0.04
YouTube	1,134,890	2,987,624	25	2.82	0.08	0.06
Orkut	3,072,441	117,185,083	150	0.17	0.12	0.11
EU2005	862,664	16,138,468	40	0.25	0.07	0.01
UK2002	18,520,486	298,113,762	200	0.16	0.13	0.09
Human	4,674	86,282	44	0.05	0.03	0.01
WordNet	76,853	120,399	5	3.29	0.06	0.05
Patents	3,774,768	16,518,947	20	0.89	0.09	0.21

Compared Methods. We implemented all the exact methods using the processing engine of the state-of-the-art continuous subgraph enumeration framework, RapidFlow [53]. We built all the sampling methods on top of the state-of-the-art sampling framework, G-CARE [47]. The examined methods are the following:

- **JointEnum** is the joint enumeration method for CRQ.
- **LookAheadEnum** is the joint enumeration method with the look-ahead set enhancement for CRQ.
- **RWC** is the RW estimator using samples on the enumeration space of LookAheadEnum for CRQ.
- **RWC-H** is the hybrid method of combining LookAheadEnum and RWC for CRQ.
- **ProbeEnum** is the probe enumeration method for DRQ.
- **IEEnum** is the pruning method using the inclusion and exclusion principle for DRQ.
- **RWD** is the sampling method using uniform samples of Δe, followed by RW sampling for DRQ.
- **RWD-W** is the weighted sampling method based on the estimated upper bounds of Δe, followed by RW sampling for DRQ.

Evaluation Metrics. We set the parameter k to 5 by default. For the exact enumeration methods, we report the ratio of queries completed within 10 minutes, as well as the average running time for the completed cases. To evaluate the accuracy of our sampling strategies, we use *q-error*, a widely adopted metric in cardinality estimation [44]. Given a true value d and an estimate $\hat{d}$, the q-error is defined as $\max\left(\max(1,d)/\max(1,\hat{d}), \max(1,\hat{d})/\max(1,d)\right)$.

Implementation. We conduct all experiments on a Linux server with a 48-core CPU@3.45GHz and 256GB memory. We compiled our code using GCC 8.4.0 with -O3 optimization.

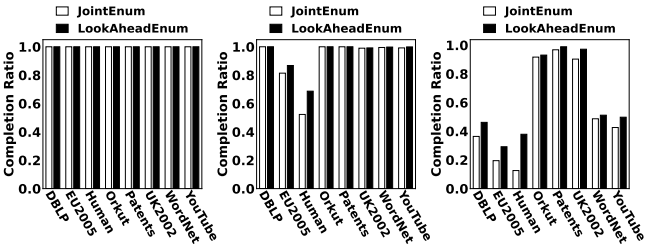
7.2 Empirical Study on CRQ

Efficiency. For the exact methods for CRQ, we note that the efficiency results are *not* affected by parameter k, because these approaches anyway compute the perturbation values $|\Delta\mathcal{M}(\Delta e)|$ for all valid perturbations Δe, enumerating all instances in $\mathcal{M}(Q^*)\setminus\mathcal{M}(Q)$. Figure 3 shows the ratio of CRQs that can be completed by the exact methods within a 10-minute time frame. As expected, larger query graphs result in a lower completion ratio, as the search space and the number of set intersection operations increase. JointEnum performs well on smaller queries, such as those with $|V_{Q^*}| = 4$ and 8. However, LookAheadEnum significantly outperforms JointEnum

Table 2: Sampling errors of CRQ. Median and max q-errors (in brackets).

Dataset	Query-4			Query-8			Query-16		
	RWC	RWC-H	Gain	RWC	RWC-H	Gain	RWC	RWC-H	Gain
DBLP	1.0 (1.3)	1.0 (1.1)	↓1× (1×)	1.3 (21.7)	1.1 (2.6)	↓1× (8×)	7.5 (274.9)	2.3 (42.3)	↓3× (6×)
Human	1.0 (61.3)	1.0 (2.3)	↓1× (26×)	1.2 (1.5)	1.1 (1.4)	↓1× (1×)	7.2 (20.6)	2.8 (6.5)	↓2× (3×)
EU2005	1.0 (6.7)	1.0 (2.5)	↓1× (2×)	8.3 (98.9)	1.5 (18.3)	↓5× (5×)	48.1 (414.3)	4.4 (43.8)	↓10× (9×)
UK2002	1.0 (1.2)	1.0 (1.1)	↓1× (1×)	1.5 (4.6)	1.1 (2.9)	↓1× (1×)	10.6 (731.2)	6.2 (57.9)	↓1× (12×)
Orkut	1.8 (207.1)	1.0 (1.2)	↓1× (173×)	37.6 (3 209.9)	4.1 (27.1)	↓9× (118×)	8.4 (660.3)	5.3 (48.2)	↓1× (13×)
Patents	1.0 (2.3)	1.0 (1.2)	↓1× (1×)	7.7 (216.8)	2.6 (25.7)	↓2× (8×)	17.9 (673.1)	3.7 (34.4)	↓4× (19×)
WordNet	1.0 (1.9)	1.0 (1.2)	↓1× (1×)	15.6 (646.5)	3.1 (23.4)	↓5× (27×)	159.1 (8 875.6)	9.2 (62.4)	↓17× (142×)
YouTube	1.0 (12.8)	1.0 (2.0)	↓1× (6×)	16.4 (785.3)	2.3 (60.8)	↓7× (12×)	79.3 (1 419.9)	12.5 (89.6)	↓6× (15×)

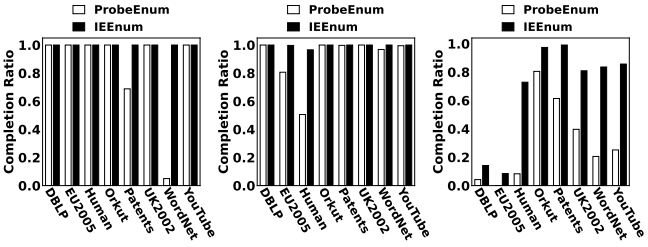
across all settings and datasets. In the particular case of 16-node queries on Human, LookAheadEnum completes *over three times* more queries on a 10-minute budget. This is attributed to LookAheadEnum's ability to prune many invalid partial matches using the look-ahead set, especially in dense query and data graphs, like the Human dataset.



(a) Query-4 (b) Query-8 (c) Query-16

Figure 3: Completion ratio for CRQ.

Error Analysis. We select those CRQs that LookAheadEnum can fully process in 10 minutes, and hence we know the exact TopkSub solutions for. To assess the accuracy of the sampling methods (RWC and RWC-H), we measure the q-error for the k-th best perturbation value, $|\Delta\mathcal{M}(\Delta e)|$, with k set to 5 by default. Since RWC-H involves an initial enumeration before sampling, we impose a 10-minute time limit and compare the q-errors produced by both methods within this time frame. The results are reported in Table 2. Overall, RWC-H consistently outperforms RWC. For smaller queries ($|V_{Q^*}| = 4$), both methods yield highly accurate estimates. However, as the query size grows to medium ($|V_{Q^*}| = 8$), RWC-H maintains its accuracy, while RWC's error increases substantially. When the number of query vertices reaches 16, RWC's performance degrades sharply, particularly in terms of maximum q-error, due to the larger sampling space. By comparison, RWC-H achieves 1-2 orders of magnitude lower approximation errors for the most challenging CRQs, due to its ability to enumerate the densest parts of the graph and increase the number of valid instance samples.



(a) Query-4 (b) Query-8 (c) Query-16

Figure 4: Completion ratio for DRQ.

7.3 Empirical Study on DRQ

Efficiency. Figure 4 shows the ratio of DRQs that can be completed by the exact methods within a 10-minute time frame. Regarding ProbeEnum, in contrast to the CRQ results, it completes only 5% of the DRQ queries with 4 query vertices in the WordNet dataset within the time limit. Upon investigation, this poor performance is due to the two specific cases for DRQ with 4 query vertices: (1) $|V_{Q_h}| = 3$ and $|V_{Q_t}| = 1$, and (2) $|V_{Q_h}| = |V_{Q_t}| = 2$. The perturbation space for both cases is extremely large in the WordNet dataset because of the limited label constraints (only 5 labels in the data graph). Therefore, ProbeEnum needs to probe a prohibitively large number of perturbations to compute the top-k. As the number of query vertices increases to 16, ProbeEnum's performance deteriorates further, especially on the EU2005 dataset. On the other hand, IEEnum significantly outperforms ProbeEnum in completion ratio.

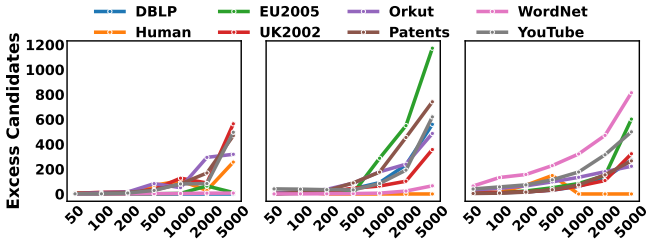


Figure 5: Number of excess candidates with varying k.

Effectiveness of the Inclusion-Exclusion Bounds. To further evaluate the pruning effectiveness of IEEnum, in Figure 5 we report the number of *excess candidates* by varying k from 50 to 5000. For a given parameter k, the excess candidates are those candidate perturbations that lie outside the true top-k set but cannot be pruned by the bounds, thus requiring enumeration. For the smaller k values (e.g., $k \leq 200$), the number of excess candidates is minuscule and remains stable. Although the increase is more pronounced for $k > 200$, it remains small with respect to the value of k itself, even for extreme cases, like $k = 5000$. This demonstrates the effectiveness of our bounds in filtering out invalid candidate perturbations, ensuring that only a minimal number of excess candidates require enumeration. We also provide the robustness study of IEEnum in the Appendix A.2.

Error Analysis. We select the DRQs that IEEnum can fully process in 10 minutes, and thus we know the exact solutions for. To assess the accuracy of the sampling methods (RWD and RWD-W), we measure the q-error for the k-th best perturbation value, $|\Delta\mathcal{M}(\Delta e)|$, with k set to 5 by default. Since RWD-W requires initial sampling for upper bound estimations, we impose a 10-minute time limit and compare the q-errors produced by both methods within this time frame. The results are reported in Table 3.

Table 3: Sampling errors of DRQ. Median and max q-errors (in brackets).

Dataset	Query-4			Query-8			Query-16		
	RWD	RWD-W	Gain	RWD	RWD-W	Gain	RWD	RWD-W	Gain
DBLP	2.4 (3.0)	1.0 (1.0)	↓**2× (3×)**	53.1 (69.2)	1.1 (1.3)	↓**49× (55×)**	261.2 (573.4)	1.6 (2.7)	↓**164× (214×)**
Human	76.8 (108.1)	1.2 (2.9)	↓**63× (37×)**	42.8 (319.5)	1.0 (5.0)	↓**42× (64×)**	60.5 (648.3)	1.5 (5.7)	↓**40× (114×)**
EU2005	38.2 (45.3)	1.1 (1.3)	↓**35× (35×)**	159.5 (778.1)	1.6 (3.9)	↓**102× (202×)**	107.7 (879.6)	4.3 (23.5)	↓**25× (38×)**
UK2002	3.2 (37.3)	1.0 (1.1)	↓**3× (33×)**	78.4 (144.8)	1.2 (1.6)	↓**67× (88×)**	88.3 (988.2)	8.8 (127.3)	↓**10× (8×)**
Orkut	3.2 (51.0)	1.1 (1.2)	↓**3× (42×)**	3.3 (95.5)	1.1 (1.2)	↓**3× (80×)**	203.3 (7029.1)	4.2 (593.7)	↓**48× (12×)**
Patents	31.2 (63.8)	1.0 (1.1)	↓**31× (57×)**	3.5 (180.7)	1.3 (2.0)	↓**4× (89×)**	13 206 (89 674)	21.5 (98.4)	↓**572× (911×)**
WordNet	42.9 (132.5)	1.0 (1.1)	↓**42× (123×)**	348.7 (709.2)	1.5 (3.1)	↓**237× (226×)**	108.7 (672 547)	2.5 (7.0)	↓**44× (96 348×)**
YouTube	28.7 (41.2)	1.2 (1.5)	↓**25× (28×)**	30.9 (63.5)	1.3 (1.9)	↓**23× (33×)**	126.6 (7 128.0)	6.1 (140.9)	↓**21× (51×)**

7.4 Application Study

In this section, we present an actual use case of TopkSub. Specifically, our perturbation analysis offers a novel stress-test for the robustness of GNN-based subgraph counting methods. As a concrete case, we use a state-of-the-art GNN method for subgraph counting, termed LSS [71]. LSS employs query decomposition and encodes the substructures using GNNs. It then applies active learning to aggregate the count estimates of the substructures, improving the accuracy of predicting the overall count for a given query. We demonstrate that when we perform the perturbations (edge additions) identified by our framework and train/apply LSS on the perturbed graph, the accuracy of LSS' estimates declines dramatically. Meanwhile, we also test LSS under alternative types of perturbation (specifically edge addition) from the literature, and show that none of them exposes the weaknesses TopkSub uncovers. Overall, our study suggests that TopkSub provides a distinct and particularly tough adversarial test for the robustness of GNN subgraph counting methods (and potentially of GNN methods for different problems).

Attack Model. The attack model is illustrated in Figure 6. The left part shows the standard application of LSS, comprising its training on the original data graph G (using its default hyper-parameter setting) and the recording of q-error in its estimates for a given set of queries. The right part applies the exact same process for LSS (including identical data split for training, validation, testing) but uses the perturbed version of G. By comparing the q-error in the original and in the perturbed case we may assess the robustness of LSS to the perturbation. Clearly, the effectiveness of the process depends on the chosen perturbation approach.

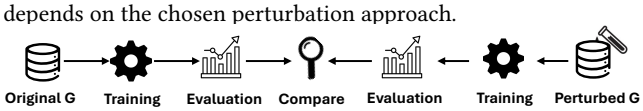


Figure 6: Workflow of robustness analysis for LSS.

Perturbation Approaches. We give each approach a budget of k edge additions to G. *Random* adds k random edges that are not already in E_G. *Gell* [54] adds the k edges that maximize the leading eigenvalue of G's adjacency matrix. *Nettack* [73] is a perturbation strategy for adversarial attacks in GNNs. *Nettack* requires the specification of target vertices; we randomly choose k nodes from V_G and apply the top suggested perturbation for each. For TopkSub, we compute the top-k perturbations for each query edge, and from their union, we apply those with the k highest perturbation values.

Setup. We align with LSS's own evaluation in [71], i.e., we use the YouTube data graph, and try the same query sizes (i.e., 4, 8, 16, 24, 32). Query generation in the experiments of [71] is identical to ours.

We present LSS's q-error for the original G and for each of the 4 perturbed versions, under the $k = 10$ setting in Figure 7. We also provide the q-error under other two different k settings (20 and 50) in our technical report [1]. To distinguish overestimated from underestimated cases, we display overestimated queries above the reference value of 1 and underestimated queries below it. For small query sizes (i.e., 4 and 8), LSS's error is not significantly affected by any perturbation approach. This is because the GNNs for modeling the substructures in LSS are trained on the same data graph. As a result, for small queries that do not require decomposition, the GNNs can provide highly accurate estimates of the corresponding queries on the same perturbed data graph. For large queries, however, LSS is strongly affected by our TopkSub approach, even for $k = 10$ edge additions, exhibiting a huge underestimation; TopkSub is the only perturbation approach with such a strong effect. The reason behind TopkSub's effective attack is its very design to approximate kSub, i.e., although the top-k edges are chosen based on their individual perturbation values, they also have a direct *combined* effect in maximizing the overall subgraph count. On the other hand, LSS cannot cope with the TopkSub-induced combinatorial increase in subgraph counts for large queries, due to its aggregation of estimates for multiple, small substructures.

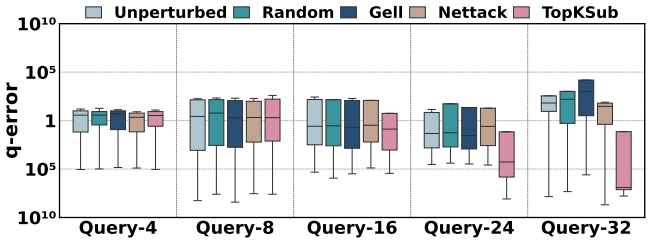


Figure 7: Error of LSS for $k = 10$ perturbations.

8 Conclusion

In this paper, we present a novel study on graph perturbation for subgraph counting. We show that the combinatorial problem of adding k edges as perturbations (kSub) is NP-hard to approximate within any constant factor. We therefore relax the problem to selecting the top-k edges (TopkSub) and develop pruning and sampling methods for both connected and disconnected relaxations, enabling perturbation analysis on million-scale graphs and revealing vulnerabilities in state-of-the-art GNN-based subgraph counters. Future work will explore other perturbation settings on graphs, such as attributed graph queries [60], GPU-based subgraph enumeration [22, 23] and cryptocurrency trading [40].

Acknowledgments

Kyriakos Mouratidis was supported by the Ministry of Education, Singapore, under its Academic Research Fund Tier 2 (Award No. MOE-T2EP20121-0002). Any opinions, findings and conclusions expressed in this material are those of the authors and do not reflect the views of the Ministry of Education, Singapore.

References

[1] 2023. The technical report. https://doi.org/10.5281/zenodo.18060323

[2] Yanif Ahmad, Oliver Kennedy, Christoph Koch, and Milos Nikolic. 2012. DBToaster: higher-order delta processing for dynamic, frequently fresh views. *PVLDB* 5, 10 (2012), 968–979.

[3] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi, and Nick Duffield. 2015. Efficient Graphlet Counting for Large Networks. In *ICDM*. 1–10.

[4] Erik Altman, Jovan Blanuša, Luc von Niederhäusern, Béni Egressy, Andreea Anghel, and Kubilay Atasu. 2023. Realistic synthetic financial transactions for anti-money laundering models. In *NeurIPS*. 29851 – 29874.

[5] Junya Arai, Yasuhiro Fujiwara, and Makoto Onizuka. 2023. GuP: Fast Subgraph Matching by Guard-based Pruning. *PACMMOD* 1, 2, Article 167 (2023), 26 pages.

[6] Francesco Belardo, Maurizio Brunetti, and Adriana Ciampella. 2019. Edge perturbation on signed graphs with clusters: Adjacency and Laplacian eigenvalues. *Discrete Applied Mathematics* 269 (2019), 130–138.

[7] Katja Berdica. 2002. An introduction to road vulnerability: What has been done, is done and should be done. *Transport Policy* 9 (04 2002), 117–127.

[8] Alina Beygelzimer, Geoffrey Grinstein, Ralph Linsker, and Irina Rish. 2005. Improving network robustness by edge modification. *Physica A: Statistical Mechanics and its Applications* 357 (11 2005), 593–612.

[9] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient Subgraph Matching by Postponing Cartesian Products. In *SIGMOD*. 1199–1214.

[10] Domingos M. Cardoso and Oscar Rojo. 2017. Edge perturbation on graphs with clusters: Adjacency, Laplacian and signless Laplacian eigenvalues. *Linear Algebra Appl.* 512 (2017), 113–128.

[11] Deepayan Chakrabarti, Yang Wang, Chenxi Wang, Jurij Leskovec, and Christos Faloutsos. 2008. Epidemic thresholds in real networks. *ACM Trans. Inf. Syst. Secur.* 10, 4 (jan 2008), 26 pages.

[12] Hau Chan and Leman Akoglu. 2016. Optimizing network robustness by edge rewiring: a general framework. *Data Min. Knowl. Discov.* 30, 5 (2016), 1395–1425.

[13] Chen Chen, Jingrui He, Nadya Bliss, and Hanghang Tong. 2015. On the Connectivity of Multi-layered Networks: Models, Measures and Optimal Control. In *ICDM*. 715–720.

[14] Chen Chen, Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2016. Eigen-Optimization on Large Graphs by Edge Manipulation. *ACM Trans. Knowl. Discov. Data* 10, 4 (jun 2016), 30 pages.

[15] W. Ellens and R. E. Kooij. 2013. Graph measures and network robustness. arXiv:1311.5064 [cs.DM] https://arxiv.org/abs/1311.5064

[16] Wenfei Fan and Chao Tian. 2022. Incremental Graph Computations: Doable and Undoable. *ACM Trans. Database Syst.* 47, 2 (2022), 44 pages.

[17] Wenfei Fan, Chao Tian, Ruiqi Xu, Qiang Yin, Wenyuan Yu, and Jingren Zhou. 2021. Incrementalizing Graph Algorithms. In *SIGMOD*. 459–471.

[18] Wenfei Fan, Xin Wang, and Yinghui Wu. 2013. Incremental graph pattern matching. *ACM Trans. Database Syst.* 38, 3, Article 18 (2013), 47 pages.

[19] Scott Freitas, Diyi Yang, Srijan Kumar, Hanghang Tong, and Duen Horng Chau. 2023. Graph Vulnerability and Robustness: A Survey. *TKDE* 35, 6 (2023), 5915–5934.

[20] Zengan Gao and Mao Ye. 2007. A framework for data mining-based anti-money laundering research. *Journal of Money Laundering Control* 10, 2 (2007), 170–179.

[21] Michael R. Garey and David S. Johnson. 1990. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

[22] Wentian Guo, Yuchen Li, Mo Sha, Bingsheng He, Xiaokui Xiao, and Kian-Lee Tan. 2020. Gpu-accelerated subgraph enumeration on partitioned graphs. In *SIGMOD*. 1067–1082.

[23] Wentian Guo, Yuchen Li, and Kian-Lee Tan. 2020. Exploiting reuse for gpu subgraph enumeration. *TKDE* 34, 9 (2020), 4231–4244.

[24] William L. Hamilton, Rex Ying, and Jure Leskovec. 2017. Inductive representation learning on large graphs. In *NIPS*. 1025–1035.

[25] 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 *SIGMOD*. 1429–1446.

[26] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, Zhengping Qian, Jingren Zhou, Jiangneng Li, and Bin Cui. 2021. Cardinality estimation in DBMS: a comprehensive benchmark evaluation. *PVLDB* 15, 4 (2021), 752–765.

[27] Tomaž Hočevar and Janez Demšar. 2014. A combinatorial approach to graphlet counting. *Bioinformatics* 30, 4 (2014), 559–565.

[28] Wei Jin, Yaxing Li, Han Xu, Yiqi Wang, Shuiwang Ji, Charu Aggarwal, and Jiliang Tang. 2021. Adversarial attacks and defenses on graphs. *ACM SIGKDD Explorations Newsletter* 22, 2 (2021), 19–34.

[29] Charilaos Kanatsoulis and Alejandro Ribeiro. 2024. Counting Graph Substructures with Graph Neural Networks. In *ICLR*.

[30] Zahra Kashani, Hayedeh Ahrabian, Elahe Elahi, Abbas Nowzari, Elnaz Saberi Ansari, Sahar Asadi, Shahin Mohammadi, and Falk Schreiber. 2009. Kavosh: A new algorithm for finding network motifs. *BMC bioinformatics* 10 (2009), 318.

[31] Elias Boutros Khalil, Bistra Dilkina, and Le Song. 2014. Scalable diffusion-aware optimization of network topology. In *KDD*. 1226–1235.

[32] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong, and Wook-Shin Han. 2021. Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching. In *SIGMOD*. 925–937.

[33] Kyoungmin Kim, Hyeonji Kim, George Fletcher, and Wook-Shin Han. 2021. Combining Sampling and Synopses with Worst-Case Optimal Runtime and Quality Guarantees for Graph Pattern Cardinality Estimation. In *SIGMOD*. 964–976.

[34] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[35] Gunnar W Klau and René Weiskircher. 2005. Robustness and resilience. In *Network analysis: Methodological foundations*. 417–437.

[36] Christoph Koch. 2010. Incremental query evaluation in a ring of databases. In *PODS*. 87–98.

[37] Hai Lan, Zhifeng Bao, and Yuwei Peng. 2021. A survey on advancing the dbms query optimizer: Cardinality estimation, cost model, and plan enumeration. *Data Science and Engineering* 6 (2021), 86–101.

[38] Yukyoung Lee, Kyoungmin Kim, Wonseok Lee, and Wook-Shin Han. 2024. In-depth Analysis of Continuous Subgraph Matching in a Common Delta Query Compilation Framework. *PACMMOD* 2, 3 (2024), 27 pages.

[39] Feifei Li, Bin Wu, Ke Yi, and Zhuoyue Zhao. 2016. Wander Join: Online Aggregation via Random Walks. In *SIGMOD*. 615–629.

[40] Bingqiao Luo, Zhen Zhang, Qian Wang, and Bingsheng He. 2024. Multi-chain graphs of graphs: a new approach to analyzing blockchain datasets. In *NeurIPS*. 28490–28514.

[41] D. Marcus and Y. Shavitt. 2012. RAGE – A rapid graphlet enumerator for large networks. *Computer Networks* 56, 2 (2012), 810–819.

[42] Ine Melckenbeeck, Pieter Audenaert, Didier Colle, and Mario Pickavet. 2018. Efficiently counting all orbits of graphlets of any order in a graph using autogenerated equations. *Bioinformatics* 34, 8 (2018), 1372–1380.

[43] Amine Mhedhbi and Semih Salihoglu. 2019. Optimizing subgraph queries by combining binary and worst-case optimal joins. *PVLDB* 12, 11 (2019), 1692–1704.

[44] Guido Moerkotte, Thomas Neumann, and Gabriele Steidl. 2009. Preventing bad plans by bounding the impact of cardinality estimation errors. *PVLDB* 2, 1 (Aug. 2009), 982–993.

[45] Mark Ortmann and Ulrik Brandes. 2017. Efficient orbit-aware triad and quad census in directed and undirected graphs. *Applied network science* 2, 1 (2017), 13.

[46] Giuliano Andrea Pagani and Marco Aiello. 2013. The Power Grid as a complex network: A survey. *Physica A: Statistical Mechanics and its Applications* 392, 11 (06 2013), 2688–2700.

[47] Yeonsu Park, Seongyun Ko, Sourav S Bhowmick, Kyoungmin Kim, Kijae Hong, and Wook-Shin Han. 2020. G-CARE: A framework for performance benchmarking of cardinality estimation techniques for subgraph matching. In *SIGMOD*. 1099–1114.

[48] Ali Pinar, C. Seshadhri, and Vaidyanathan Vishal. 2017. ESCAPE: Efficiently Counting All 5-Vertex Subgraphs. In *WWW*. 1431–1440.

[49] Wonseok Shin, Siwoo Song, Kunsoo Park, and Wook-Shin Han. 2024. Cardinality Estimation of Subgraph Matching: A Filtering-Sampling Approach. *PVLDB* 17, 7 (2024), 1697–1709.

[50] Michele Starnini, Charalampos E. Tsourakakis, Maryam Zamanipour, André Panisson, Walter Allasia, Marco Fornasiero, Laura Li Puma, Valeria Ricci, Silvia Ronchiadin, Angela Ugrinoska, Marco Varetto, and Dario Moncalvo. 2021. Smurf-Based Anti-money Laundering in Time-Evolving Transaction Networks. In *ECML-PKDD*. 171–186.

[51] Lichao Sun, Yingtong Dou, Carl Yang, Kai Zhang, Ji Wang, S Yu Philip, Lifang He, and Bo Li. 2022. Adversarial attack and defense on graph data: A survey. *TKDE* 35, 8 (2022), 7693–7711.

[52] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. RapidMatch: a holistic approach to subgraph query processing. *PVLDB* 14, 2 (2020), 176–188.

[53] Shixuan Sun, Xibo Sun, Bingsheng He, and Qiong Luo. 2022. RapidFlow: an efficient approach to continuous subgraph matching. *PVLDB* 15, 11 (2022), 2415–2427.

[54] Hanghang Tong, B. Aditya Prakash, Tina Eliassi-Rad, Michalis Faloutsos, and Christos Faloutsos. 2012. Gelling, and melting, large graphs by edge manipulation. In *CIKM*. 245–254.

[55] Vicenç Torra and Julián Salas. 2019. Graph Perturbation as Noise Graph Addition: A New Perspective for Graph Anonymization. In *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. 121–137.

[56] Stojan Trajanovski, Javier Martín-Hernández, Wynand Winterbach, and Piet Van Mieghem. 2013. Robustness envelopes of networks. *Journal of Complex Networks* 1, 1 (2013), 44–62.

[57] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *ICLR*.

[58] Alessandro Vespignani. 2010. Complex networks: The fragility of interdependency. *Nature* 464 (2010), 984–985.

[59] Hanchen Wang, Rong Hu, Ying Zhang, Lu Qin, Wei Wang, and Wenjie Zhang. 2022. Neural Subgraph Counting with Wasserstein Estimator. In *SIGMOD*. 160–175.

[60] Yanhao Wang, Yuchen Li, Ju Fan, Chang Ye, and Mingke Chai. 2021. A survey of typical attributed graph queries. *WWWJ* 24, 1 (2021), 297–346.

[61] Yexin Wang, Zhi Yang, Junqi Liu, Wentao Zhang, and Bin Cui. 2023. Scapin: Scalable Graph Structure Perturbation by Augmented Influence Maximization. *PACMMOD* 1, 2, Article 146 (2023), 21 pages.

[62] Sebastian Wernicke. 2005. A Faster Algorithm for Detecting Network Motifs. In *Algorithms in Bioinformatics (LNCS 3692)*. Springer, 165–177.

[63] Wenwen Xia, Yuchen Li, and Shenghong Li. 2022. On the substructure countability of graph neural networks. *TKDE* 35, 11 (2022), 11681–11692.

[64] Hanhua Xiao, Yuchen Li, Yanhao Wang, Panagiotis Karras, Kyriakos Mouratidis, and Natalia Rozalia Avlona. 2024. How to Avoid Jumping to Conclusions: Measuring the Robustness of Outstanding Facts in Knowledge Graphs. In *KDD*. 3539–3550.

[65] Kaidi Xu, Hongge Chen, Sijia Liu, Pin-Yu Chen, Tsui-Wei Weng, Mingyi Hong, and Xue Lin. 2019. Topology attack and defense for graph neural networks: an optimization perspective. In *IJCAI*. 3961–3967.

[66] Zuoyu Yan, Junru Zhou, Liangcai Gao, Zhi Tang, and Muhan Zhang. 2024. An Efficient Subgraph GNN with Provable Substructure Counting Power. In *KDD*. 3702–3713.

[67] Chang Ye, Yuchen Li, Shixuan Sun, and Wentian Guo. 2024. gSWORD: GPU-accelerated Sampling for Subgraph Counting. *PACMMOD* 2, 1, Article 33 (2024), 26 pages.

[68] Rose Yu, Huida Qiu, Zhen Wen, ChingYung Lin, and Yan Liu. 2016. A Survey on Social Media Anomaly Detection. *SIGKDD Explor. Newsl.* 18, 1 (2016), 1–14.

[69] Xingtong Yu, Zemin Liu, Yuan Fang, and Xinming Zhang. 2023. Learning to count isomorphisms with graph neural networks. In *AAAI*, Vol. 37. 4845–4853.

[70] Zhijie Zhang, Yujie Lu, Weiguo Zheng, and Xuemin Lin. 2024. A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction. *PACMMOD* 2, 1, Article 60 (2024), 29 pages.

[71] Kangfei Zhao, Jeffrey Xu Yu, Qiyan Li, Hao Zhang, and Yu Rong. 2023. Learned sketch for subgraph counting: a holistic approach. *VLDB J.* 32, 5 (2023), 937–962.

[72] David Zuckerman. 2006. Linear degree extractors and the inapproximability of max clique and chromatic number. In *STOC*. 681–690.

[73] Daniel Zügner, Amir Akbarnejad, and Stephan Günnemann. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *KDD*. 2847–2856.

[74] Daniel Zügner, Oliver Borchert, Amir Akbarnejad, and Stephan Günnemann. 2020. Adversarial Attacks on Graph Neural Networks: Perturbations and their Patterns. *TKDD* 14, 5, Article 57 (2020), 31 pages.

A Appendix

A.1 Examples for DRQ and CRQ

We provide examples to demonstrate: (1) JointEnum and LookAheadEnum for CRQ and (2) ProbeEnum and IEEnum for DRQ.

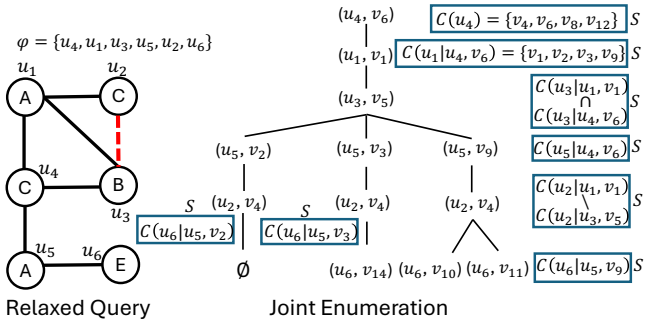


Figure 8: Example of JointEnum.

Example A.1. Assuming the query and data graph in Figure 1, and given the query edge $(u_h, u_t) = (u_3, u_2)$, removing (u_h, u_t) from Q results in a connected relaxed query Q^*. The candidate graph $C(Q^*, G)$ is the same as $C(Q, G)$ in Figure 1. The matching order φ is $(u_4, u_1, u_3, u_5, u_2, u_6)$. Starting from the first query vertex u_4 with $S = C(u_4) = \{v_4, v_6, v_8, v_{12}\}$, Figure 8 shows a part of the DFS tree, rooted at $M = \{(u_4, v_6)\}$. The next query vertex is u_1 and the corresponding set S is $C(u_1|u_4, v_6) = \{v_1, v_2, v_3, v_9\}$. In DFS fashion, we pick v_1 from S and update M to $\{(u_4, v_6), (u_1, v_1)\}$. Set S for u_3 is $C(u_3|u_1, v_1) \cap C(u_3|u_4, v_6) = \{v_5\}$, since $N^{\varphi}_{<}(u_3) = \{u_1, u_4\}$. We thus append (u_3, v_5) to M. Next, for u_5 we obtain $S = \{v_2, v_3, v_9\}$ (note that v_1 is excluded from S because it has already been matched to u_1) and extend M to (u_5, v_2). Then, we compute S for the tail vertex u_2 according to Line 7 in Algorithm 1, i.e., $S = C(u_2|u_1, v_1) \setminus C(u_2|u_3, v_5) = \{v_4\}$. Next, for u_6 we get $S = C(u_6|u_5, v_2) = \emptyset$ so this partial instance (i.e., $M = \{(u_4, v_6), (u_1, v_1), (u_3, v_5), (u_5, v_2), (u_2, v_4)\}$) cannot be extended to full. Therefore, we backtrack to u_5 because there are two unexplored candidates (u_5, v_3) and (u_5, v_9) which branch out to new partial instances. (u_5, v_3) eventually leads to a full instance, and (u_5, v_9) to two, for $\Delta e = (v_5, v_4)$ in all three cases.

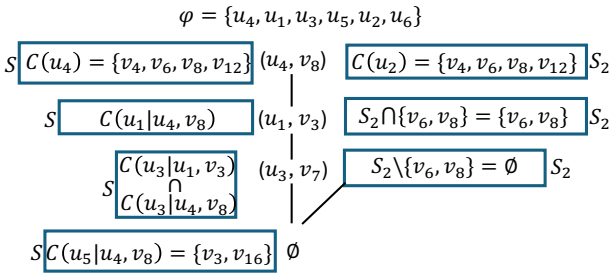


Figure 9: Example of LookAheadEnum.

Example A.2. We demonstrate LookAheadEnum using the same setup as Example A.1. Figure 9 presents part of the DFS tree, starting from (u_4, v_8). Initially, the look-ahead set of the tail vertex u_2 is $S_2 = C(u_2) = \{v_4, v_6, v_8, v_{12}\}$. Since S_2 is non-empty, we extend M by appending (u_1, v_3) to it. The addition of (u_1, v_3) updates the look-ahead set to $S_2 = S_2 \cap C(u_2|u_1, v_3) = \{v_6, v_8\}$, because u_1 is a backward neighbor of the tail vertex u_2. Since $S_2 \neq \emptyset$, and $S = C(u_3|u_1, v_3) \cap C(u_3|u_4, v_8) = \{v_7\}$, we expand M by (u_3, v_7). After the addition of (u_3, v_7), and since u_3 is the head vertex, S_2 is updated to $S_2 \setminus C(u_2|u_3, v_7) = \emptyset$, which indicates that the current partial instance $M = \{(u_4, v_8), (u_1, v_3), (u_3, v_7)\}$ cannot lead to a full instance. Therefore, M can be safely dismissed, despite the fact that the candidate set S for u_5 (i.e., $C(u_5|u_4, v_8)$) is non-empty.

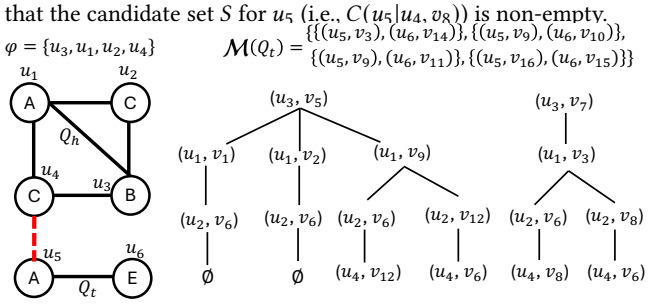
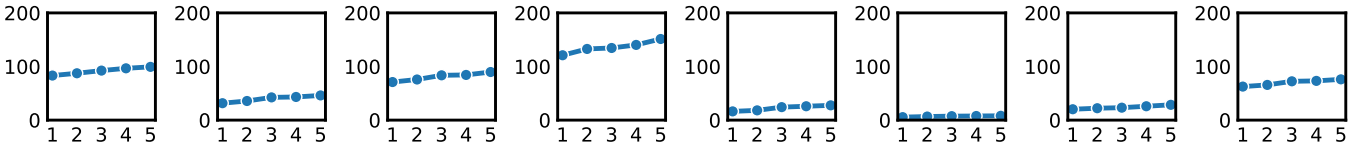


Figure 10: Example of ProbeEnum.

Example A.3. We exemplify ProbeEnum in Figure 10, for the illustrated components Q_t and Q_h of the relaxed query. We first find all instances of Q_t in the data graph, i.e., set $\mathcal{M}(Q_t)$ shown at the top of the figure. We then enumerate all the instances of Q_h in the data graph to probe $\mathcal{M}(Q_t)$ with. The matching order (for Q_h in G) is $\varphi = (u_3, u_1, u_2, u_4)$. The candidates for the first query vertex u_3 are $C(u_3) = \{v_5, v_7\}$; we show the DFS trees for either instantiation. Among the 4 instances of Q_h, consider $M_h = \{(u_3, v_5), (u_1, v_9), (u_2, v_6), (u_4, v_{12})\}$. When probing $\mathcal{M}(Q_t)$ with M_h we (I) ignore instances $M_t \in \mathcal{M}(Q_t)$ that share vertices with M_h (i.e., $\{(u_5, v_9), (u_6, v_{10})\}$ and $\{(u_5, v_9), (u_6, v_{11})\}$ due

Table 4: Sampling errors of CRQ Query-16 with varying k. The median and max q-errors (in brackets) are reported.

	k=1			k=2			k=3			k=4		
Dataset	**RWC**	**RWC-H**	**Gain**	**RWC**	**RWC-H**	**Gain**	**RWC**	**RWC-H**	**Gain**	**RWC**	**RWC-H**	**Gain**
DBLP	36.8 (165.8)	6.2 (24.4)	**↓5x (6x)**	40.7 (162.9)	7.9 (21.5)	**↓5x (7x)**	50.6 (192.6)	8.5 (26.4)	**↓5x (7x)**	39.5 (188.7)	7.8 (23.9)	**↓5x (7x)**
Human	7.9 (17.1)	2.4 (3.5)	**↓3x (4x)**	8.8 (18.2)	2.8 (3.7)	**↓3x (4x)**	9.2 (18.5)	3.2 (4.2)	**↓2x (4x)**	8.6 (20.9)	3.0 (4.6)	**↓2x (4x)**
EU2005	5.1 (337.6)	2.4 (23.9)	**↓2x (14x)**	6.9 (254.2)	2.2 (28.2)	**↓3x (9x)**	8.6 (265.2)	3.7 (26.6)	**↓2x (9x)**	11.3 (262.4)	3.9 (37.3)	**↓2x (7x)**
UK2002	18.8 (208.6)	5.5 (35.7)	**↓3x (5x)**	28.1 (241.4)	4.5 (29.2)	**↓6x (8x)**	31.8 (252.1)	5.3 (44.3)	**↓6x (5x)**	38.5 (296.0)	5.9 (36.8)	**↓6x (8x)**
Orkut	6.6 (875.9)	2.3 (54.9)	**↓2x (15x)**	7.8 (716.6)	3.1 (43.1)	**↓2x (16x)**	5.3 (711.7)	3.7 (38.8)	**↓1x (18x)**	9.7 (682.9)	4.2 (32.1)	**↓2x (21x)**
Patents	5.3 (665.8)	2.9 (27.6)	**↓1x (24x)**	5.9 (632.4)	3.5 (31.5)	**↓1x (20x)**	8.8 (612.5)	4.6 (47.8)	**↓1x (12x)**	14.5 (666.9)	3.3 (28.6)	**↓4x (23x)**
WordNet	132.2 (1751.6)	3.4 (43.2)	**↓38x (40x)**	130.6 (1478.0)	4.0 (49.3)	**↓32x (29x)**	131.1 (3996.5)	5.4 (83.1)	**↓24x (48x)**	135.7 (7614.0)	6.5 (54.3)	**↓20x (140x)**
YouTube	11.7 (336.2)	3.3 (41.8)	**↓3x (8x)**	18.4 (603.1)	4.1 (53.3)	**↓4x (11x)**	28.5 (681.8)	6.0 (76.8)	**↓4x (8x)**	51.3 (1119.7)	5.5 (85.8)	**↓9x (13x)**



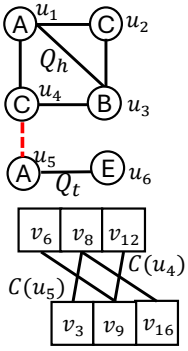
(a) DBLP (b) Human (c) EU2005 (d) UK2002 (e) Orkut (f) Patents (g) WordNet (h) YouTube

Figure 11: Average running time (in seconds) for IEEnum for DRQ Query-16 with varying k.

Table 5: Sampling errors of DRQ Query-16 with varying k. The median and max q-errors (in brackets) are reported.

	k=1			k=2			k=3			k=4		
Dataset	**RWD**	**RWD-W**	**Gain**	**RWD**	**RWD-W**	**Gain**	**RWD**	**RWD-W**	**Gain**	**RWD**	**RWD-W**	**Gain**
DBLP	250.8 (545.0)	1.4 (3.3)	**↓179x (165x)**	254.6 (545.4)	2.0 (4.6)	**↓127x (118x)**	256.5 (549.4)	2.7 (5.1)	**↓95x (107x)**	256.7 (567.7)	1.9 (4.8)	**↓135x (118x)**
Human	60.4 (635.2)	8.7 (14.0)	**↓6x (45x)**	60.3 (646.3)	3.6 (5.7)	**↓16x (113x)**	60.4 (643.9)	3.3 (5.7)	**↓18x (112x)**	60.4 (644.8)	3.1 (5.9)	**↓19x (109x)**
EU2005	87.8 (696.5)	3.4 (9.2)	**↓25x (75x)**	93.1 (879.6)	4.8 (10.6)	**↓19x (82x)**	101.7 (879.6)	5.1 (17.3)	**↓19x (50x)**	103.2 (879.6)	4.4 (19.1)	**↓23x (46x)**
UK2002	84.7 (904.7)	5.5 (108.0)	**↓15x (8x)**	89.9 (923.7)	6.8 (107.7)	**↓13x (8x)**	93.7 (932.2)	8.9 (107.6)	**↓10x (8x)**	91.8 (980.4)	9.3 (116.9)	**↓9x (8x)**
Orkut	119.5 (8583.3)	4.7 (347.4)	**↓25x (24x)**	149.0 (6545.5)	3.1 (319.7)	**↓48x (20x)**	2.8 (8660.4)	1.1 (334.4)	**↓2x (25x)**	195.5(6767.8)	2.7 (468.5)	**↓72x (14x)**
Patents	9488 (71165)	14.2 (44.3)	**↓668x (1606x)**	11016 (70490)	15.5 (41.1)	**↓710x (1715x)**	11275 (82478)	20.1 (99.3)	**↓560x (830x)**	12026 (74230)	19.5 (98.1)	**↓616x (756x)**
WordNet	15.9 (3622)	2.7 (4.8)	**↓5x (754x)**	36.1 (45318)	4.3 (6.8)	**↓8x (6664x)**	58.0 (81304)	3.9 (8.0)	**↓14x (10163x)**	80.2 (193018)	3.4 (6.6)	**↓23x (29245x)**
YouTube	83.3 (7634)	9.3 (215.3)	**↓8x (35x)**	113.7 (6951.0)	7.1 (189.6)	**↓16x (36x)**	127 (9078)	6.9 (77.5)	**↓18x (117x)**	116 (7037)	6.2 (107.4)	**↓18x (65x)**

to v_9's presence in M_h) and (II) use each of the remaining instances $M_t \in \mathcal{M}(Q_t)$ to derive an instance $M_h \cup M_t$ of Q^*. For the specific M_h, this produces two instances of Q^* that increment by one the perturbation values of edges (v_{12}, v_3) and (v_{12}, v_{16}), respectively.



v_h	$\lvert M(Q_h\lvert v_h)\rvert$
v_6	2
v_8	1
v_{12}	1

v_t	$\lvert M(Q_t\lvert v_t)\rvert$
v_9	2
v_3	1
v_{16}	1

v_h	v'	$\lvert M(Q_h\lvert v_h, v')\rvert$
v_6	v_3	1
v_6	v_9	1
v_8	v_3	1
v_{12}	v_9	1

v_t	v'	$\lvert M(Q_t\lvert v_t, v')\rvert$
v_9	v_9	2
v_3	v_3	1
v_{16}	v_{16}	1

Figure 12: Example of IEEnum.

Example A.4. We demonstrate IEEnum in Figure 12, where $Q^* = Q \setminus (u_4, u_5)$ and $k = 1$. We first enumerate Q^*'s two components Q_h and Q_t in G, and build hash tables for each of $|\mathcal{M}(Q_h|v_h)|$, $|\mathcal{M}(Q_t|v_t)|$, $|\mathcal{M}(Q_h|v_h, v')|$ and $|\mathcal{M}(Q_t|v_t, v')|$, where $v_h \in C(u_4)$, $v_t \in C(u_5)$ and $v' \in V_G$. Note that since A is the only vertex label that Q_h and Q_t have in common, we need only consider vertices v' where $L_G(v') = A$. Another illustrated insight is that $C(u_5)$ plays the role of $C(u_t)$ but is also the set of vertices v' where $L_G(v') = A$, thus the coincidence of two columns in the hash table for $|\mathcal{M}(Q_t|v_t, v')|$. With hash tables built, we sort $C(u_h)$ and $C(u_t)$ by $|\mathcal{M}(Q_h|v_h)|$ and $|\mathcal{M}(Q_t|v_t)|$; in the interest of space, we illustrate the sorted orders within the two hash tables at the top. Our nested loop traversal first considers (v_6, v_9) which is an invalid perturbation (since (v_6, v_9) is already in E_G), thus moving on to (v_6, v_3). Edge (v_6, v_3) is marked as a candidate perturbation and $LB(v_6, v_3) = 2 - 1 = 1$ is recorded as the top-1 lower bound so far. The next iteration considers (v_6, v_{16}) with $UB(v_6, v_{16}) = LB(v_6, v_{16}) = 2$, becoming our second candidate perturbation and updating the top-1 lower bound to 2. For the next pair, (v_8, v_9), the outer loop breaks directly because $UB(v_8, v_9) = 2$ does not exceed the current top-1 lower bound. Upon exiting from the nested loop, we evaluate (i.e., compute $|\Delta\mathcal{M}(\Delta e)|$ for) candidates in decreasing order of upper bound, with the lower bound as tie-breaker. I.e., we evaluate (v_6, v_{16}) before (v_6, v_3), and because its perturbation value is 2, we report it and terminate directly, without evaluating (v_6, v_3), since $UB(v_6, v_3) = 2$ is no greater than $|\Delta\mathcal{M}(\Delta e)|$ for $\Delta e = (v_6, v_{16})$.

A.2 Supplemental Materials for Experiments

Robustness Study for IEEnum. The pruning efficiency of IEEnum is impacted by the k highest lower bounds. To show the robustness of IEEnum, we present the running time of IEEnum for different k values in Figure 11. As k increases, IEEnum evaluates more candidate perturbations, resulting in a longer running time. However, this increase remains minimal due to the strong pruning rendered by the inclusion-exclusion bounds.

Error Analysis for 16-node Queries. We report the q-error for 16-node CRQ queries across different datasets and various k values in Table 4. The results are consistent with the previous experiment, with RWC-H remaining widely superior to RWC. For some cases (e.g., the WordNet dataset), the advantage of RWC-H over RWC becomes more pronounced for smaller k values. This is because smaller k values lead to more delta instances $|\Delta\mathcal{M}|$ associated with the top-k perturbations, which makes it easier for RWC-H to capture these changes through its hybrid approach of enumeration and sampling. Additionally, we report the q-error and DRQ queries across different datasets and various k values in Table 5 respectively. We find that RWD-W maintains consistent gains over RWD.