Towards Plug-and-Play Visual Graph Query Interfaces: Data-driven Selection of Canned Patterns for Large Networks

Zifeng Yuan
Fudan University & NTU
zfyuan16@fudan.edu.cn

Huey Eng Chua
Nanyang Technological University
hechua@ntu.edu.sg

Sourav S Bhowmick
Nanyang Technological University
assourav@ntu.edu.sg

Zekun Ye
Fudan University & NTU
zkye16@fudan.edu.cn

Wook-Shin Han
POSTECH
wshan@dblab.postech.ac.kr

Byron Choi
Hong Kong Baptist University
bchoi@comp.hkbu.edu.hk

ABSTRACT

Canned patterns (*i.e.,* small subgraph patterns) in visual graph query interfaces (a.k.a GUI) facilitate efficient query formulation by enabling *pattern-at-a-time* construction mode. However, existing GUIs for querying large networks either do not expose any canned patterns or if they do then they are typically selected manually based on domain knowledge. Unfortunately, manual generation of canned patterns is not only labor intensive but may also lack diversity for supporting efficient visual formulation of a wide range of subgraph queries. In this paper, we present a novel, generic, and extensible framework called TATTOO that takes a data-driven approach to *automatically* select canned patterns for a GUI from large networks. Specifically, it first *decomposes* the underlying network into *truss-infested* and *truss-oblivious* regions. Then *candidate* canned patterns capturing different real-world query topologies are generated from these regions. Canned patterns based on a user-specified *plug* are then *selected* for the GUI from these candidates by maximizing *coverage* and *diversity*, and by minimizing the *cognitive load* of the pattern set. Experimental studies with real-world datasets demonstrate the benefits of TATTOO. Importantly, this work takes a concrete step towards realizing *plug-and-play* visual graph query interfaces for large networks.

PVLDB Reference Format:
Zifeng Yuan, Huey Eng Chua, Sourav S Bhowmick, Zekun Ye, Wook-Shin Han, and Byron Choi. Towards Plug-and-Play Visual Graph Query Interfaces: Data-driven Selection of Canned Patterns for Large Networks. PVLDB, 14(11): 1979 - 1991, 2021.
doi:10.14778/3476249.3476256

1 INTRODUCTION

A recent survey [34] revealed that graph query languages and usability are considered as some of the top challenges for graph processing. A common starting point for addressing these challenges is the deployment of a visual query interface (a.k.a GUI) that can enable an end user to draw a graph query interactively by utilizing *direct-manipulation* [36] and visualize the result matches effectively [1, 31]. A useful component of such a GUI is a panel containing a set of

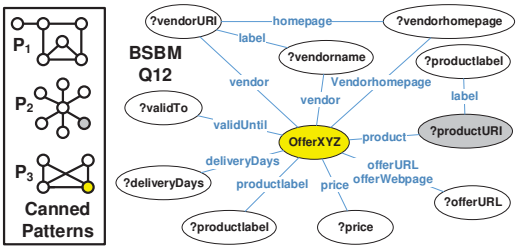

Figure 1: Q_{12} in BSBM and canned patterns.

canned patterns (*i.e.,* small subgraphs) which is beneficial to visual querying in at least three possible ways [9, 23, 24]. First, it can potentially decrease the time taken to visually construct a query by facilitating *pattern-at-a-time* query mode (*i.e.,* construct multiple nodes and edges by performing a *single* click-and-drag action) in lieu of *edge-at-a-time* mode. Second, it can facilitate "bottom-up" search when a user does not have upfront knowledge of what to search for. Third, canned patterns (patterns for brevity) may alleviate user frustration of repeated edge construction especially for larger queries.

Example 1.1. Consider the subgraph query in Figure 1 from BSBM [2] (Query Q_{12}). Suppose Wei, a non-programmer, wishes to formulate it using a GUI containing a set of canned patterns (a subset of them is shown). Specifically, he may drag and drop p_2 and p_3 on the *Query Canvas*, merge the yellow vertex of p_3 with the center vertex of p_2, add a vertex and connect it with the grey vertex of p_2. Finally, Wei can assign appropriate vertex labels. Observe that it requires five steps to construct the topology. On the other hand, if Wei takes an edge-at-a-time approach, it would require 23 steps to construct it. Clearly, canned patterns enable more efficient (*i.e.,* fewer number of steps or lesser time) formulation of the query.

It is worth noting that Wei may not necessarily have the complete query structure "in his head" during query formulation. He may find p_3 interesting while browsing the pattern set, which may initiate his bottom-up search leading to the query. Clearly, without the existence of a pattern set, such bottom-up search would be infeasible in practice. ∎

Data-driven selection of relevant canned patterns for a GUI (*e.g.,* p_1, p_2, p_3 in Fig. 1.1) is important to facilitate efficient query formulation [9, 23]. In particular, data-driven selection paves the way for *plug-and-play* visual graph query interfaces, which are like a plug-and-play device that can be plugged into any kind of socket (*i.e.,*

Proceedings of the VLDB Endowment, Vol. 14, No. 11 ISSN 2150-8097.
doi:10.14778/3476249.3476256

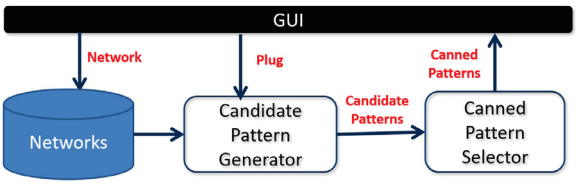

Figure 2: Overview of TATTOO.

graph data) and used. A plug-and-play GUI is dynamically built from a high-level specification of canned pattern properties known as the *plug* (detailed in Section 3). Specifically, given a network G and a plug b, the GUI is automatically constructed by populating its various components (*e.g.,* node/edge attributes, canned patterns) from G without the need for manual GUI coding. This enhances portability and maintainability of GUIs across different data sources [9].

In this paper, we present a novel framework called TATTOO (daTa-driven cAnned paTtern selecTiOn from netwOrks) that takes a data-driven approach to the *canned pattern selection* (CPS) problem for *large networks*. Given a network G, a user-specified plug specification b which is the number of canned patterns to display and their minimum and maximum permissible sizes, TATTOO automatically selects canned patterns from G that satisfy b.

The CPS problem is technically challenging. First, it is a NP-hard problem [23]. Second, the availability of query logs can facilitate the selection of relevant patterns as they provide rich information of past queries. In practice, however, such information is often publicly unavailable (*e.g.,* none of the networks in SNAP [5] reveal query logs) due to privacy and legal reasons. Hence, we cannot realistically assume the availability of query logs to select patterns. Furthermore, users may demand a GUI *prior* to querying a network. Hence, there may not exist any query log prior to the creation of a visual query interface. Third, it is paramount to find *unlabeled* patterns (*e.g.,* Example 1.1) that are potentially useful for query formulation (detailed in Sec. 4). However, the selection of such patterns is challenging as there is an exponential number of them in a large network. Fourth, these selected patterns should not only be *topologically diverse* so that they are useful for a wide variety of queries but they should also impose low *cognitive load* (*i.e.,* mental load to visually interpret a pattern's edge relationships to determine if it is useful for a query) on users. In particular, large graphs overload the human perception and cognitive systems, resulting in poor performance of tasks such as identifying edge relationships [22, 42].

At this point, a keen reader may wonder why building blocks of real-world networks (*e.g.,* paths of length k, triangle patterns) [29, 40] cannot be simply utilized as canned patterns since they have high coverage and low cognitive load. Unfortunately, it may take a larger number of steps to formulate a variety of queries using these patterns due to their small size. For instance, reconsider Example 1.1. Suppose the pattern set consists of an edge, a path of length 2 (*i.e.,* 2-path), a triangle, and a rectangle. In this case, Q_{12} may be formulated by dragging and dropping the rectangle once, the 2-path three times, construction of a single node and two edges, along with three node mergers. That is, it takes 10 steps altogether, which is more than using the patterns in Figure 1. Furthermore, these patterns do not expose "interesting" substructures to facilitate bottom-up search as they occur in almost all large real-world networks.

TATTOO addresses the aforementioned challenges as follows. It exploits a recent analysis of real-world query logs [12] to *classify* topologies of canned patterns into *categories* that are consistent with the topologies of real-world queries (detailed in Section 5). This enables us to reach a middle ground where TATTOO does not need to be restricted by the availability of query logs but yet exploit topological characteristics of real-world queries to guide the selection process. Next, it realizes a novel and efficient *candidate canned pattern generation* technique based on the classified topologies to identify potentially useful patterns. Lastly, canned patterns are *selected* from these candidates for display on the GUI based on a novel *pattern set score* that is sensitive to coverage, diversity, and cognitive load of patterns. Specifically, we leverage recent progress in the algorithm community to propose a selection algorithm that guarantees $\frac{1}{e}$-approximation [13]. Figure 2 depicts an overview of the TATTOO framework. Experiments with several real-world large networks and users reveal that TATTOO can select canned patterns within few minutes. Importantly, these patterns can reduce the number of steps taken to formulate a subgraph query and query formulation time by up to 9.7X and 18X, respectively, compared to several baseline strategies.

In summary, this paper makes the following contributions: (1) We describe TATTOO, an end-to-end canned pattern selection framework for any plug-and-play visual graph query interface for large networks independent of domains and data sources. A video of a plug-and-play interface that incorporates TATTOO can be viewed at https://youtu.be/sL0yHV1eEPw. (2) We formally introduce the *CPS problem for large networks* (Section 4) and present a novel categorization of potentially useful canned patterns in Section 5. (3) We present an efficient solution to select canned patterns for a GUI (Sections 6 - 7). Specifically, we present a novel *candidate pattern generation* framework that is grounded on topologies of real-world subgraph queries. Furthermore, for the first time in graph querying literature, we utilize the recent technique in [13] from the algorithm community to select canned patterns with good theoretical quality guarantees. (4) Using real-world networks, we show the superiority of TATTOO to several baselines (Section 8).

Formal algorithms and selected proofs of theorems and lemmas are provided in [41].

2 RELATED WORK

Most germane to our work is our prior efforts on data-driven construction of visual graph query interfaces in [10, 23, 44]. The work in [24] focuses on the maintenance of canned patterns for evolving data graphs. Our work differs from these efforts in the following ways. First, we focus on selecting *unlabelled* canned patterns from *large* networks in contrast to labelled patterns from a collection of small- or medium-sized data graphs in [10, 23, 24, 44]. Specifically, existing efforts such as CATAPULT [23] first partitions a collection of data graphs into a set of clusters and summarizes each cluster to a *cluster summary graph* (CSG). Then, it selects the canned patterns with the aforementioned characteristics from these CSGs using a weighted random walk approach. This clustering-based approach is prohibitively expensive for large networks as detailed in Section 8. Second, these approaches do not exploit characteristics of real-world subgraph queries for selecting canned patterns. In contrast,

we utilize topological characteristics of real-world queries to guide our solution design. Third, we present a novel real-world query topology-aware candidate pattern generation technique and a selection technique that provides quality guarantee. No theoretical guarantee is provided in [10, 23, 44] for selecting canned patterns. Lastly, as detailed in Section 7, the computation of *pattern score* to assess the quality of canned patterns is different as the computation of cognitive load and diversity is different from [23] due to the nature of large networks. Furthermore, in this work we provide a theoretical analysis of the pattern score.

Motif discovery techniques [19, 29] do not consider diversity and cognitive load. Sizes of these motifs are generally bounded in the range of [3-7] [19, 29]. For the same reason, it is difficult to use graphlets [7, 20, 33] as patterns. Also, frequent subgraphs [16] may not constitute good canned patterns [9] and are prohibitively expensive to compute for large networks (detailed in Section 8).

3 BACKGROUND

We first introduce several graph terminologies that we shall be using subsequently. Next, we formally define the notion of *plugs*. Finally, we briefly describe the desirable characteristics of canned patterns as introduced in [23].

3.1 Terminology

We denote a graph or network as $G = (V, E)$, where V is a set of nodes/vertices and $E \subseteq V \times V$ is a set of edges. Vertices and edges can have labels as attributes. The *size* of G is defined as $|G| = |E|$. The *degree* of a vertex $v \in V$ is denoted as $deg(v)$. In this paper, we assume that G is an undirected, unweighted graph with labeled vertices.

A *triangle* is a cycle of length 3 in G. The *support* of an edge $e = (u, v) \in E$ (denoted by $sup(e)$) is the number of triangles in G containing u and v [38]. $G_S = (V_S, E_S)$ is a *subgraph* of G (denoted by $G_S \subseteq G$) if $V_S \subseteq V$ and $E_S \subseteq E$. Consider another graph $G' = (V', E')$ where $|V| = |V'|$. G and G' are *isomorphic* if there exists a bijection $f : V \rightarrow V'$ such that $(u, v) \in E$ iff $(f(u), f(v)) \in E'$. Further, there exists a *subgraph isomorphism* from G to a graph Q if G contains a subgraph G_S that is isomorphic to Q. We refer to G_S as the *embedding* of Q in G.

Given G, the *k-truss* of G is the largest subgraph $G' = (V', E')$ of G in which every edge $e \in E'$ is contained in at least $k - 2$ triangles within the subgraph. A 2-truss is simply G itself. We define the *trussness* of an edge e as $t(e) = \max\{k | e \in E_{T_k}\}$ where $T_k = (V_{T_k}, E_{T_k})$ is the k-truss in G. Further, k_{max} denotes the maximum trussness.

3.2 Plugs

Recall that data-driven selection of canned patterns facilitates the construction of a plug-and-play visual query interface. A *plug* is a high-level specification of the patterns in a GUI. Given the specification, TATTOO dynamically generates the canned patterns satisfying it from the underlying network. Formally, it is defined as follows.

Definition 3.1. **[Plug]** *Given a network G and a GUI $\mathbb{I}$, a* ***plug*** $b = (\eta_{min}, \eta_{max}, \gamma)$ *where $\eta_{min} > 2$ (resp. η_{max}) is the minimum (resp. maximum) size of a pattern, $\gamma > 0$ is the number of patterns to be displayed on $\mathbb{I}$.*

Essentially a plug[1] is a collection of attribute-value pairs that specifies the high-level content of a canned pattern panel in a GUI. For example, $b = (3, 15, 30)$ is a plug. Accordingly, the minimum and maximum sizes of patterns in $\mathbb{I}$ are 3 and 15, respectively, and the total number of patterns to be displayed is 30. Observe that there can be multiple plugs for G as well. Similarly, the same plug can be used for different G. Hence, different GUIs can be constructed by different plug specifications.

A plug should possess the following properties. (a) *Data independence* - A plug should not depend upon a specific network (*i.e.,* socket). The specification of plug enables this by not admitting any network-specific information. Observe that this property is important for plug-and-play interfaces as a plug can be used on different network data across different application domains. (b) *Able to select canned patterns with the required specifications* - The resulting canned pattern selection mechanism should select patterns exactly as specified by the plug.

3.3 Characteristics of Canned Patterns

Since it is impractical to display a large number of patterns in $\mathbb{I}$, the number of patterns should be small and satisfy certain desirable characteristics as introduced in [23].

High coverage. A pattern $p \in \mathcal{P}$ *covers* G if G contains a subgraph s that is isomorphic to p. Since p may have many embeddings in G, the pattern set $\mathcal{P}$ should ideally cover as large portion of G as possible. Then a large number of subgraph queries on G can be constructed by utilizing $\mathcal{P}$.

High diversity. High coverage of patterns is insufficient to facilitate efficient visual query formulation [23]. In order to make efficient use of the limited display space on $\mathbb{I}$, $\mathcal{P}$ should be *structurally diverse* to serve a variety of queries. This also facilitates bottom-up search where a user gets a bird's-eye view of the diverse substructures in G.

Low cognitive load. Cognitive load refers to the memory demand or mental effort required to perform a given task [22]. A topologically complex pattern may demand substantial cognitive effort from an end user to decide if it can aid in her query formulation [23]. Hence, it is desirable for the canned patterns in $\mathcal{P}$ to impose low cognitive load on an end user to make browsing and selecting relevant patterns cognitively efficient during visual query formulation.

4 THE CPS PROBLEM

Given a data graph or network $G = (V, E)$, a visual graph query interface $\mathbb{I}$ and a user-specified plug b, the goal of the *canned pattern selection* (CPS) problem is to select a set of *unlabelled* patterns $\mathcal{P}$ for display on $\mathbb{I}$, which satisfies the specifications in b and *optimizes coverage*, *diversity* and *cognitive load* of $\mathcal{P}$.

Observe that our CPS problem differs from [23] in two key ways. First, we focus on a single large network instead of a large collection of small- or medium-sized data graphs. Second, we select *unlabelled* patterns instead of labelled ones. In large networks, a subgraph query may not always contain labels on its vertices or edges. Specifically, unlabelled query graphs are formulated in the *subgraph enumeration* problem [6] whereas query graphs are labelled in the subgraph matching problem [37]. Hence, by selecting

[1] Additional application-specific constraints (*e.g.*, pattern distribution) can be included in a plug.

unlabelled patterns TATTOO facilitates visual formulation of both these categories of queries. In particular, one may simply drag-and-drop specific vertex/edge labels from the *Attribute* panel of a GUI to add labels to the vertices/edges of a pattern (*e.g.,* Example 1).

We now formally define the CPS problem addressed in this paper. We begin by introducing *coverage*, *diversity*, and *cognitive load* of canned patterns. Let $S(p) = \{s_1, \cdots, s_n\}$ be a bag of subgraphs in G isomorphic to p (*i.e.,* embeddings of p) where vertex labels in $G = (V, E)$ and $p = (V_p, E_p)$ are assumed to be the same and $s_i = (V_i, E_i)$. We say an edge $e \in E_i$ is *covered* by p. The *coverage* of p is given as $cov(p) = |\bigcup_{i \in |S(p)|} E_i|/|E|$. Similarly, $cov(\mathcal{P}) = |E^{\dagger}|/|E|$ (*i.e.,* $f_{cov}(\mathcal{P})$) where every $e \in E^{\dagger}$ is covered by at least one $p \in \mathcal{P}$. Since $|E|$ is constant for a given G, *coverage* can be rewritten as $cov(p) = |\bigcup_{i \in |S(p)|} E_i|$ and $cov(\mathcal{P}) = |E^{\dagger}|$. The *diversity* of p w.r.t to $\mathcal{P}$ is the inverse of *similarity* of p. In particular, the *similarity* of a set of canned patterns $\mathcal{P}$ is denoted as $f_{sim}(\mathcal{P}) = \sum_{(p_i, p_j) \in \mathcal{P} \times \mathcal{P}} sim(p_i, p_j)$ where $sim(p_i, p_j)$ is the similarity between patterns p_i and p_j (detailed in Section 7). Finally, we measure *cognitive load* of p (denoted by $cog(p)$) based on the size, density, and edge crossings in p (detailed in Section 7) as a user tends to spend more time identifying relationships between vertices in denser graphs with more edge crossings [21, 22, 42]. The cognitive load of $\mathcal{P}$ (*i.e.,* $f_{cog}(\mathcal{P})$) is given as $\sum_{p \in \mathcal{P}} cog(p)$.

Definition 4.1. **[CPS Problem]** *Given a network G, a GUI $\mathbb{I}$, and a plug $b = (\eta_{min}, \eta_{max}, \gamma)$, the goal of* ***canned pattern selection (CPS) problem*** *is to find a set of unlabelled canned patterns $\mathcal{P}$ from G that satisfies*

$$\begin{aligned} &\max f_{cov}(\mathcal{P}), -f_{sim}(\mathcal{P}), -f_{cog}(\mathcal{P}) \\ &\text{subject to } |\mathcal{P}| = \gamma, \mathcal{P} \in \mathcal{U} \end{aligned} \tag{1}$$

where $\mathcal{P}$ is the solution; $\mathcal{U}$ is the feasible set of canned pattern sets in G; $f_{cov}(\mathcal{P})$, $f_{sim}(\mathcal{P})$ and $f_{cog}(\mathcal{P})$ are the coverage, similarity, and cognitive load of $\mathcal{P}$, respectively.

Remark. Observe that CPS is a multi-objective optimization problem as our goal is to maximize coverage and diversity (*i.e.,* minimize similarity) of canned patterns while minimizing their cognitive load. Hence, we address it by converting CPS into a single-objective optimization problem using a *pattern score* (detailed in Section 7). Also, observe that we aim to find patterns of size greater than 2 (*i.e.,* $\eta_{min} > 2$). Small-size patterns that are basic building blocks of networks [29, 40] (*e.g.,* edge, 2-path, triangle) are provided by *default* for *all* datasets (*i.e.,* referred to as *default patterns*).

The CPS problem is shown to be NP-hard in [23, 41] by reducing it from the classical maximum coverage problem.

5 CATEGORIES OF CANNED PATTERNS

In theory, numerous different patterns can be selected from a given network. Which of these are "useful" for subgraph query formulation in practice? In this section, we provide an answer to this question.

5.1 Topologies of Real-world Queries

Although basic building blocks of networks [29, 40] are presented as default patterns in our GUI, as remarked earlier, they are insufficient as they do not expose to a user more domain-specific and larger patterns in the underlying data. Such larger substructures not only facilitate more efficient construction of subgraph queries but also guide users for bottom-up search by exposing substructures that are network-specific. However, which topologies of these substructures should be considered for canned patterns?

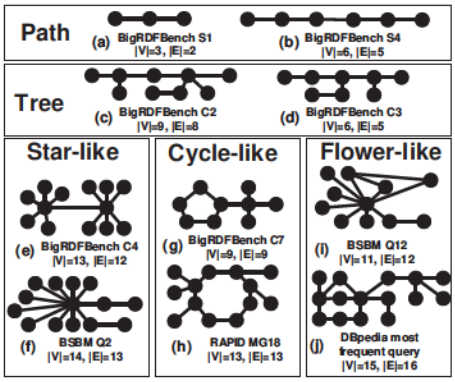

Figure 3: Examples of real-world query topologies.

Ideally, real-world subgraph query logs can provide guidance to resolve this challenge. However, as remarked in Section 1, such data may be unavailable. Hence, we exploit results from a recent study [12] that analysed a large volume of real-world SPARQL query logs. It revealed that topologies of many real-world subgraph queries map to chains, trees, stars, cycles, petals, and flowers[2] [12]. Figure 3 depicts examples of these topologies in real-world subgraph queries extracted from BigRDFBench [35], BSBM [2], Rapid [4], and DBPedia [17]. Consequently, canned patterns in any GUI should facilitate efficient construction of these topologies.

5.2 Topologies of Canned Patterns

We consider the following types of topological structures of canned patterns in order to facilitate construction of the above query substructures.

Path and cycle patterns. A subgraph query may contain paths of different lengths (*i.e.,* chain) and/or cycles. Figure 3 depicts some examples. Hence, our canned patterns should expose representative *k-paths* and *k-cycles* in the underlying data. Given a graph $G = (V, E)$, a *k-path*, denoted as $P_k = (V_k, E_k)$, is a walk of length k containing a sequence of vertices $v_1, v_2, \cdots, v_k, v_{k+1}$ where $E_k \subseteq E$, $V_k \subseteq V$ such that all vertices in V_k are distinct. A *k-cycle* is simply a closed $(k-1)$-path where $k \geq 3$.

Star and asterism patterns. Intuitively, a *star* is a connected subgraph containing a vertex r where the remaining vertices are connected only to r (*i.e.,* neighbors of r). A *k-star* is a single-level, rooted tree $S_k = (V, E)$ where $V = \{r\} \bigcup L$, r is the root vertex and L is the set of leaves such that $\forall e = \{u, v\} \in E, u = r, v \in L$ and $|V| = k + 1$. We refer to the root as the *center vertex*. Note that $k \geq \epsilon$ where ϵ is the minimum value of k for which the single-level rooted tree is considered a star.

Real-world queries may contain multiple k-stars that are *combined* together. For instance, the query topology in Figure 3(e) is a combination of 6-star and 7-star by merging on a pair of

[2] A *petal* is a graph consisting of a source node s, target node t and a set of at least 2 node-disjoint paths from s to t. A *flower* is a graph consisting of a node x with three types of attachments: chains (stamens), trees that are not chains (the stems), and petals. A *flower set* is a graph in which every connected component is a flower.

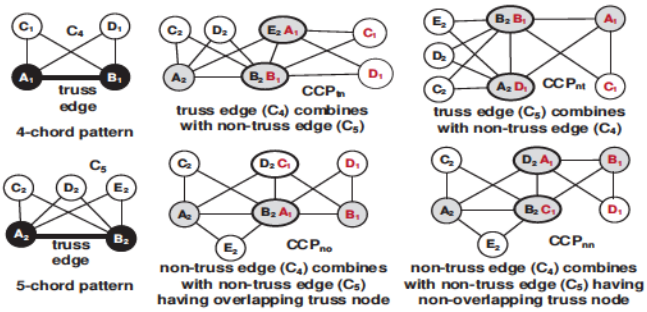

Figure 4: k-chord and composite chord patterns. Grey nodes are truss nodes and oval-shaped nodes are combined nodes.

Table 1: No. of steps for constructing queries.

ID	Edge-at-a-time	Default patterns	Canned patterns
(c)	17	6 [2 2-path + 1 square - 1 edge + 1 edge + 1 merge]	5 [4-path + 2 2-path + 2 merge] 5 [4-star + 1 2-path + 1 node + 2 edge]
(e)	25	11 [5 2-path + 1 node + 2 edge + 3 merge]	1 [$A_{6,7}$] 3 [5-star + 6-star + 1 edge]
(g)	18	8 [4 2-path + 1 edge + 3 merge]	3 [5-cycle + 4-star + 1 merge] 4 [6-path + 2-path + 1 edge + 1 merge]
(i)	23	10 [square + 3 2-path + 1 node + 2 edge + 3 merge]	5 [4-CP + 6-star + 1 node + 1 edge + 1 merge] 5 [CCP_{no}(4,4) - 2 edge + 5-star + 1 merge]

edges. Hence, our canned pattern topology also involves stars that form an *asterism* pattern by *merging* them on a pair of edges. Formally, given n stars $S = \{S_{k_1}, \cdots, S_{k_n}\}$ and $n-1$ merged edges $E_m = \{e_{m_1}, \cdots, e_{m_{n-1}}\}$ where $S_{k_i} = (V_i, E_i)$ and $e_{m_i} \in E_i$, let $R = \{r_1, \cdots, r_n\}$ be the center vertices such that $r_i \in V_i$. The *asterism* pattern of S is defined as $A_S = (V, E)$ where $e_i = (r_i, v_i)$, $e_{i+1} = (r_{i+1}, v_{i+1})$, $E = \bigcup_{1 \le i < n}(\{(r_i, r_{i+1})\} \bigcup (E_i \setminus \{e_i\}) \bigcup (E_{i+1} \setminus \{e_{i+1}\}))$, $V = \bigcup_{1 \le i < n}((V_i \setminus \{v_i\}) \bigcup (V_{i+1} \setminus \{v_{i+1}\}))$, $k_i \ge \epsilon$ and $|E| \le \eta_{max}$.

k-chord and composite chord patterns. Observe that tree-structured queries can be constructed by combining chains and stars (*e.g.,* Figure 3(c)-(d)). However, they are insufficient to construct more complex petal and flower queries efficiently. In particular, petal and flower queries may often contain *triangle-like* structures. For example, the query in Figure 3(i) contains two triangles. Hence, at first glance it may seem that we can simply select different k-trusses ($k > 2$) of sizes within the plug specification b as canned patterns. However, a subgraph query may not necessarily always contain k-trusses. For instance, the query in Figure 3(j) contains multiple "triangle-like" structures as some common edges of triangles are missing. Consequently, the use of only k-truss as a canned pattern may make query formulation inefficient as it demands deletion of multiple edges in order to construct a triangle-like query topology. This increases the number of steps required to formulate a query, thereby increase the formulation time. Hence, it is desirable to have "k-truss-like" substructures as patterns.

To this end, we extract two types of k-truss-based structures as canned patterns, namely, *k-chord patterns* (k-CP) and *composite chord patterns* (CCP). Intuitively, a k-CP is a connected graph containing a *truss edge* e (*i.e.,* edge belonging to a k-truss) and k-2 triangles of e. Formally, given a k-truss $G_k = (V_k, E_k)$ for $k > 2$, the *k-chord pattern* (k-CP) $C_k = (V_{ck}, E_{ck})$ associated with every edge $e = (u, v) \in E_k$ where $u, v \in V_k$ is defined as $V_{ck} = \{u, v\} \bigcup V'_{ck}$ and $E_{ck} = \{(u, v)\} \bigcup E'_{ck}$ where $V'_{ck} = \{w_i : 0 \le i \le k-2\}$ and $E'_{ck} = \{(u, w_i), (w_i, v) : 0 \le i \le k-2\}$. k-CP can be considered as a building block of k-trusses since it is found with respect to each edge in a given k-truss. Examples of k-CPs (4-CP and 5-CP) are illustrated in Figure 4. We refer to the edge in a k-chord pattern that is involved in (k-2) triangles as a *truss edge* and the remaining edges as *non-truss edges*. For example, in Figure 4, edges (A_1, B_1) and (A_2, B_2) are truss edges whereas (A_1, C_1) and (B_2, D_2) are non-truss edges. Correspondingly, vertices of a truss edge (*e.g.,* A_1, B_1, A_2, B_2) are referred to as *truss vertices*. Observe that we can formulate a simple petal query in two steps by selecting the 4-CP pattern and deleting the truss edge.

To select larger canned patterns with greater structural diversity, we *combine* k-CPs to yield additional *composite chord patterns* (CCP) that occur in the underlying network. Observe that combining a set of k-CPs in different ways results in different patterns as demonstrated in Figure 4. However, this is an overkill as they are not only expensive to compute but also may generate patterns with higher density (higher cognitive load) or are larger than η_{max}. Hence, we focus on the CCP generated by merging a *single* edge of two k-CPs as it not only reduces the complexity of CCP generation, but also produces CCPs with lower density.

Unique small graph patterns. Lastly, we find small connected subgraphs that do not fall under above categories but occur multiple times in the underlying network.

Table 1 reports the number of steps taken by various modes of query construction of selected query topologies in Figure 3. Observe that query construction using canned patterns often takes fewer number of steps compared to construction using only default patterns, emphasizing the need for patterns beyond the default ones. One can also formulate a specific query following multiple alternatives, *i.e.,* using multiple sets of patterns (canned and default). This gives users the flexibility to formulate a query using these patterns in many ways, all of which often take fewer steps compared to the edge-at-a-time or default pattern-based modes.

Table 2: TIR and TOR graphs in real networks.

Data	Name	$\|V\|$	$\|E\|$	% (G_T)	% (G_O)
BK	loc-Brightkite	58K	214K	67.3	32.7
GO	loc-Gowalla	197K	950K	78.2	21.8
DB	com-DBLP	317K	1.05M	93	7
AM	com-Amazon	335K	926K	77.2	22.8
RP	RoadNet-PA	1.09M	1.54M	12.7	87.3
YT	com-Youtube	1.13M	2.99M	46.8	53.2
RT	RoadNet-TX	1.38M	1.92M	12.5	87.5
SK	as-Skitter	1.7M	11M	79.1	20.9
RC	RoadNet-CA	1.97M	2.77M	12.6	87.4
LJ	com-LiveJournal	4M	34.7M	83.2	16.8

6 CANDIDATE PATTERNS GENERATION

In the preceding section, we classified the topologies of canned patterns broadly into "k-truss-like" and "non-k-truss-like" structures. In this section, we describe how candidate canned patterns conforming to these topological categories are extracted from the underlying network G. To this end, we first *decompose* G into *truss-infested* and *truss-oblivious regions* and then generate "k-truss-like" and "non-k-truss-like" candidate patterns from these regions, respectively. We discuss these two steps in turn.

6.1 Truss-based Graph Decomposition

In order to extract "non-k-truss-like" and "k-truss-like" structures as candidate patterns, we first decompose a network G into *sparse* (containing non-k-trusses) and *dense* (containing k-trusses) regions. The latter region is referred to as *truss-infested region* (TIR graph) and the former *truss-oblivious region* (TOR graph), and are denoted by G_T and G_O, respectively. Table 2 reports the sizes of G_T and G_O in several real-world networks measured as the percentage of the total number of edges. We observe G_T basically consists of relatively large connected subgraphs that comprise multiple k-trusses. On the other hand, G_O mainly consists of chains (*i.e.,* paths), stars, cycles, and small connected components. Furthermore, although some networks have small G_O (*e.g.,* com-DBLP), there are networks where G_O is large (*e.g.,* RoadNet-CA), encompassing up to 87.5% of the total number of edges. Consequently, by decomposing a network into G_T and G_O, we can improve efficiency [41] by limiting the search for k-truss-like patterns in G_T instead of the entire network and extract non-truss-like patterns from G_O. Additionally, generating candidate patterns of aforementioned topological categories from *both* TIR and TOR graphs enables us to select a *holistic* collection of patterns having higher coverage and diversity. Cognitive load of the pattern set is often reduced when patterns from both regions are considered due to the sparse structure of TOR [41].

TATTOO utilizes the state-of-the-art truss decomposition approach in [39] to decompose G into G_T and G_O. Briefly, this approach identifies k-trusses ($k \in [2 - k_{max}]$) in G iteratively by removing edges with support less than $k-2$ from G. Hence, our graph decomposition algorithm adapts it to assign 2-truss as G_O and the remaining k-trusses as G_T.

We keep track of the edge trussness (denoted as $t(e)$) in G_T. Since the goal is to select canned patterns with maximum size η_{max}, the upper bound of edge trussness is set to this value. The algorithm first identifies the support of each edge. Then, regions of the data graph are iteratively extracted by removing edges with the lowest support, starting from the sparsest (*i.e., sup*$(e) = 0$) to the densest. In particular, TATTOO considers all edges with $sup(e) = 0$ as sparse regions and these edges form the TOR graph G_O. The remaining edges form the TIR graph G_T.

In summary, the above approach makes the following two simple modifications to the truss decomposition technique in [39]: (1) instead of storing each k-truss as a separate graph, it stores 2-truss as G_O and the remaining k-trusses are combined as a single graph G_T; (2) it assigns a trussness value $t(e)$ to every edge in G_T and G_O. The worst-case time and space complexities of this algorithm are $O(|E|^{1.5})$ and $O(|V| + |E|)$, respectively [39].

6.2 Patterns from a TIR Graph

Next, we generate k-CPs and CCPs as candidate patterns from a TIR graph. For each pattern we also compute its frequency as it will be used subsequently to measure its coverage. We discuss them in turn.

Generation of k-chord patterns. We can find k-CPs with respect to each edge in a given k-truss. For instance, every edge in a 4-truss and a 5-truss is part of at least 2 and 3 triangles, respectively. Observe that the 2-chord pattern of an edge e is simply the edge itself. Hence, TATTOO generates k-CPs for $k \geq 3$. The *frequency* of a

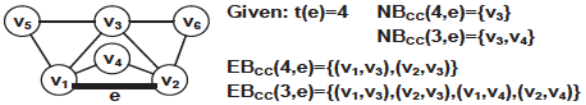

Figure 5: k-CCP node and edge neighbourhoods.

k-CP is measured by the frequency of the pattern occurring in G_T, which is essentially the number of edges having trussness greater than or equals to k. Formally, given a TIR graph $G_T = (V_T, E_T)$ and a k-chord pattern $C_k = (V_{ck}, E_{ck})$, the *frequency* of C_k is defined as $freq(C_k) = |\{e \in E | t(e) \geq k\}|$. Then, the set of k-CPs of a G_T is simply the set of patterns C_k whose frequency is greater than 0. We first generate k-chord patterns in G_T and then compute their frequencies using edge trussness.

LEMMA 6.1. *The worst-case time and space complexities of k-CP generation are $O(k_{max}|E_T|^{1.5})$ and $O(|V_T| + |E_T|)$, respectively.*

Generation of composite chord patterns. Next, we generate the CCPs. Specifically, we generate the following categories of CCPs based on different ways of merging truss and non-truss edges.

Definition 6.2. Let $C_{k_1} = (V_{ck_1}, E_{ck_1})$ and $C_{k_2} = (V_{ck_2}, E_{ck_2})$ be two k-chord patterns where $s, t \in V_{ck_1}$ and $u, v \in V_{ck_2}$ are truss vertices. Then, we can generate the following categories of ***composite chord patterns*** *of C_{k_1} and C_{k_2} by merging C_{k_1} and C_{k_2} as follows:*

1. *$CCP_{tn}(k_1, k_2)$: merge the truss edge of C_{k_1} with a non-truss edge of C_{k_2}.*
2. *$CCP_{nt}(k_1, k_2)$: merge the truss edge of C_{k_2} with a non-truss edge of C_{k_1}.*
3. *$CCP_{no}(k_1, k_2)$: merge a non-truss edge of C_{k_1} with a non-truss edge of C_{k_2} such that there is an overlapping truss vertex.*
4. *$CCP_{nn}(k_1, k_2)$: merge a non-truss edge of C_{k_1} with a non-truss edge of C_{k_2} such that there is no overlapping truss vertex.*

Figure 4 depicts examples of these four categories of CCPs. When the context is clear, we shall simply refer to a CCP as CCP_i. A keen reader may observe that it is possible to create another CCP by merging the truss edge of C_{k_1} with the truss edge of C_{k_2}. However, this CCP is in fact a k-CP where $k = k_1 + k_2 - 2$. For instance, when C_4 and C_5 in Figure 4 are merged on their truss edges, the resultant pattern is a 7-CP. Also, combining two 3-CPs always yields a 4-CP (Lemma 6.3). Since k-CPs have already been handled earlier, these combinations are ignored.

LEMMA 6.3. *Two 3-CPs always yield a CCP that is 4-CP.*

PROOF. (Sketch.) The 3-truss pattern $C_3 = (V_{c3}, E_{c3})$ is a triangle. Then, $\forall e = (u, v) \in E_{c3}$, there is a vertex w that is adjacent to both u and v. Hence, all different types of single edge merger between two C_3 produces a pattern with a merged edge $e_m = (x, y)$ and vertices x and y have two common adjacent vertices w_1 and w_2. This is essentially C_4 where its truss edge corresponds to the merged edge of the two C_3. □

We now elaborate on how the CCPs and their *frequencies* are computed in TATTOO efficiently. We shall introduce two terminologies related to *node* and *edge neighbourhoods* of a CCP to facilitate exposition. Given an edge $e = (u, v)$ in a k-truss, the *k'-CCP node neighbourhood* (denoted as $NB_{cc}(k', e)$) of e is a set of vertices W adjacent to

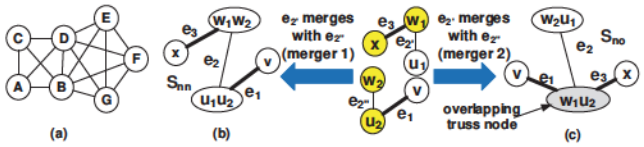

Figure 6: (a) A G_T; (b) Skeleton structure of CCP_{nn}; (c) Skeleton structure of CCP_{no}. e_1 and e_3 are truss edges.

u and v such that $\forall w \in W$, $t((u,w)) \geq k'$ and $t((w,v)) \geq k'$ where $k' \leq k$. The k'-CCP *edge neighbourhood* (denoted as $EB_{cc}(k', e)$) of e is the set of edges S adjacent to e such that $\forall (u, x_1), (x_2, v) \in S$, $x_1, x_2 \in NB_{cc}(k', e)$ where $k' \leq k$. Figure 5 illustrates examples of k'-CCP node and edge neighborhoods. For instance, $NB_{cc}(4, e)$ consists of v_3 since $t(v_1, v_3) \geq 4$ and $t(v_2, v_3) \geq 4$.

LEMMA 6.4. *Given a truss edge e, there is at least a k-chord pattern C_k on e if $|NB_{cc}(k, e)| \geq (k-2)$.*

PROOF. (Sketch). Observe that k-chord pattern on an edge $e = (u, v)$ implies that k-2 triangles in the graph contain e. Since $\text{NB}_{cc}(k, e)$ is the set of nodes W adjacent to u and v such that $\forall w \in W$, $t((u,w)) \geq k$ and $t((w,v)) \geq k$, $|\text{NB}_{cc}(k, e)|$ is equivalent to the number of triangles around e. Hence, when $|\text{NB}_{cc}(k, e)| \geq (k-2)$, a k-chord pattern must exist on e. □

Frequencies of $CCP_{tn}(k_1, k_2)$ and $CCP_{nt}(k_1, k_2)$. Consider two different k-CPs. CCP_{tn} and CCP_{nt} involve merger of a truss edge belonging to one k-CP with a non-truss edge belonging to another k-CP. Given two k-CPs C_{k_1} and C_{k_2}, let edges e_1 and e_2 be the truss edges of C_{k_1} and C_{k_2}, respectively. Intuitively, a pattern is a $CCP_{tn}(k_1, k_2)$ if it contains an embedding of C_{k_1} and of C_{k_2} whereby there is an edge e_m in the pattern that belongs to the two embeddings such that e_m is a truss edge of C_{k_1}'s embedding and is a non-truss edge of C_{k_2}'s embedding, respectively. In other words, C_{k_1} and C_{k_2} can form a CCP ($CCP_{tn}(k_1, k_2)$) by merging a truss edge e_1 from C_{k_1} with a non-truss edge from C_{k_2} if the following conditions are satisfied: (a) *Condition 1:* There is a C_{k_1} pattern on e_1 containing e_2. (b) *Condition 2:* There is a C_{k_2} pattern on e_2 where $e_2 \neq e_1$.

Note that due to Lemma 6.4, *Condition 1* holds if $|NB_{cc}(k_2, e_2) \setminus \{u, v\}| \geq (k_2 - 2)$ where $e_1 = (u, v)$. Further, if $|NB_{cc}(k_1, e_1) \bigcup NB_{cc}(k_2, e_2) \setminus \{u, v\}| \geq (k_1 - 2) + (k_2 - 2))$, then the pattern $CCP_{tn}(k_1, k_2)$ must exist. Hence, TATTOO checks the conditions iteratively on decreasing k_2 and skips checks for $k_2' < k_2$ if the conditions are satisfied for k_2. The *frequency* of $CCP_{tn}(k_1, k_2)$ is simply the number of such e_1 edges. For $CCP_{nt}(k_1, k_2)$, the approach is the same by swapping C_{k_1} with C_{k_2}.

Frequencies of $CCP_{nn}(k_1, k_2)$ and $CCP_{no}(k_1, k_2)$. Recall that (Def. 6.2) a single-edge merge can also involve the merger of two non-truss edges, each from a different k-CP. Each non-truss edge contains a truss vertex. There are two ways in which two non-truss edges can merge as shown in Figures 6(b) and (c). In the former (resp. latter), vertex pairs (w_1, w_2) (resp. (w_2, u_1)) and (u_1, u_2) (resp. (w_1, u_2)) are merged. Hence, a pattern is a CCP_{nn} if it contains at least one embedding of a structure shown in Figure 6(b) which we refer to as the *skeleton structure* of CCP_{nn} (denoted as $\mathbb{S}_{nn}$). Hence, we can search for the $\mathbb{S}_{nn}$ of a CCP_{nn} in a TIR graph to compute its occurrence and frequency. Specifically, a CCP_{nn} can be obtained if the followings are satisfied: (a) *Condition 1*: There is a C_{k_1} pattern on its truss edge $e_1 = (u_1u_2, v)$ which contains $e_2 = (u_1u_2, w_1w_2)$. (b) *Condition 2*: There is a C_{k_2} pattern on its truss edge $e_3 = (w_1w_2, x)$ which contains e_2.

Note that *Condition 1* holds if $|NB_{cc}(k_1, e_1) \setminus \{u_1u_2, w_1w_2\}| \geq (k_1 - 3)$ (Lemma 6.4). Similarly, *Condition 2* holds if $|NB_{cc}(k_2, e_3) \setminus \{u_1u_2, w_1w_2\}| \geq (k_2 - 3)$. Further, if $|NB_{cc}(k_1, e_1) \setminus \{u_1u_2, w_1w_2\} \bigcup NB_{cc}(k_2, e_3) \setminus \{u_1u_2, w_1w_2\}| \geq (k_1 - 3) + (k_2 - 3)$, then the pattern CCP_{nn} must exist. The *frequency* of a CCP_{nn} is simply the number of skeleton structures $\mathbb{S}_{nn}$ in a TIR graph.

CCP_{no} is very similar to CCP_{nn} except that the truss vertices of the merged edges are not combined during the merger. Figure 6(c) illustrates the *skeleton structure* of a CCP_{no} ($\mathbb{S}_{no}$), which occurs in all CCP_{no}. The *frequency* of a CCP_{no} is the number of skeleton structures $\mathbb{S}_{no}$.

Observe that $freq(CCP_{nn}(k_1, k_2)) = freq(CCP_{no}(k_2, k_1))$ since k_1 and k_2 can be swapped. The same is true for CCP_{tn} and CCP_{nt}. Hence, when combining two k-CPs, we only consider the case when $k_1 \geq k_2$.

Algorithm. Putting the above strategies together, the CCPs are computed as follows. For each edge in G_T, compute the k_1-CCP node and edge neighbourhoods. Next, it computes the four types of CCPs based on the above strategies. Note that the smallest CCP generated is a CCP(3,4) due to Lemma 6.3. Also, we only compute $CCP_{tn}(k_1, k_2)$ instead of both $CCP_{tn}(k_1, k_2)$ and $CCP_{nt}(k_1, k_2)$ as $CCP_{nt}(k_1, k_2)$ is covered when k_2 and k_1 are swapped.

THEOREM 6.5. *The worst-case time and space complexities of the CCP generation technique are $O(k_{max}^2 |E_T||EB_{max}|^2)$ and $O(k_{max}|E_T| + |V_T|)$, respectively.*

6.3 Patterns from a TOR Graph

Generation of candidates from a TOR graph consists of two phases: *star pattern extraction* and *small pattern extraction*. The former extracts star and asterism patterns. Subsequently, the edges involved in these patterns are removed from G_O resulting in further decomposition of the TOR graph. The resultant graph is referred to as the *remainder graph* (G_R). Then, the second phase extracts paths, cycles, and small connected subgraphs from G_R.

Extraction of star and asterism patterns. The frequencies of these patterns can be derived directly from their definitions (Section 5.2). Specifically, $freq(S_k) = |\{v | v \in V_O, deg(v) = k\}|$ and $freq(A_S) = freq(\{E_m = \{e_{m_1}, \ldots, e_{m_{n-1}}\})$ where $e_{m_i} = (r_i, r_{i+1}) \in E_O$, $\{k, k_i\} \geq \epsilon$, $deg(r_i) = k_i$ and $deg(r_{i+1}) = k_{i+1}$. Briefly, asterism patterns are found using breadth-first search (BFS). A vector of vertices is used to keep track of star centers in an asterism pattern. We "grow" the pattern by adding a neighbouring vertex z of the current star center being considered only if $deg(z) \geq \epsilon$ and when the size of the grown pattern is less than or equals to η_{max}.

LEMMA 6.6. *The worst-case time and space complexities of star and asterism pattern extraction are $O(|V_O|^2)$ and $O(|E_O| + |V_O|)$, respectively.*

Extraction of small patterns. The remainder graph G_R is primarily composed of small connected components such as paths, cycles, and subgraphs with unique topology. We refer to small subgraph patterns as connected components in G_R that are neither

k-paths nor k-cycles. Note that 1-path, 2-path, 3-cycle and 4-cycle are basic building blocks of real-world networks [29]. Recall that in TATTOO, we consider them as *default* patterns and they are not part of the candidate canned pattern set. Hence, we extract all k-paths for $k > 2$ and k-cycles for $k > 4$ and their frequencies. After that, small connected subgraphs and their corresponding frequencies are extracted.

LEMMA 6.7. *Worst-case time and space complexities to find small patterns are $O(\eta_{max}|V_R|\eta_{max}!)$ and $O(|E_R| + |V_R|)$, respectively.*

Remark. Exponential time complexity of the small pattern extraction phase is due to the isomorphism check. The time cost is small in practice due to the small size of candidate patterns and their number is typically small in G_R.

7 SELECTION OF CANNED PATTERNS

In this section, we describe the algorithm to select canned pattern set $\mathcal{P}$ from the generated candidate patterns. We begin by presenting the theoretical underpinning that influences the design of our algorithm.

7.1 Theoretical Analysis

Due to the hardness of the CPS problem, we design an approximation algorithm to address it. We draw on insights from a related problem, *team formation problem* (TFP) [11, 14], which aims to hire a team of individuals T from a group of experts S for a specific project where $T \subseteq S$. Bhowmik *et al.* [11] proposed that several aspects should be considered in TFP, namely, skill coverage (*skill*), social compatibility (*social*), teaming cost (*team*) and miscellaneous aspects such as redundant skills avoidance (*red*) and inclusion of selected experts (*exp*). The formulation of TFP is given as $s(T') = \alpha_{skill}f_{skill}(T') - \alpha_{social}f_{social}(T') - \alpha_{team}f_{team}(T') - \alpha_{red}f_{red}(T') + \alpha_{exp}f_{exp}(T')$ where α_{skill}, α_{social}, α_{team}, α_{red} and α_{exp} are non-negative coefficients that represent the relative importance of each aspect of team formation [11]. The goal is to find a team $T' \subseteq S$ where the *non-negative* and *non-monotone* function $s(T')$ is maximized. According to [11], this formulation can be posed as an *unconstrained submodular function maximization problem* which is NP-hard for arbitrary submodular functions.

Selecting a set of canned patterns in CPS is akin to hiring a team of individuals in TFP where f_{skill}, f_{red}, f_{team} correspond to f_{cov}, f_{sim} and f_{cog}, respectively. Hence, CPS can be formulated in the form $s(P') = \alpha_{f_{cov}}f_{cov}(P') - \alpha_{f_{sim}}f_{sim}(P') - \alpha_{f_{cog}}f_{cog}(P')$ (Definition 7.1) where P' is the set of candidate patterns which yields an optimized $s(P')$.

Definition 7.1. **[Pattern Set Score]** *Given a pattern set $\mathcal{P}'$, the **score** of $\mathcal{P}'$ is $s(\mathcal{P}') = \frac{1}{3|\mathcal{P}'|}(f_{cov}(\mathcal{P}') - f_{sim}(\mathcal{P}') - f_{cog}(\mathcal{P}') + 2|\mathcal{P}'|)$ where f_{cov}, f_{sim} and f_{cog} are the coverage, similarity and cognitive load of $\mathcal{P}'$, respectively.*

Definition 7.2. **[Good Candidate Pattern]** *Given a pattern set $\mathcal{P}'$ and two candidate patterns p_1 and p_2, p_1 is considered a **good candidate pattern** if $s(\mathcal{P}' \bigcup p_1) > s(\mathcal{P}' \bigcup p_2)$ and is added to $\mathcal{P}'$ instead of p_2.*

Note that Definition 7.2 can be utilized for determining inclusion of a candidate pattern in $\mathcal{P}$. Next, we analyze the properties of f_{cov}, f_{sim}, f_{cog}, and the pattern score.

LEMMA 7.3. *Coverage of a pattern set $\mathcal{P}$, $f_{cov}(\mathcal{P})$, is submodular.*

PROOF. Given a set of n elements (N), a function $f(.)$ is submodular if for every $A \subseteq B \subseteq N$ and $j \notin B$, $f(A \bigcup \{j\}) - f(A) \geq f(B \bigcup \{j\}) - f(B)$. Given a graph G and canned pattern sets $\mathcal{P}_A$ and $\mathcal{P}_B$ where $\mathcal{P}_A \subseteq \mathcal{P}_B$, let the coverage of $\mathcal{P}_A$ and $\mathcal{P}_B$ be $f_{cov}(\mathcal{P}_A)$ and $f_{cov}(\mathcal{P}_B)$, respectively. Observe that $\mathcal{P}_B$ consists of $\mathcal{P}_A$ and additional patterns (*i.e.*, $\mathcal{P}' = \mathcal{P}_B \setminus \mathcal{P}_A$). For each canned pattern $p \in \mathcal{P}'$, we let $s = \min(|f_{cov}(p)|, |f_{cov}(\mathcal{P}_A)|)$ and K denotes the overlapping set $f_{cov}(p) \bigcap f_{cov}(\mathcal{P}_A)$. The coverage of p falls under one of the four possible scenarios: (1) $K = f_{cov}(p)$ if $s = |f_{cov}(p)|$, (2) $K = f_{cov}(\mathcal{P}_A)$ if $s = |f_{cov}(\mathcal{P}_A)|$, (3) K is an empty set and (4) otherwise (*i.e.*, $0 < |f_{cov}(p) \bigcap f_{cov}(\mathcal{P}_A)| < s$).

In the case where coverage of every p falls under scenario 1, $f_{cov}(\mathcal{P}_A) = f_{cov}(\mathcal{P}_B)$. If any p falls under scenario 2, 3 or 4, then $f_{cov}(\mathcal{P}_A) \subset f_{cov}(\mathcal{P}_B)$. Hence, $f_{cov}(\mathcal{P}_A) \subseteq f_{cov}(\mathcal{P}_B)$. Consider a pattern $p' \notin \mathcal{P}_B$, let $t = \min(|f_{cov}(p')|, |f_{cov}(\mathcal{P}_A)|)$. Suppose $f_{cov}(p') \bigcap f_{cov}(\mathcal{P}_A) = f_{cov}(p')$ where $|f_{cov}(p')| < |f_{cov}(\mathcal{P}_A)|$ (Scenario 1), then $f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A)$ is an empty set. Note that we use the minus and set minus operator interchangeably in this proof. Since $f_{cov}(\mathcal{P}_A) \subseteq f_{cov}(\mathcal{P}_B)$, $f_{cov}(\mathcal{P}_B \bigcup \{p'\}) = f_{cov}(\mathcal{P}_B)$. Hence, $f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A) = f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)$.

Now, consider $f_{cov}(p') \bigcap f_{cov}(\mathcal{P}_A) = f_{cov}(\mathcal{P}_A)$ where $|f_{cov}(p')| > |f_{cov}(\mathcal{P}_A)|$ (Scenario 2). $f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A) = f_{cov}(p') - f_{cov}(\mathcal{P}_A)$ where $f_{cov}(\mathcal{P}_A) \subset f_{cov}(p')$. Let L and M be $f_{cov}(p') \setminus f_{cov}(\mathcal{P}_A)$ and $f_{cov}(\mathcal{P}_B) \setminus f_{cov}(\mathcal{P}_A)$, respectively. Observe that, similar to previous observation, it is possible for (1) L to be fully contained in M if $|L| < |M|$, (2) M to be fully contained in L if $|M| < |L|$, (3) $L \bigcap M$ to be empty or (4) otherwise (*i.e.*, $0 < |L \bigcap M| < t$ where $t = \min(|L|, |M|)$). Hence, $|L \bigcap M| \in [0, t]$. When $|L \bigcap M| = 0$, $f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A) = f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)$. Otherwise, there are some common graphs covered by L and M, resulting in $f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B) = L \setminus (L \bigcap M)$. Hence, $|f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A)| > |f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)|$. Taken together, for scenario 2, $|f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A)| \geq |f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)|$.

Scenario 3 is similar to scenario 2 where L is $f_{cov}(p')$. $f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A) = L$ and $f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B) = L \setminus (L \bigcap M)$. Since $|L \bigcap M| \in [0, t]$, $|f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A)| \geq |f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)|$.

Scenario 4 is the same as scenario 3 except that $L = f_{cov}(p') \setminus (f_{cov}(\mathcal{P}_A) \bigcap f_{cov}(p'))$. Observe that $|f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A)| \geq |f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)|$ as $|L \bigcap M| \in [0, t]$.

Hence, in all cases, $|f_{cov}(\mathcal{P}_A \bigcup \{p'\}) - f_{cov}(\mathcal{P}_A)| \geq |f_{cov}(\mathcal{P}_B \bigcup \{p'\}) - f_{cov}(\mathcal{P}_B)|$ and $f_{cov}(.)$ is submodular. □

LEMMA 7.4. *The similarity (resp. cognitive load) of a pattern set $\mathcal{P}$, $f_{sim}(\mathcal{P})$ (resp. $f_{cog}(\mathcal{P})$), is supermodular.*

PROOF. Given a submodular function $f(.)$, for every $\mathcal{P}_A \subseteq \mathcal{P}_B \subseteq D$ and every $p \subset D$ s.t $p \notin \mathcal{P}_A, \mathcal{P}_B$, the first order difference states that $f(\mathcal{P}_A \bigcup \{p\}) - f(\mathcal{P}_A) \geq f(\mathcal{P}_B \bigcup \{p\}) - f(\mathcal{P}_B)$. Given a graph G, a canned pattern $p \notin \mathcal{P}_B$ and canned pattern sets $\mathcal{P}_A$ and $\mathcal{P}_B$

where $\mathcal{P}_A \subseteq \mathcal{P}_B$, let the similarity of $\mathcal{P}_A$ and $\mathcal{P}_B$ be $f_{sim}(\mathcal{P}_A)$ and $f_{sim}(\mathcal{P}_B)$, respectively. $f_{sim}(\mathcal{P}_B \bigcup \{p\}) - f_{sim}(\mathcal{P}_B) = \sum_{p_i \in \mathcal{P}_B} sim(p, p_i)$ and $f_{sim}(\mathcal{P}_A \bigcup \{p\}) - f_{sim}(\mathcal{P}_A) = \sum_{p_i \in \mathcal{P}_A} sim(p, p_i)$. Since $sim(p_i, p_j) \geq 0\ \forall p_i, p_j \subset G$, $\mathcal{P}_A \subseteq \mathcal{P}_B$ and by definition of the first order difference, $f_{sim}(.)$ is supermodular.

The proof is similar for $f_{cog}(.)$. □

Theorem 7.5. *The pattern set score $s(\mathcal{P}')$ in Definition 7.1 is a non-negative and non-monotone submodular function.*

Proof. (Sketch). Consider a partial pattern set $\mathcal{P}'$ and a candidate pattern p. Suppose p does not improve the set coverage of $\mathcal{P}'$ and adds a high cost in terms of cognitive load and diversity. Then, $s(\mathcal{P}') > s(\mathcal{P}' \bigcup \{p\})$. Hence, $s(.)$ is *non-monotone*. Since $f_{cov}(\mathcal{P}'), f_{sim}(\mathcal{P}'), f_{cog}(\mathcal{P}') \in [0, |\mathcal{P}'|]$, $f_{cov}(\mathcal{P}') - f_{sim}(\mathcal{P}') - f_{cog}(\mathcal{P}')$ is in the range [-2|$\mathcal{P}'$|,|$\mathcal{P}'$|]. Hence, $\frac{1}{3|\mathcal{P}'|}(f_{cov}(\mathcal{P}') - f_{sim}(\mathcal{P}') - f_{cog}(\mathcal{P}') + 2|\mathcal{P}'|)$ is in the range [0, 1] and is *non-negative*. Since supermodular functions are negations of submodular functions and non-negative weighted sum of submodular functions preserve submodular property [18], $s(\mathcal{P}')$ is submodular. Note that adding a positive constant (*i.e.,* $\frac{2}{3}$) does not change the submodular property [11] and ensures that $s(\mathcal{P}')$ is non-negative. The scaling factors of $\alpha_{f_{cov}} = \alpha_{f_{sim}} = \alpha_{f_{cog}} = \frac{1}{3|\mathcal{P}'|}$ further bound $s(\mathcal{P}')$ within the range $[0, 1]$. □

Similar to $s(T')$ in TFP, $s(\mathcal{P}')$ in CPS is non-negative and non-monotone. However, unlike TFP, CPS imposes a cardinality constraint where $|\mathcal{P}|$ is at most γ. Thus, CPS can be posed instead as a maximization of submodular function problem subject to cardinality constraint [13].

7.2 Coverage, Cognitive Load, and Similarity

Next, we quantify the coverage, cognitive load, and similarity measures used in the pattern score $s(P')$.

Coverage. Recall from Section 4, we can compute the coverage of a pattern p as $cov_p = |\bigcup_{i \in |S(p)|} E_i|$. Since the edge sets of $G_T = (V_T, E_T)$ and $G_O = (V_O, E_O)$ are mutually exclusive, we further modify cov_p to include a weight factor to account for effects exerted by the sizes of G_T and G_O. Specifically, $cov_p = |\bigcup_{i \in |S(p)|} E_i| \frac{|G_x|}{|E|}$ where $G_x \in \{G_T, G_O\}$ for patterns obtained from G_x. However, exact computation of coverage for each candidate pattern is prohibitively expensive. Hence, we approximate cov_p as follows: $cov_{ub(p)} = |E_p| \times freq(p) \times \frac{|G_x|}{|E|}$. Observe that $cov_{ub(p)}$ is in fact the upper bound of cov_p when no isomorphic instances of p in G overlap. Any superior upper bound that can be computed efficiently can be incorporated. Unlike cov_p, computation of $cov_{ub(p)}$ requires only $freq(p)$, which is significantly more efficient.

The order of pattern extraction in G_O (*e.g.*, extracting stars and asterisms before small patterns) may affect the frequency of the extracted patterns. Hence, *normalization* of cov_{ub} is performed for each class of patterns (k-CP, CCP, star, asterism, and small pattern) as follows:

$$cov_{ub(p)} = \frac{cov'_{ub(p)} - Min(cov'_{ub}(P_t)) + 1}{Max(cov'_{ub}(P_t)) - Min(cov'_{ub}(P_t)) + 1} \quad (2)$$

where $t \in \{k-CP, CCP, star, asterism, small\}$ represents a class of pattern. Specifically, we compute k-CPs and CCPs in G_T. Stars, asterisms and small patterns are computed in G_O. The normalized cov_{ub} is in [0-1].

Cognitive Load. [23, 24] measure cognitive load based on size and density only, ignoring edge crossings. Since it is designed for a collection of small- or medium-sized data graphs, it is a reasonable measure as in many applications such data graphs have very few edge crossings (*e.g.*, chemical compounds), if any. In contrast, edge crossings occur frequently in large networks and hence cannot be ignored in our context. In fact, Huang and colleagues examined the effect of edge crossings on mental load of users and found that cognitive load displays a relationship with edge crossings that resembles the logistic curve [21] $f(x) = \frac{L}{1+e^{-k(x-x_0)}}$ where L is the curve's maximum value, x_0 is the x value of sigmoid's midpoint and k is the logistic growth rate [43].

Lemma 7.6. *The crossing number (*i.e., *number of edge crossings) of any simple graph $G = (V, E)$ with at least 3 vertices satisfies $cr \geq |E| - 3|V| + 6$.*

Proof. (Sketch) Consider a graph $G = (V, E)$ with cr crossings. Since each crossing can be removed by removing an edge from G, a graph with $|E| - cr$ edges and $|V|$ vertices contains no crossings (*i.e.,* planar graph). Since $|E| \leq 3|V| - 6$ for the planar graph (*i.e.,* Euler's formula), hence, $|E| - cr \leq 3|V| - 6$ for $|V| \geq 3$. Rewriting the inequality, we have $cr \geq |E| - 3|V| + 6$. □

Hence cognitive load of a pattern p is computed based on the size ($sz_p = |E_p|$), density ($d_p = 2\frac{|E_p|}{|V_p|(|V_p|-1)}$) and edge crossing ($cr_p$). $cr_p = 0$ if p is planar. Otherwise, it is $cr_p = |E_p| - 3|V_p| + 6$. We modelled the normalized cognitive load function in TATTOO according to the logistic curve:

$$cog_p = 1/(1 + e^{-0.5 \times (sz_p + d_p + cr_p - 10)}) \quad (3)$$

Parameters of cog_p are set empirically to ensure even distribution within the range of [0 1].

Similarity. Given a partial pattern set $\mathcal{P}'$ and two candidate patterns p_1 and p_2, TATTOO selects p_1 preferentially to add to $\mathcal{P}'$ if $\max_{p \in \mathcal{P}'} sim(p_1, p) < \max_{p \in \mathcal{P}'} sim(p_2, p)$. To this end, we utilize *NetSimile*, a size-independent graph similarity approach based on distance between feature vectors [8]. It is scalable with runtime complexity linear to the number of edges.

7.3 CPS-Randomized Greedy Algorithm

The canned pattern selection algorithm is as follows. First, it retrieves the default pattern set (1-path, 2-path, 3-cycle and 4-cycle). Next, it prunes candidate patterns whose sizes do not satisfy the plug specification or are "nearly-unique" (*i.e.,* $freq(p) < \delta$ where δ is a pre-defined threshold). Note that the latter patterns have very low occurrences in G and are unlikely to be as useful for query construction in their entirety[3]. Then, it selects $\mathcal{P}$ from the remaining candidates.

Recall from Section 7.1, the CPS problem can be cast as a maximization of submodular function problem subject to cardinality

[3]In the case, a user is interested in patterns with low coverage, δ can be set to 0 along with the reduction in $\alpha_{f_{cov}}(P')$ in $s(P')$ (Defn. 7.1).

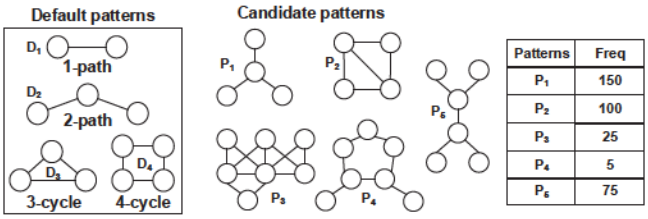

Figure 7: Default patterns and candidate patterns.

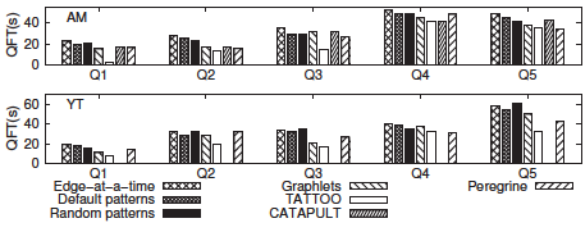

Figure 8: Query formulation time in user study.

constraint. Recently, the algorithm community has proposed a technique with quality guarantee in [13] to address it. We exploit this approach, referred to as *CPS-Randomized Greedy* (CPS-R-Greedy), in our CPS problem. To the best of our knowledge, this approach has not been utilized for graph querying.

In particular, CPS-R-Greedy extends the discrete greedy algorithm [30] using a randomized approach. At every step, a random candidate pattern is chosen from a set of "reasonably good" candidates. Intuitively, these candidates should have very few edge crossings, good coverage and are different from patterns already in $\mathcal{P}$. These candidates are identified as follows. For every candidate pattern p, we compute the pattern set score (Definition 7.1) assuming p is added to the canned pattern set. A "good" candidate p improves on the score of the set when it is added (Definition 7.2). Note that cov_{ub}, cog, and sim changes as $\mathcal{P}$ changes. Hence, we recompute them at every iteration. Then, we randomly select a "good" candidate and assign it to $\mathcal{P}$. The algorithm terminates either when the set contains the desired number of patterns or when there exists no more good candidates. The following quality guarantee can be derived from [13].

THEOREM 7.7. *CPS-R-Greedy achieves $\frac{1}{e}$-approximation of CPS.*

THEOREM 7.8. *CPS-R-Greedy has worst-case time and space complexity of $O(|P_{cand}|\gamma|V_{max}||V_{max}|!)$ and $O(|P_{cand}|(|V_{max}|+|E_{max}|))$, respectively, where $|V_{max}|$ and $|E_{max}|$ are the number of vertices and edges in the largest candidate pattern.*

Example 7.9. Consider a GUI $\mathbb{I}$ and a plug $b = (3, 11, 6)$. Suppose there are four default patterns and five candidate patterns (*i.e.,* P_{cand}) as depicted in Figure 7. Let $\delta = 10$. The algorithm first removes p_4 since $freq(p_4) < \delta$. Then, for the remaining patterns in P_{cand}, each is considered in turn to be added to $\mathcal{P}$ by exploiting CPS-R-Greedy. It first considers adding p_1 to $\mathcal{P}$ and computes the resulting coverage ($f_{covub(\mathcal{P} \cup p_1)}$), cognitive load ($f_{cog(\mathcal{P} \cup p_1)}$) and similarity ($f_{sim(\mathcal{P} \cup p_1)}$). The pattern set score of $\mathcal{P} \bigcup p_1$ is then computed using Definition 7.1. The scores of the other candidate patterns are computed similarly. Suppose the scores are 0.72, 0.63, 0.54, 0.68 for p_1, p_2, p_3, p_5, respectively. Then, in the first iteration, p_1 is selected (and removed from subsequent iterations) and the current best score s_{best} is updated to 0.72. In the next (*i.e.,* final) iteration, the candidates are again considered in turn to be added to $\mathcal{P}$ and corresponding pattern set scores are computed. However, unlike the first iteration, only those candidates whose scores are greater than s_{best} are considered. Let the scores of p_2, p_3 and p_5 be 0.81, 0.7 and 0.77, respectively. Then, a candidate will be randomly selected from p_2 or p_5. Suppose p_2 is chosen, then the final pattern set is $\{d_1, d_2, d_3, d_4, p_1, p_2\}$. ∎

8 PERFORMANCE STUDY

TATTOO is implemented in C++ with GCC 4.2.1 compiler. We now report the key performance results of TATTOO. Additional results and a case study are discussed in [41]. All experiments are performed on a 64-bit Windows 10 desktop with Intel(R) Core(TM) i7-4770K CPU (3.50GHz) and 16GB RAM.

8.1 Experimental Setup

Datasets. We evaluate TATTOO's performance using 10 large networks (Table 2) from SNAP (http://snap.stanford.edu/data/index.html) containing up to 34.7 million edges.

Algorithms. State-of-the-art GUIs for large networks [31, 32] do not support canned patterns. Hence, we compare TATTOO with the following baselines: (a) *CATAPULT* [23]: We assign same labels to all nodes of a network and partition it into a collection of small- or medium-sized data graphs using METIS [27]. Then the algorithm in [23] is used to select canned patterns. (b) *Use graphlets, frequent subgraphs, random patterns, default patterns, and edge-at-a-time* (i.e., *pattern oblivious*): *x-node graphlets* where $x \in [2-5]$ are generated using the approach in [15]. *Random patterns* are generated by randomly selecting subgraphs of specific sizes from a network. The number of candidates per size follows a uniform distribution. *Frequent subgraphs* are generated using *Peregrine* [26] (downloaded from [3]). These subgraphs are considered as candidates from which the canned patterns are selected using our algorithm in Section 7.3.

Query sets and GUI. We use different query sets for the user study and automated performance study. We shall elaborate on them in respective sections. The GUI used for user study is viewable at https://youtu.be/sL0yHV1eEPw.

Parameter settings. Unless specified otherwise, we set $\eta_{min} = 3$, $\eta_{max} = 15$, $\gamma = 30$, $\delta = 3$, and $\epsilon = 5$.

Performance measures. We measure the performance of TATTOO using the followings: (1) *Run time*: Execution time of TATTOO. (2) *Memory requirement* (MR): Peak memory usage when executing TATTOO. (3) *Reduction ratio* (denoted as μ): Given a subgraph query Q, $\mu = \frac{step_{total} - step_P}{step_{total}}$ where $step_P$ is the *minimum* number of steps required to construct Q when $\mathcal{P}$ is used and $step_{total}$ is the total number of steps needed when *edge-at-a-time* approach is used. Note that the number of steps excludes vertex label assignments which is a constant for a given Q regardless of the approach. For simplicity in automated performance study, we follow the same assumptions in [23]: (1) a canned pattern $p \in \mathcal{P}$ can be used in Q iff $p \subseteq Q$; (2) when multiple patterns are used to construct Q, their corresponding isomorphic subgraphs in Q do not overlap. In the user study, we shall jettison these assumptions by allowing users to modify the canned patterns and no restrictions are imposed (*i.e.,*

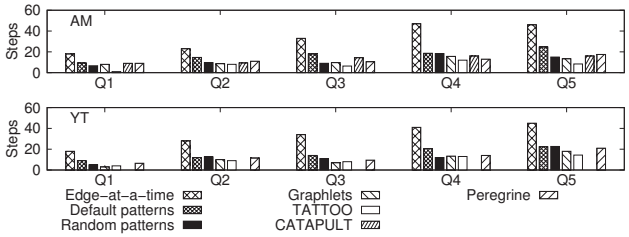

Figure 9: Query construction steps in user study.

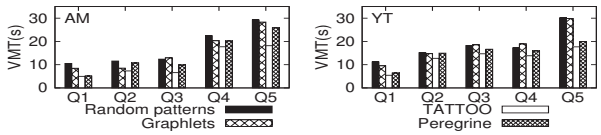

Figure 10: Visual mapping time of canned patterns.

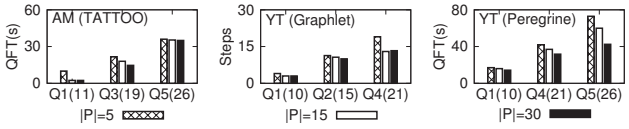

Figure 11: Effect of varying $|P|$ on QFT and steps. Query size is indicated in round brackets.

step$_P$ does not need to be minimum). Smaller values of *div* imply better pattern diversity. For ease of comparison, the diversity plots are based on the inverse of *div*.

8.2 User Study

We undertake a user study to demonstrate the benefits of using our framework from a user's perspective. 27 unpaid volunteers (ages from 20 to 35), who were students of, or, researchers within different majors took part in the user study. None of them has used our GUI prior to the study. First, we presented a 10-min scripted tutorial of our GUI describing how to visually formulate queries. Then, we allowed the subjects to play with the tool for 15 min.

For each dataset, 5 subgraph queries with size in the range [10-28] are selected. These queries mimic topology of real-world queries containing various structures described in Section 5.2. To describe the queries to the participants, we provided printed visual subgraph queries. A subject then draws the given query using a mouse in our GUI. The users are asked to make maximum use of the patterns to this end. Each query was formulated 5 times by different participants. We ensure the same query set is constructed in a random order (the order of the query and the approach are randomized) to counterbalance learning effects (see [41] for details).

The canned patterns on the GUI are grouped by size and displayed using *ForceAtlas2* layout [25] in different pages according to their sizes. This multi-page-based organization yields faster average query formulation time and fewer steps compared to other alternatives (see [41] for details).

Visual mapping time. In order to use canned patterns for query formulation, a user needs to browse the pattern set and visually map them to her query. We refer to this as *visual mapping time* (VMT). For each pattern used, we record the *pattern mapping time* (PMT) as the duration when the mouse cursor is in the *Pattern Panel* to the time a user selects and drags it to the *Query Canvas*. The

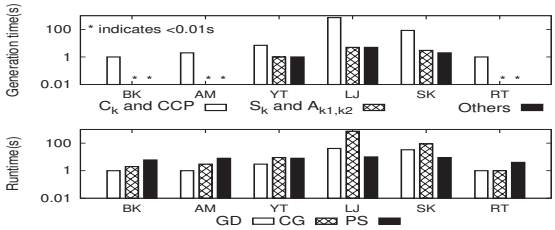

Figure 12: Run time. GD, CG and PS represent truss-based graph decomposition, candidate generation and pattern selection, respectively.

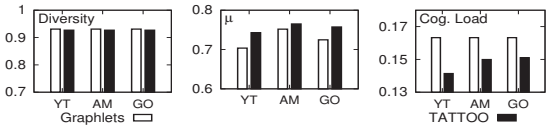

Figure 13: TATTOO vs graphlet patterns.

VMT of a query is its average PMT. Intuitively, a longer VMT implies greater cognitive load on a user. Figure 10 shows the VMT of TATTOO patterns, graphlets, frequent subgraphs, and random patterns on AM and YT datasets. On average, TATTOO patterns consume the least VMT.

Query formulation time (QFT) and number of steps. Figures 8 and 9 plot the average QFT and the average number of steps taken, respectively, for AM and YT. Note that a QFT includes the VMT and the steps include addition/deletion of nodes and edges and merger of nodes. As expected, the edge-at-a-time approach took the most steps. Paired t-test shows that the superior performance of TATTOO is statistically significant ($p < 0.05$) for 79.4% of the comparisons (see [41] for details). In particular, it takes up to 18X, 9.3X, 6.7X, 8X, 9X, and 9X fewer steps compared to edge-at-a-time, default pattern, random patterns, graphlet, frequent patterns, and CATAPULT-generated patterns, respectively. For QFT, TATTOO is up to 9.7X, 8.6X, 9X, 6.6X, 7.1X, and 7.4X faster, respectively. The results are qualitatively similar in other datasets. Note that we can run CATAPULT only on AM for reasons discussed later.

Effect of $|\mathcal{P}|$. The number of patterns on a GUI may also impact a user cognitively as larger $|\mathcal{P}|$ means a user needs to browse more patterns to select relevant ones. Hence, we investigate the effect of $|\mathcal{P}|$ on QFT and the number of steps (Figure 11). Interestingly, QFT and steps are reduced by average of 12% and 22% (maximum reduction of 77% and 80%), respectively, when $|\mathcal{P}|$ is increased from 5 to 30. Increase in $|\mathcal{P}|$ exposes more patterns that could be leveraged for query formulation, reducing query formulation steps. Further, it results in two opposing effects: (1) longer time needed to browse and select appropriate patterns (longer VMT) and (2) potentially more and larger patterns available for query construction resulting in fewer construction steps and shorter QFT. The latter effect dominates.

8.3 Automated Performance Study

In this section, we evaluate TATTOO from the following perspectives. First, we compare the runtime and quality of patterns of TATTOO with the baseline approaches (*Exp 1, 2*). Second, we present results that support some of our design decisions (*Exp 3, 4*). To this end, we generate **1000** queries (size [4-30]) for each dataset where 500

are randomly generated and remaining ones (evenly distributed) are path-like, tree, star-like, cycle-like and flower-like queries.

Exp 1: Run time. First, we evaluate the generation time of different patterns types in canned pattern sets. Figure 12 (top) shows the results. In particular, generation of chord-like patterns requires significantly more time (up to 146% more for LJ) than other pattern types. This is primarily due to checks for different types of edge merger required for CCPs. Figure 12 (bottom) reports the time taken by various phases of TATTOO as well as runtime of CATAPULT. *TATTOO selects canned patterns efficiently within a few minutes.* Observe that the time cost for the small pattern extraction phase is small in practice. In general, pattern selection is the most expensive phase and requires a couple of minutes or less. Results are qualitatively similar for other datasets. Memory usage is reported in [41].

Lastly, observe that TATTOO is 735X faster than CATAPULT, which is not designed for large networks. Except AM, other datasets either cannot be processed by METIS or fail to generate patterns in a reasonable time (within 12 hrs) due to too many possible matches of unlabelled graphs that require expensive graph edit distance computation. In the sequel, we shall omit discussions on CATAPULT.

Exp 2: Comparison with graphlets and frequent subgraphs. Next, we compare TATTOO's patterns with those of graphlets (30 patterns derived from graphlets). Figure 13 reports the results. Observe that TATTOO's patterns are superior to graphlets in all aspects. The results are qualitatively similar for other datasets. Note that coverage is not examined since it is 100% in all cases as all queries can be constructed using a 2-node graphlet.

We compare the canned pattern set derived from frequent subgraphs generated by *Peregrine* (denoted as $\mathcal{P}_P$) to those generated by TATTOO. We observe that *Peregrine* failed to extract larger size patterns (*i.e.,* $|V| \geq 8$) within 12 hrs for all networks. Specifically, for RP, RC, and RT (resp. AM), it was able to extract frequent patterns of size $|V| \leq 7$ (resp. $|V| \leq 6$) within 2.5 hrs. For BK and DB (resp. YT, LJ, SK, and GO) it can extract up to size $|V| \leq 5$ (resp. $|V| \leq 4$) within 2.5hrs. However, it took around 39 hrs on AM to yield a meaningful number of candidate patterns (994 patterns with $|V| \leq 7$ and $|E| \leq 21$) when the minimum threshold is set to 100. *Hence, TATTOO is orders of magnitude faster than frequent pattern-based solution.* Consequently, we restrict the canned pattern sets of both TATTOO and $\mathcal{P}_P$ to 30 patterns with $|V| \leq 7$ and $|E| \leq 21$ for AM in our experiments for fair comparison. Consistent with our user study, TATTOO's pattern set is superior to $\mathcal{P}_P$ in most aspects. The average coverage, cognitive load, diversity and μ for TATTOO (resp. *Peregrine*) are 0.3 (resp. 0.27), 0.15 (resp. 0.14), 0.64 (resp. 0.59) and 0.23 (resp. 0.24), respectively. In summary, TATTOO generates better quality canned patterns.

Exp 3: Measuring cognitive load. We now justify the choice of our proposed cognitive load measure. Specifically, we compare several ways of measuring cognitive load of a pattern p, namely, $f_{cog1} = \frac{1}{3}\sum_{x\in\{sz_p,d_p,cr_p\}}(1-e^{-x})$; $f_{cog2} = 1/(1+e^{-0.5\times(sz_p+d_p+cr_p-10)})$; $f_{cog3} = sz_p + d_p + cr_p$; $f_{cog4} = sz_p \times d_p$ (used in [23]); and $f_{cog5} = cr_p$ (recall sz_p, d_p, cr_p from Section 7.2). 20 volunteers were asked to rank the visual representations of six graphs (Figure 14) of varying sizes and topology, in terms of cognitive effort required to interpret these graphs. A "ground truth" ranking for these graphs is obtained based on the average ranks assigned by the volunteers.

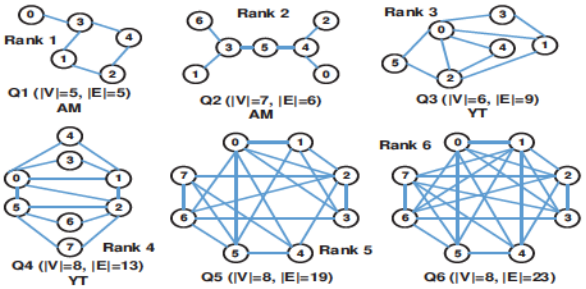

Figure 14: Graphs used for assessing cognitive load.

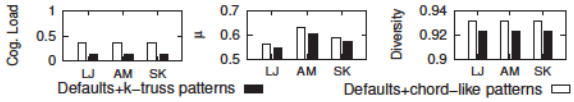

Figure 15: Chord-like patterns vs k-trusses.

Then, the graphs are ranked according to the five cognitive load measures and compared against the ground truth using Kendall's τ [28]. f_{cog2} and f_{cog3} achieve the highest $\tau = 1$. We select f_{cog2} as the cognitive load measure since it is in the range of $[0, 1]$ and facilitates easy formulation of a non-negative and non-monotone submodular pattern score function (Theorem 7.5).

Exp 4: Chord patterns vs k-trusses. Lastly, we show the benefits of using k-CP/CCPs (*i.e.,* k-truss-like structures) compared to simply utilizing k-trusses as topology for canned patterns (recall from Section 5.2). We generate 100 random queries of size [4-30] from G_T and these yielded 11 k-CP/CCPs and 3 k-trusses. Observe that k-CP/CCPs improve both μ and diversity but have poorer cognitive load (Figure 15). Here the cognitive load and diversity of a pattern set is the average value for respective measures. Importantly, more k-CP/CCPs than k-trusses satisfying the plug are generated due to relaxed structure of the former. For instance, the RP dataset produces 266.67% more k-CP/CCPs due to the small size of G_T (see Table 2). That is, k-trusses may not result in sufficient number of canned patterns on a GUI. Hence, chord patterns improve the quality of canned patterns in terms of μ and diversity compared to k-trusses and yield more candidate patterns.

9 CONCLUSIONS & FUTURE WORK

Canned patterns play a pivotal role in supporting efficient visual subgraph query formulation using direct-manipulation interfaces. We present TATTOO, which takes a data-driven approach to selecting them from the underlying network by exploiting real-world query characteristics and optimizing coverage, diversity, and cognitive load of the patterns. Our experimental study demonstrates superiority of our framework to several baselines. As part of future work, we plan to explore the problem in a distributed settings.

ACKNOWLEDGMENTS

The first four authors are supported by the AcRF Tier-2 Grant MOE2015-T2-1-040. Wook-Shin Han was supported by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2018-0-01398). Byron Choi is supported by HKBU12201518.

REFERENCES

[1] 2021. Neo4j Bloom. https://neo4j.com/bloom.

[2] 2021. BSBM. http://wifo5-03.informatik.uni-mannheim.de/bizer/berlinsparqlbenchmark/spec/20080912/index.html#queriesTriple.

[3] 2021. Peregrine.https://github.com/pdclab/peregrine.

[4] 2021. RAPID. https://research.csc.ncsu.edu/coul/RAPID/RAPIDAnalytics/.

[5] 2021. Stanford Large Network Dataset Collection. http://snap.stanford.edu/data/index.html.

[6] Foto N. Afrati, Dimitris Fotakis and Jeffrey D. Ullman. 2013. Enumerating subgraph instances using map-reduce. In *IEEE 29th International Conference on Data Engineering*. IEEE, 62-73.

[7] Nesreen K. Ahmed, Jennifer Neville, Ryan A. Rossi and Nick Duffield. 2015. Efficient graphlet counting for large networks. In *2015 IEEE International Conference on Data Engineering*. IEEE, 1-10.

[8] Michele Berlingerio, Danai Koutra, Tina Eliassi-Rad and Christos Faloutsos. 2013. Network similarity via multiple social theories. In *Proceedings of the 2013 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining*. IEEE/ACM, 1439-1440.

[9] Sourav S. Bhowmick, Byron Choi and Curtis E. Dyreson. 2016. Data-driven visual graph query interface construction and maintenance: challenges and opportunities. *Proceedings of the VLDB Endowment* 9, 12 (2016), 984-992.

[10] Sourav S. Bhowmick, Kai Huang, Huey Eng Chua, Zifeng Yuan, Byron Choi and Shuigeng Zhou. 2020. AURORA: Data-driven construction of visual graph query interfaces for graph databases. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. ACM, 2689-2692.

[11] Avradeep Bhowmik, Vivek Borkar, Dinesh Garg and Madhavan Pallan. 2014. Submodularity in team formation problem. In *Proceedings of the 2014 SIAM International Conference on Data Mining*. SIAM, 893-901.

[12] Angela Bonifati, Wim Martens and Thomas Timm. 2017. An analytical study of large sparql query logs. *Proceedings of the VLDB Endowment* 11, 2 (2017), 149-161.

[13] Niv Buchbinder, Moran Feldman, Joseph Naor and Roy Schwartz. 2014. Submodular maximization with cardinality constraints. In *Proceedings of the 2014 Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 1433-1452.

[14] Shi-Jie Chen and Li Lin. 2004. Modeling team member characteristics for the formation of a multifunctional team in concurrent engineering. *IEEE Transactions on Engineering Management* 51, 2 (2004), 111-124.

[15] Xiaowei Chen, Yongkun Li, Pinghui Wang and John C.S. Lui. 2016. A general framework for estimating graphlet statistics via random walk. *Proceedings of the VLDB Endowment* 10, 3 (2016), 253-264.

[16] Aarzoo Dhiman and S.K. Jain. 2016. Frequent subgraph mining algorithms for single large graphs - A brief survey. In *2016 International Conference on Advances in Computing, Communication, & Automation*. IEEE, 1-6.

[17] Basil Ell, Denny Vrandečić and Elena Simperl. 2011. Deriving human-readable labels from SPARQL queries. In *Proceedings of the 7th International Conference on Semantic Systems*. ACM, 126-133.

[18] Satoru Fujishige. 2005. *Submodular functions and optimization (2nd edition)*. Elsevier B.V., Amsterdam, The Netherlands.

[19] Saket Gurukar, Sayan Ranu and Balaraman Ravindran. 2015. Commit: A scalable approach to mining communication motifs from dynamic networks. In *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 475-489.

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

[21] Weidong Huang and Maolin Huang. 2010. Exploring the relative importance of crossing number and crossing angle. In *Proceedings of the 3rd International Symposium on Visual Information Communication*. ACM, 1-8.

[22] Weidong Huang, Peter Eades and Seok-Hee Hong. 2009. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Information Visualization* 8, 3 (2009), 139-152.

[23] Kai Huang, Huey Eng Chua, Sourav S. Bhowmick, Byron Choi and Shuigeng Zhou. 2019. CATAPULT: Data-driven selection of canned patterns for efficient visual graph query formulation. In *Proceedings of the 2019 International Conference on Management of Data*. ACM, 900-917.

[24] Kai Huang, Huey Eng Chua, Sourav S. Bhowmick, Byron Choi and Shuigeng Zhou. 2021, MIDAS: towards efficient and effective maintenance of canned patterns in visual graph query interfaces. In *Proceedings of the 2021 International Conference on Management of Data*. ACM, 764-776.

[25] Mathieu Jacomy, Tommaso Venturini, Sebastien Heymann and Mathieu Bastian. 2014. ForceAtlas2, a continuous graph layout algorithm for handy network visualization designed for the Gephi software. *PLOS ONE* 9, 6 (2014), e98679.

[26] Kasra Jamshidi, Rakesh Mahadasa and Keval Vora. 2020. Peregrine: a pattern-aware graph mining system. In *Proceedings of the Fifteenth European Conference on Computer Systems*. ACM, 1-6.

[27] George Karypis and Vipin Kumar. 1997. *METIS: A software package for partitioning unstructured graphs, partitioning meshes, and computing fill-reducing orderings of sparse matrices*. Technical Report. University of Minnesota.

[28] Maurice George Kendall. 1948. *Rank correlation methods*. Griffin.

[29] Ron Milo, Shai Shen-Orr, Shalev Itzkovitz, Nadav Kashtan, Dmitri Chklovskii and Uri Alon. 2002. Network motifs: Simple building blocks of complex networks. *Science* 298, 5594 (2002), 824-827.

[30] George L. Nemhauser, Laurence A. Wolsey and Marshall L. Fisher. 1978. An analysis of approximations for maximizing submodular set functions-I. *Mathematical Programming* 14, 1 (1978), 265-294.

[31] Robert Pienta, Fred Hohman, Acar Tamersoy, Alex Endert, Shamkant Navathe, Hanghang Tong and Duen Horng Chau. 2017. Visual graph query construction and refinement. In *Proceedings of the 2017 ACM International Conference on Management of Data*. ACM, 1587-1590.

[32] Robert Pienta, Fred Hohman, Alex Endert, Acar Tamersoy, Kevin Roundy, Chris Gates, Shamkant Navathe and Duen Horng Chau. 2018. VIGOR: Interactive visual exploration of graph query results. *IEEE Transactions on Visualization and Computer Graphics* 24, 1 (2018), 215-225.

[33] Natasa Pržulj, Derek G. Corneil and Igor Jurisica. 2004. Modeling interactome: Scale-free or geometric? *Bioinformatics* 20, 18 (2004), 3508-3515.

[34] Siddhartha Sahu, Amine Mhedhbi, Semih Salihoglu, Jimmy Lin and M. Tamer Özsu. 2017. The ubiquity of large graphs and surprising challenges of graph processing. *Proceedings of the VLDB Endowment* 11, 4 (2017), 420-431.

[35] Muhammad Saleem, Ali Hasnain and Axel-Cyrille Ngonga Ngomo. 2018. LargeRDFBench: A billion triples benchmark for sparql endpoint federation. *Journal of Web Semantics* 48, 85-125.

[36] Ben Shneiderman and Catherine Plaisant. 2010. *Designing the user interface: Strategies for effective human-computer interaction (5th edition)*. Addison-Wesley, Boston, M.A.

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

[38] Chad Voegele, Yi-Shan Lu, Sreepathi Pai and Keshav Pingali. 2017. Parallel triangle counting and k-truss identification using graph-centric methods. In *2017 IEEE High Performance Extreme Computing Conference*. IEEE, 1-7.

[39] Jia Wang and James Cheng. 2012. Truss decomposition in massive networks. *Proceedings of the VLDB Endowment* 5, 9 (2012), 812-823.

[40] Xiao Fan Wang and Guanrong Chen. 2003. Complex networks: Small-world, scale-free and beyond. *IEEE Circuits and Systems Magazine* 3, 1 (2003), 6-20.

[41] Zifeng Yuan, Huey Eng Chua, Sourav S. Bhowmick, Zekun Ye, Wook-Shin Han and Byron Choi. 2021. *Towards plug-and-play visual graph query interfaces: data-driven canned pattern selection for large networks*. Technical Report. Nanyang Technological University. Available at: http://arxiv.org/abs/2107.09952

[42] Vahan Yoghourdjian, Daniel Archambault, Stephan Diehl, Tim Dwyer, Karsten Klein, Helen C. Purchase and Hsiang-Yun Wu. 2018. Exploring the limits of complexity: a survey of empirical studies on graph visualization. *Visual Informatics* 2, 4 (2018), 264-282.

[43] Boris Zeide. 1993. Analysis of growth equations. *Forest Science* 39, 3 (1993), 594-616.

[44] Jinbo Zhang, Sourav S. Bhowmick, Hong H. Nguyen, Byron Choi and Feida Zhu. 2015. DaVinci: Data-driven visual interface construction for subgraph search in graph databases. In *2015 IEEE 31st International Conference on Data Engineering*. IEEE, 1500-1503.