MIDAS: Towards Efficient and Effective Maintenance of Canned Patterns in Visual Graph Query Interfaces

Kai Huang [§,‡] Huey Eng Chua[‡] Sourav S Bhowmick[‡] Byron Choi[†] Shuigeng Zhou[§]

[‡]School of Computer Science and Engineering, Nanyang Technological University, Singapore
[§]Shanghai Key Lab of Intelligent Information Processing, Sch. of Computer Science, Fudan University, China
[†]Department of Computer Science, Hong Kong Baptist University, Hong Kong SAR
hechua|assourav@ntu.edu.sg,bchoi@comp.hkbu.edu.hk,khuang14|sgzhou@fudan.edu.cn

ABSTRACT

Several visual graph query interfaces (a.k.a GUI) expose a set of *canned patterns* (*i.e.,* small subgraph patterns) to expedite subgraph query formulation by enabling *pattern-at-a-time* construction. Unfortunately, manual generation of canned patterns is not only labour intensive but also may lack diversity to support efficient visual formulation of a wide range of subgraph queries. Recent efforts have taken a *data-driven* approach to select high-quality canned patterns for a GUI automatically from the underlying graph database. However, as the underlying database evolves, these selected patterns may become stale and adversely impact efficient query formulation. In this paper, we present a novel framework called MIDAS for efficient and effective maintenance of the canned patterns as the database evolves. Specifically, it adopts a *selective maintenance strategy* that guarantees progressive gain of *coverage* of the patterns without sacrificing their *diversity* and *cognitive load*. Experimental study with real-world datasets and visual graph interfaces demonstrates the effectiveness of MIDAS compared to static GUIs.

ACM Reference Format:
Kai Huang, Huey Eng Chua, Sourav S Bhowmick, Byron Choi, 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 (SIGMOD '21), June 20–June 25, 2021, Virtual Event, China. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3448016.3457251

1 INTRODUCTION

Visual graph query interfaces (a.k.a GUI) for interactive construction of subgraph queries encourage non-programmers to take advantage of graph querying frameworks. Several commercial querying frameworks (*e.g., PubChem* [5], *Drugbank* [2]) for querying a large collection of small- or medium-sized data graphs (*i.e.,* graph database) provide *direct-manipulation* interfaces [36] for visual query formulation. Specifically, they expose a set of *canned patterns* (*i.e.,* small subgraph patterns), which is beneficial to visual querying in at least three possible ways. First, they can potentially decrease the time taken to visually construct a query. Specifically, a canned pattern (pattern for brevity) enables a user to construct multiple nodes and edges in a subgraph query by performing a *single* click-and-drag action (*i.e., pattern-at-a-time* mode) in lieu of iterative construction of edges one-at-a-time (*i.e., edge-at-a-time* mode). Second, they can facilitate "bottom-up" search when a user does not have upfront knowledge of what to search for. She observes the key patterns that exist in a dataset through a diverse set of canned patterns that may provoke further inquiries. Third, HCI research shows that users may become frustrated if a large number of small atomic actions (*e.g.,* repeated edge construction) is necessary to accomplish a higher-level task (*e.g.,* subgraph query) [36]. Naturally, patterns may ease such frustration.

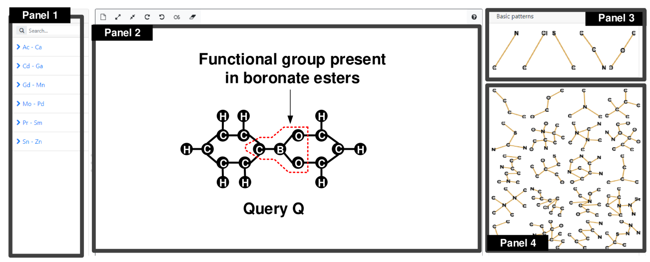

Figure 1: A GUI.

Example 1.1. Consider the GUI in Figure 1 to query a chemical compound database (*e.g., PubChem*). Figure 2(a) depicts examples of patterns of size 3 or larger in it (Panel 4) to facilitate visual query formulation. Suppose John, a chemist, constructs a query graph of *boronic acid* (Panel 2). In pattern-at-a-time mode, it would take him 20 steps (102 sec) by dragging-and-dropping relevant patterns and editing them if necessary. In particular, John uses p_4 and p_1 patterns; removes a H and its associated edge from p_4; add 7 vertices (3 H, 1 C, 1 B and 2 O); and 10 edges. On the other hand, if he had constructed it using the edge-at-a-time mode, it would have consumed 41 steps (145 sec). Consequently, it is paramount to *select* the canned pattern set judiciously so that it can support efficient visual formulation of a *large* number of different subgraph queries.

Observe that John may not necessarily have the complete query structure "in his head" during query formulation. He may find p_4 interesting while browsing the pattern set, which may initiate his bottom-up search for boronic acid. It is worth noting that without the existence of a pattern set, such bottom-up search is infeasible. ∎

SIGMOD '21, June 20–25, 2021, Virtual Event, China

ACM ISBN 978-1-4503-8343-1/21/06…$15.00
https://doi.org/10.1145/3448016.3457251

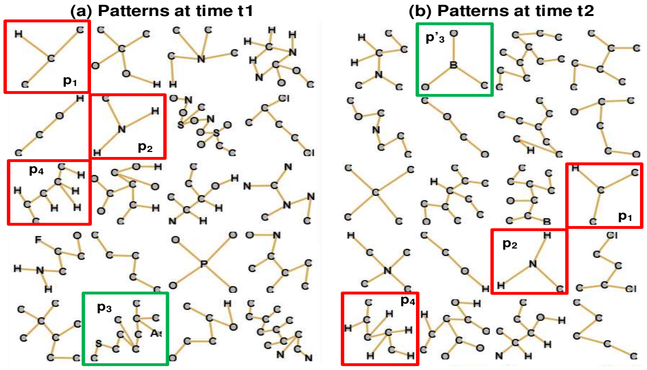

Figure 2: Canned pattern sets.

Manual selection of canned patterns is not only labour-intensive but the selected patterns may not be *diverse* enough to expedite formulation of a wide range of subgraph queries [12]. Catapult [23] is the first effort that systematically selects canned patterns in a *data-driven* manner. Given a graph database D and a *pattern budget* (*i.e.*, minimum and maximum size of patterns, number of patterns on the gui), it selects canned patterns that exhibit high *coverage* and high *diversity*. Although coverage of patterns is intuitive, diverse patterns ensure efficient usage of the limited display space on the gui by not displaying very similar patterns. Furthermore, it preferentially selects patterns that have potentially 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 end users as patterns with high load may adversely impact query formulation time (qft) [23]. To this end, Catapult first partitions D into a set of clusters and summarizes each cluster to a *cluster summary graph* (csg). Then, it selects the canned patterns with aforementioned characteristics from these csgs using a weighted random walk approach.

Observe that these patterns are selected from D at a particular time point. However, real-world graph repositories are dynamic in nature. A study [47] reported that approximately 4,000 new structures were added daily to the *SCI finder* database (www.cas.org/products/scifinder). Similarly, new compounds are added to the *PubChem* (pubchemdocs.ncbi.nlm.nih.gov/submissions-getting-started) and *Drugbank* (dev.drugbank.com/guides/faqs) daily. Consequently, these patterns may grow stale quickly over time and adversely impact efficient visual query formulation.

Example 1.2. Reconsider Example 1.1. Chemical compounds are discovered at an exponential rate [28]. Suppose that the patterns are regenerated (Fig. 2(b)) after *PubChem* added a new group of 6375 compounds called *boronic esters* which is characterized by the functional group outlined in Fig. 1. In particular, some patterns (*e.g.,* p_3) have become stale and are replaced by new patterns (*e.g.,* p'_3) relevant to *boronic esters*. John only requires 14 steps (70 sec) now to formulate the query by using p_4, p_1 and p'_3; removing a H vertex and its associated edge from p_4; adding 3 H vertices and 7 edges. That is, the refreshed pattern set led to more efficient formulation compared to its stale version. Also, existence of the new p'_3 pattern may trigger bottom-up search for boronic ester-based compounds that may not be possible if the stale gui is used. ∎

The aforementioned example motivates the need for maintaining the canned pattern set as the underlying graph repository evolves in order to support efficient visual query formulation. Unfortunately, as we shall see in Section 7, the straightforward approach of executing Catapult repeatedly as D evolves to maintain the pattern set can be extremely inefficient. Hence, in this paper we present Midas (**M**a**I**ntenance of canne**D** p**A**ttern**S**) for effective and efficient canned pattern maintenance as the underlying database evolves. It is built on top of Catapult. Specifically, it seeks to update the existing canned patterns $\mathcal{P}$ in a gui $\mathbb{I}$ to $\mathcal{P}'$ due to evolution of D such that $\mathcal{P}'$ continues to have high coverage, high diversity, and low cognitive load. In particular, Midas guarantees that the quality of $\mathcal{P}'$ is at least the same or better than $\mathcal{P}$. We assume the database changes as a *batch* of graph insertions and deletions. This assumption is reasonable as unlike large networks (*e.g.,* social) where data may continuously appear in streaming mode, as remarked above, several real-world databases of small- or medium-sized data graphs are updated periodically (*e.g.,* daily).

The *canned pattern maintenance* (cpm) problem introduces several non-trivial challenges. First, it is NP-hard. Second, not every modification to D demands maintenance of $\mathcal{P}$. Consider a modification involving deletion of a data graph G_1 in a cluster C. Suppose $G_5 \in C$ contains a subgraph isomorphic to G_1. Since the csgs are generated by *integrating* the data graphs in a cluster (*e.g.,* G_1, G_5), there is no change to the csg of C from which canned patterns are selected. Hence, $\mathcal{P}$ does not need to be maintained. On the other side of the spectrum, suppose a large number of data graphs are added to D. Such a modification is likely to cause drastic changes to the clusters and warrants maintenance of $\mathcal{P}$. Third, Catapult utilizes frequent subtrees as feature vectors for clustering. Although this is reasonable when $\mathcal{P}$ is constructed from static data, it makes efficient maintenance of the clusters a challenging task due to the lack of *closure property* (detailed in Section 4.1). Hence, more efficient data structure needs to be considered for the cpm problem.

Midas addresses the aforementioned challenges as follows. It exploits degree of changes to *graphlet frequency* distribution in D to *selectively* maintain $\mathcal{P}$. Second, it replaces frequent subtrees with *frequent closed trees* (fct) [11] as feature vectors for clustering. Intuitively, a frequent tree is *closed* if none of its proper supertrees has the same support as it has. That is, f' is a proper supertree of f if f' is a supertree of f but $f \neq f'$. Importantly, fcts display closure property [11], paving the way for efficient maintenance of the clusters. Third, $\mathcal{P}$ is updated opportunely using a novel *multi-scan swapping strategy* that guarantees progressive gain of coverage without sacrificing diversity and cognitive load. To this end, we leverage on a *coverage-based pruning* strategy and two indices, namely, *frequent closed tree index* (*fct-Index*) and *infrequent edge index* (*ife-Index*), to facilitate pruning of unpromising candidate patterns for selecting new patterns for $\mathcal{P}$. Our experimental study reveals that Midas is up to 80 times faster than maintenance-from-scratch approach. Importantly, it can reduce the number of formulation steps and qft by up to 50% and 42%, respectively, compared to "static" guis.

In summary, this paper makes the following contributions. (a) To the best of our knowledge, we are the first to formally propose the novel *canned pattern maintenance problem* and present a holistic strategy to address it (Section 3). (b) We describe Midas, an end-to-end framework that can support efficient and high quality maintenance of canned patterns in any visual graph query interface independent of domains and data sources (Sections 4-6).

Table 1: List of key notations.

Notation	Description
G, D	a graph, a graph database
$p, \mathcal{P}, \mathcal{P}'$	a pattern, a pattern set, updated pattern set
$b, \eta_{min}/\eta_{max}, \gamma$	pattern budget, min/max pattern size, number of displayed patterns
$scov(), lcov()$	subgraph coverage, label coverage
$cog()$	cognitive load
$GED(), div()$	graph edit distance, diversity
$C, \mathcal{C}$	a graph cluster, a set of graph clusters
$\Delta^+(\Delta^-), \Delta D$	insertion (deletion) of graphs, updates of graphs
FCT, IFE	frequent closed trees, infrequent edges
CSG	closure summary graph
ϵ	evolution ratio threshold
κ, λ	swapping thresholds
PMT	pattern maintenance time
MP, μ	missing percentage, reduction ratio
QFT, VMT	query formulation time, visual mapping time

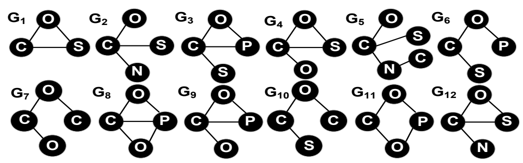

Figure 3: A sample graph database.

(c) Using real-world data graph repositories and GUIs, we show the superiority and applicability of MIDAS in comparison to static GUIs (Section 7). Formal algorithms and proofs of all theorems and lemmas are given in [24].

2 BACKGROUND

In this section, we begin by introducing some graph concepts. Then, we briefly describe the characteristics of canned patterns. Lastly we summarize the CATAPULT framework [23] to address the *canned pattern selection* (CPS) problem. Table 1 lists the key notations and acronyms used in this paper.

2.1 Graph Terminology

Let $G = (V, E)$ be a simple graph where V is a set of vertices and $E \subseteq V \times V$ is a set of edges. We assume that the data graphs and visual subgraph queries are undirected simple graphs with labeled vertices. The label of vertex $v \in V$ is denoted as $l(v)$. The label of an edge (u, v) is given as $l(e) = l(u).l(v)$. The *size* of G is defined as $|G| = |E|$. Given two graphs $G = (V, E)$ and $G' = (V', E')$, G is a *subgraph* of G' if there exists a subgraph isomorphism from G to G' and it is denoted by $G \subseteq G'$. In this work, we focus on a graph database or repository containing a large collection of small- or medium-sized data graphs (denoted as D). A unique *index* (*i.e.,* ID) is assigned to each data graph in D. We denote a data graph with index i as $G_i \in D$. Figure 3 depicts a sample graph database.

2.2 Characteristics of Canned Patterns

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

High coverage. A pattern $p \in \mathcal{P}$ *covers* a data graph $G \in D$ if G contains a subgraph s that is isomorphic to p. In particular, two types of coverage are considered, namely, *subgraph coverage* and *label coverage*. The *subgraph coverage* of a pattern $p = (V_p, E_p)$ is given as $scov(p, D) = |\mathcal{G}_p|/|D|$ where $\mathcal{G}_p \subseteq D$ is a set of data graphs containing p. The *label coverage* of an edge e of D is given as $lcov(e, D) = |L(e, D)|/|D|$ and $L(e, D)$ is the set of graphs in D containing edges having same label as e. Intuitively, a canned pattern set should cover a large number of data graphs in D (*i.e.,* subgraph coverage) and has as many unique vertex labels as possible (*i.e.,* label coverage). Hence, coverage of a canned pattern set $\mathcal{P}$ is given as $f_{scov}(\mathcal{P}) = |\mathcal{G}_{\mathcal{P}}|/|D|$ and $f_{lcov}(\mathcal{P}) = |\bigcup L(e_{\mathcal{P}}, D)|/|D|$ where $\mathcal{G}_{\mathcal{P}} \subseteq D$ is a set of data graphs containing at least one pattern in $\mathcal{P}$ and $e_{\mathcal{P}}$ is an edge in at least one pattern in $\mathcal{P}$.

High diversity. In order to make efficient use of the limited display space on $\mathbb{I}$, every pattern p should ideally be *diverse* from every other pattern in $\mathcal{P}$. This will enable $\mathcal{P}$ to potentially serve a larger variety of queries. Given the patterns p, p_1, and p_2, we say p_1 is more *diverse* from (resp. similar to) p compared to p_2 if $GED(p_1, p) > GED(p_2, p)$ (resp. $GED(p_1, p) < GED(p_2, p)$) where $GED(.)$ is the graph edit distance [32]. Consequently, *diversity* of p, denoted as $div(p, \mathcal{P} \setminus p) = min\{GED(p, p_i)\}$ where $p_i \in \mathcal{P} \setminus p$. Similarly, diversity of $\mathcal{P}$ is given as $f_{div}(\mathcal{P}) = \min_{p \in \mathcal{P}} div(p, \mathcal{P} \setminus p)$.

Low cognitive load. *Cognitive load* refers to the memory demand or mental effort required to perform a given task [25]. A relatively large and complex pattern may demand substantial cognitive effort from an end user to decipher it and to decide if it can aid in her query formulation [13, 23]. Hence, it is desirable for the patterns to impose low cognitive load on the end user. We adopt the measure in [23] to quantify cognitive load of a pattern p: $cog(p) = |E_p| \times \rho_p$ where $\rho_p = 2\frac{|E_p|}{|V_p|(|V_p|-1)}$ is the density of p. This follows from the intuition that the cognitive load increases with density as users tend to spend more time identifying relationship between different vertices in denser graphs [25, 41]. Cognitive load of $\mathcal{P}$ is given as $f_{cog}(\mathcal{P}) = \max_{p \in \mathcal{P}} cog(p)$.

2.3 The CATAPULT Framework

The CATAPULT framework comprises of the following three steps [23].

Small Graph Clustering. A 2-step clustering approach is used to partition D into a set of *graph clusters* $\mathcal{C} = \{C_1, C_2, \ldots, C_k\}$, where $C_i \subseteq D$ and $\forall i \neq j, C_i \cap C_j = \emptyset$. The first step (*coarse clustering*) is a *feature vector-based* approach that uses *frequent subtrees* of D as feature vector for k-means clustering where the k seeds are chosen using the k-means++ algorithm [8]. Since the generated clusters (referred to as *coarse clusters*) may still be large and expensive for generating cluster summary graphs (CSGs), the second step (*fine clustering*) is performed on those coarse clusters that exceed the maximum cluster size threshold N. In particular, it leverages *maximum connected common subgraph* (MCCS) [35] as the clustering property. That is, fine clustering replaces a large coarse cluster with smaller clusters $\mathcal{C}_{fine} = \{C'_1, \cdots, C'_m\}$ where $\forall C'_i \in \mathcal{C}_{fine}, |C'_i| \leq N$, $\omega_{\text{MCCS}}(G_m, G_n) \geq \omega_{\text{MCCS}}(G_m, G_p)$, $G_m, G_n \in C'_i$, $G_p \in C'_j$, $C'_j \in \mathcal{C}_{fine}$ and $i \neq j$, and $\omega_{\text{MCCS}}(G_1, G_2) = \frac{|G_{\text{MCCS}}|}{Min(|G_1|, |G_2|)}$ is MCCS *similarity.*

Cluster Summary Graph (CSG) Generation. CATAPULT summarizes *each* cluster $C_i \in \mathcal{C}$ into a CSG by performing *graph closure*

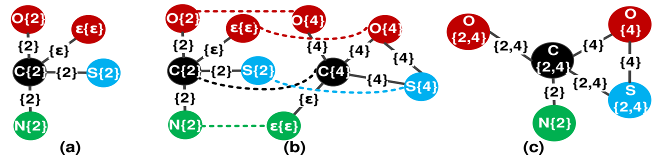

Figure 4: (a) Extended graph of G_2 (Figure 3); (b) mapping of extended graphs of G_2 and G_4; (c) closure graph of G_2 and G_4.

iteratively on pairs of data graphs in the cluster. A *closure graph* [22] *integrates* graphs of varying sizes into a single graph referred to as *extended graph* (denoted by $G^* = (V^*, E^*)$) by inserting dummy vertices or edges with a special label ε such that every vertex and edge is represented in G^*. Given two extended graphs G_1^* and G_2^* and a *mapping* ϕ between them, a *vertex* and an *edge closure* can be obtained by performing an element-wise union of the attribute values of each vertex and each edge in the two graphs, respectively. Then the *closure graph* of G_1^* and G_2^* is a labelled graph $G_c = (V_c, E_c)$ where V_c is the vertex closure of V_1^* and V_2^* and E_c is the edge closure of E_1^* and E_2^*. Note that attribute values ε corresponding to a dummy vertex or edge are removed from G_c. Figure 4 illustrates the notion of closure graphs.

Canned Patterns Selection. Finally, CATAPULT follows a greedy iterative approach based on *weighted random walks* for selecting canned patterns from CSGs. First, each edge in every CSG is assigned a *weight* based on its label coverage in the dataset and in the cluster. In particular, the weight w_e of an edge e is given as $w_e = lcov(e, D) \times lcov(e, C)$ where $lcov(e, X) = |L(e, X)|/|X|$. Next, it performs random walks on these weighted CSGs. Given a weighted CSG S, for each size in the range $[\eta_{min} - \eta_{max}]$ (*i.e.,* pattern budget b), it leverages on the statistics obtained from the random walks to propose a variety of *potential candidate patterns* (PCP) from which a *final candidate pattern* (FCP) is derived. In particular, an FCP of a particular size η_i for a CSG is found by retrieving a connected subgraph of size η_i with the most frequently traversed edges. A *pattern score* is computed for each FCP based on the following definition.

Definition 2.1. Given D with clusters C, a FCP p and a canned pattern set $\mathcal{P}$, the pattern score of p is defined as $s_p = ccov(p, cw, C) \times lcov(p, D) \times \frac{div(p, \mathcal{P} \backslash p)}{cog(p)}$ where $cw_i = \frac{|C_i|}{|D|}$ and $ccov(p, cw, C) = \sum_{i \in C} cw_i \times I_i$ such that $I_i = 1$ if the CSG of C_i contains a subgraph isomorphic to $p \in \mathcal{P}$, otherwise $I_i = 0$.

The candidate pattern with the largest pattern score is selected as the best pattern to be added to $\mathcal{P}$. Weights of the CSGs are then updated using the *multiplicative weights update* approach [7]. These steps are repeated until either the required number of patterns are discovered or when no new pattern can be found.

3 THE CPM PROBLEM

In this section, we first define the *canned pattern maintenance* (CPM) problem. Next, we introduce the technical challenges and our strategies to tackle them. Lastly, we provide an overview of MIDAS.

3.1 Problem Definition

Intuitively, the *canned pattern maintenance* (CPM) problem seeks to update existing canned patterns $\mathcal{P}$ in a GUI $\mathbb{I}$ to $\mathcal{P}'$ due to evolution of D such that $\mathcal{P}'$ continues to have high coverage, high diversity and low cognitive load.

We consider *w.l.o.g* the following *unit updates.*

- graph insertion: insertion of a new data graph in D.
- graph deletion: deletion of a data graph in D.

A batch update ΔD to D is a sequence of unit updates. We denote insertion of a set of data graphs in ΔD as Δ^+ and deletion of a set of data graphs as Δ^-. Also, $D \oplus \Delta D$ denotes a graph database after applying ΔD to D.

*Definition 3.1. The **canned pattern maintenance (CPM) problem** is stated as follows.*

- *Input: A graph database D, a visual graph query interface $\mathbb{I}$ with canned pattern set $\mathcal{P}$, a pattern budget $b = (\eta_{min}, \eta_{max}, \gamma)$ where η_{min} (resp. η_{max}) is the minimum (resp. maximum) size of a pattern and γ is the number of patterns to be displayed on $\mathbb{I}$, and updates ΔD to the input graph database D.*
- *Output: Let $f_{div}(\mathcal{P}')$, $f_{cog}(\mathcal{P}')$, $f_{scov}(\mathcal{P}')$ and $f_{lcov}(\mathcal{P}')$ be the diversity, cognitive load, subgraph and label coverage of $\mathcal{P}'$, respectively. Then, the output is an updated set of canned patterns $\mathcal{P}'$ on $\mathbb{I}$ for modified database $D \oplus \Delta D$ that*

$$\begin{aligned} &\max f_{scov}(\mathcal{P}'), f_{lcov}(\mathcal{P}'), f_{div}(\mathcal{P}'), -f_{cog}(\mathcal{P}') \\ &\textit{subject to } |\mathcal{P}'| = \gamma, \mathcal{P}' \in U \end{aligned} \quad (1)$$

where $\mathcal{P}'$ is the solution; U is the feasible set of canned pattern sets in $D \oplus \Delta D$; $\eta_{min} > 2$; $\lceil \frac{\gamma}{\eta_{max} - \eta_{min} + 1} \rceil$ is the maximum number of patterns for each k-sized pattern; $k \in [\eta_{min}, \eta_{max}]$.

Remark. Observe that CPM is a multi-objective optimization problem where infinite number of Pareto optimal solutions may exist, making it hard to decide on a single suitable solution [29]. Furthermore, it is rare to find a feasible solution that optimizes all objective functions simultaneously. We address this by converting CPM into a single-objective optimization problem using a *multiplicative score function* [37] (detailed in Section 6). Note that here we focus on maintenance of patterns with $\eta_{min} > 2$. The maintenance of patterns with $\eta_{min} \leq 2$ is straightforward and is given in [24].

THEOREM 3.2. *The CPM problem is NP-hard.*

3.2 Design Challenges

Recall that CATAPULT utilizes frequent subtrees as feature vectors for coarse clustering in the small graph clustering phase. Observe that the evolution of D may impact the content of the graph clusters generated by this phase. Unfortunately, frequent subtrees make efficient maintenance of these clusters a challenging task due to the lack of *closure property.* If we utilize frequent subtrees, we need to mine them again from scratch on $D \oplus \Delta D$, which is time-consuming. Note that the closure property of a data structure plays a pivotal role in designing efficient maintenance strategies [11]. Hence, we need a data structure with closure property for the CPM problem.

Second, batch updates to D may result in different degree of evolution of the graph clusters. Naturally, this may impact the structures of CSGs from which patterns are selected. However, as remarked in Section 1, not all modifications to D demand refreshing of existing canned pattern set $\mathcal{P}$ as the updated version should not sacrifice the characteristics of canned patterns w.r.t coverage, diversity and cognitive load. Hence, we need to maintain $\mathcal{P}$ *opportunely.*

3.3 Scaffolding Strategy

We tackle the first challenge using *scaffolding.* In particular, we adapt the existing CATAPULT framework by replacing the frequent subtrees (FS) with *frequent closed trees* [11] (FCT). Given D and a threshold sup_{min}, let f be a subtree in D and $sup(f)$ be the support of f. The subtree f is a *frequent closed tree (FCT)* if $sup(f) \geq sup_{min}$ and there exists no $f' \in D$ such that f' is a proper supertree of f and $sup(f') = sup(f)$.

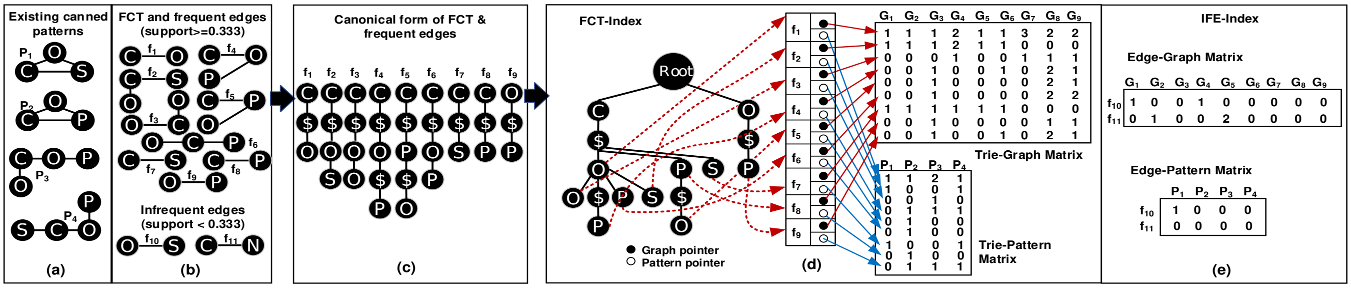

Figure 5: Frequent closed trees, frequent and infrequent edges, *FCT-Index*, and *IFE-Index*.

Example 3.3. Consider a graph database containing G_1 to G_9 in Figure 3. Let $sup_{min} = \frac{3}{9}$. The tree f_4 in Figure 5(b) is a FCT since $sup(f_4) = \frac{4}{9}$ and none of its supertrees (*e.g.*, G_6, a supertree of f_4 has support of $\frac{2}{9}$) has the same support as it. Similarly, the edge f_1 in Figure 5(b) is also a FCT. ∎

Note that the set of FCTs forms the basis from which *all* FS can be generated [11]. Hence, it is closely related to FS. Furthermore, there are fewer closed trees than frequent ones in general [11]. Consequently, FCTs significantly reduce the number of frequent structures being considered. More importantly, the closure property of FCT facilitates efficient incremental maintenance as the underlying database evolves.

LEMMA 3.4. *If a subtree f is closed in either D or ΔD, it must be closed in D ⊕ ΔD.*

For example, consider the sample graph database in Figure 3. Suppose ΔD contains G_{10} to G_{12} and $sup_{min} = 3/9$. Then f_{10} (resp. f_7) in Figure 5(b) is infrequent (resp. closed) in D containing G_1 to G_9 and in $D \oplus \Delta D$, although it is frequent (resp. not closed, since f_7's proper supertree f_2 has the same support as it) in ΔD. Hence, without scanning $D \oplus \Delta D$ and testing subgraph isomorphism, we cannot determine whether the frequent subtrees generated from D or ΔD are frequent in $D \oplus \Delta D$. In contrast, we can conclude that the closed subtrees generated from D or ΔD are closed in $D \oplus \Delta D$ (Lemma 3.4). This advantage is captured by *closure property* of FCT (detailed in Section 4.1), which greatly alleviates the computational demand of maintaining graph clusters. In addition, similar graphs have similar FCTs [27].

Finally, we add two indices, namely, *frequent closed tree index* (*FCT-Index*) and *infrequent edge index* (*IFE-Index*) to facilitate pruning of unpromising candidate patterns and fast estimation of the *pattern score*. In the sequel, we shall refer to this extension of CATAPULT as CATAPULT++.

3.4 Selective Maintenance Strategy

To address the second challenge, MIDAS considers two types of modifications to D that are identified by exploiting changes to *graphlet* frequencies in D. Graphlets are small network patterns and their frequencies have been found to characterize the topology of a network [31]. Intuitively, D can be logically viewed as a single network consisting of many disconnected subgraphs. Then, modifications to graphlet frequencies in D may provide an indication of the degree of topological changes in D as graphlets characterize network topology. Consequently, we focus on the degree of modifications to graphlet frequencies to determine the strategy for maintaining the canned patterns. Specifically, we identify the *type* of modification by comparing the Euclidean distance between the *graphlet frequency distributions* (denoted as ψ) of D and $D \oplus \Delta D$, denoted as $dist(\psi_D, \psi_{D \oplus \Delta D})$. Note that the larger the distance, the more likely D has undergone significant changes. Also, the choice of alternative distance measures do not have significant impact on the performance as reported in [24].

The rationale for using graphlet frequencies to determine the maintenance strategy is based on the observation that any canned pattern $p \in \mathcal{P}$ consists of one or more graphlets and edges (Lemma 3.5). Observe that size-3 patterns are essentially 3-node and 4-node graphlets and larger patterns are grown from them. Hence, changes to graphlet frequency distributions may impact the current set of canned patterns $\mathcal{P}$. To elaborate further, let the graphlets in D be $g_1, g_2, \ldots, g_k$, and their frequencies be $f_1, f_2, \ldots, f_k$, where $f_1 \geq f_2 \geq \ldots \geq f_k$. After database modification, let the frequencies in $D \oplus \Delta D$ be $f'_1, f'_2, \ldots, f'_k$. It is indeed possible that for $i < j$, $f_i \geq f_j$ but $f'_i < f'_j$. Since canned patterns are generated using a random walk-based approach, the probability that a particular candidate pattern is selected as a canned pattern is highly dependent on the frequencies of its edges and graphlets. Hence, a canned pattern containing graphlets whose frequencies have drastically reduced after database modification may no longer be relevant for $D \oplus \Delta D$, and needs to be updated.

LEMMA 3.5. *Any canned pattern $p_i \in \mathcal{P}$ contains one or more graphlets and edges.*

Based on the above discussion, we can classify the degree of modifications into the following two types.

- **Major modification (Type 1)**: This occurs when graphlet frequency distributions undergo significant changes. A modification is deemed *major* if $dist(\psi_D, \psi_{D \oplus \Delta D}) \geq \epsilon$ where ϵ is the *evolution ratio threshold*.
- **Minor modification (Type 2)**: In minor modification, changes to D do not impact the current set of canned patterns $\mathcal{P}$. That is, none of the patterns in $\mathcal{P}$ needs to be replaced. A modification is considered *minor* if $dist(\psi_D, \psi_{D \oplus \Delta D}) < \epsilon$.

3.5 The MIDAS Framework

Algorithm 1 outlines the MIDAS framework. First, it assigns all newly added graphs to existing clusters in D (Line 1) and removes all graphs marked for deletion (Line 2). The affected clusters are denoted as C^+ and C^-, respectively. Note that for cluster assignment (Line 1), MIDAS first computes the Euclidean distance between the FCT feature vector of a newly added graph G and that of the

Algorithm 1 The Midas Algorithm.

Require: $D, \Delta D, b = (\eta_{min}, \eta_{max}, \gamma)$, initial canned pattern set $\mathcal{P}$, existing clusters C, existing CSG set $\mathbb{S}$, existing FCT set $\mathcal{F}$, FCT support threshold sup_{min}, evolution ratio threshold ϵ;
Ensure: Updated canned pattern set $\mathcal{P}'$;
1: $(C^+, C) \leftarrow$ AssignToCluster$(C, \Delta D)$
2: $(C^-, C) \leftarrow$ RemoveFromCluster$(C, \Delta D)$
3: $\psi_D \leftarrow$ GetGraphletDistribution(D)
4: $\psi_{D \oplus \Delta D} \leftarrow$ GetGraphletDistribution$(D \oplus \Delta D)$
5: $\mathcal{F} \leftarrow$ MaintainFCT$(\mathcal{F}, \Delta D, sup_{min})$
6: $\mathbb{S} \leftarrow$ MaintainClusterSet$(C^+, C, \mathbb{S})$
7: $\mathbb{S} \leftarrow$ MaintainCSGSet$(\mathbb{S}, C, C^+, C^-)$
8: **if** Distance$(\psi_D, \psi_{D \oplus \Delta D}) \geq \epsilon$ **then**
9: $(I_{FCT}, I_{IFE}) \leftarrow$ GetIndices(D, sup_{min})
10: $\mathcal{P}' \leftarrow$ MajorModification$(C^+, C^-, \mathbb{S}, b, \mathcal{P}, I_{FCT}, I_{IFE})$
11: **end if**
12: $(I_{FCT}, I_{IFE}) \leftarrow$ MaintainIndices$(D, \Delta D, \mathcal{P}, \mathcal{P}', sup_{min}, I_{FCT}, I_{IFE})$

centroid of every cluster, then assigns G to the cluster which results in the smallest distance. Then, it calculates graphlet frequency distributions for D and $D \oplus \Delta D$ (Lines 3 and 4). Next, it performs FCT maintenance (Line 5) (Section 4.2). The modified clusters and CSGs are maintained in Lines 6 (Section 4.3) and 7 (Section 4.4), respectively. In Line 8, Midas computes the Euclidean distance between the graphlet distributions of D and ΔD to determine the type of modification and corresponding action. For major modification (Lines 9-12), Midas generates candidates patterns from CSGs of newly-generated and modified clusters (Section 5). Finally, the existing canned patterns $\mathcal{P}$ are updated using a *multi-scan swapping strategy* (Section 6). In the case of minor modification (*i.e.,* Type 2), no pattern maintenance is required. However, observe that we do maintain the underlying clusters and CSGs (Line 12) to ensure that they are consistent with $D \oplus \Delta D$.

Observe that our framework is query log-oblivious as most publicly-available graph repositories do not make such data available. Nevertheless, Midas can be easily extended to accommodate query logs by considering the weight of a pattern based on its frequency in the log during multi-scan swapping.

4 MAINTENANCE OF CLUSTERS & CSGS

In this section, we present how existing graph clusters and CSGs are maintained due to ΔD. We begin by introducing the *closure property* of FCTs and how it is utilized to maintain FCTs in Catapult++.

4.1 Closure Property of FCT

According to [11], a subgraph is *maximal* in D if it is common, and it is not a subgraph of any other common subgraph of the graphs in D. The *intersection* of a set of graphs D, denoted as $G_1 \bigcap \cdots \bigcap G_n$, is the set of all maximal subgraphs in D. The *closure* of a CT f for D is the intersection of all graphs in D containing f (denoted as $\Omega_D(f)$). The following propositions and corollaries established in [11] related to *closed trees* (CT) are also applicable to FCT since the latter is essentially a subset of CT (Section 3.3).

Proposition 4.1. *Adding (resp. deleting) a graph G containing a CT f to (resp. from) a graph dataset D does not modify the number of CT for D.*

Proposition 4.2. *Let D_1 and D_2 be two graph datasets. A tree f is closed for $D_1 \bigcup D_2$ if and only if it is in the intersection of its closures $\Omega_{D_1}(f)$ and $\Omega_{D_2}(f)$.*

Corollary 4.3. *Let D_1 and D_2 be two graph datasets. A tree f is closed for $D_1 \bigcup D_2$ if and only if (1) f is a CT for D_1, or (2) f is a CT for D_2, or (3) f is a subtree of a CT in D_1 and a CT in D_2 and it is in $\Omega_{D_1 \bigcup D_2}(\{f\})$.*

Proposition 4.4. *A tree f is closed if f is in the intersection of all its closed supertrees.*

As we shall see later, Corollary 4.3 and Proposition 4.4 can be exploited as checking conditions for closure when graphs are added to D and removed from D, respectively.

4.2 Maintenance of FCT

In Catapult++, FCTs are represented using the canonical form of frequent trees in Catapult [23] where canonical trees are first generated via normalization and then converted to canonical strings. We now describe the maintenance of FCTs. We begin by briefly describing how they are generated in Catapult++. We generate a set of closed tree (CT) by leveraging the TreeNat approach in [9]. Briefly, TreeNat uses a recursive framework to identify the set of CT (denoted as $\mathcal{F}$) in D. At each iteration, the support of all new subtrees $\mathcal{F}'$, that are extensible from f in one step, are checked. Recursive calls to TreeNat are made for all subtrees $f' \in \mathcal{F}'$ where $sup(f') \geq sup_{min}$. Note that f is added to $\mathcal{F}$ only if there is no f' s.t $sup(f) = sup(f')$. In addition, checks are done on $\mathcal{F}$ to identify $\mathcal{F}''$ that are subtrees of f where $sup(f) = sup(f'')$ and $f'' \in \mathcal{F}''$. Observe that existence of f'' violates the definition of CT [9]. Hence, they are removed from $\mathcal{F}$. In addition, $\mathcal{F}$ has to be maintained as the dataset evolves.

Midas takes the following steps to maintain $\mathcal{F}$. First, it relaxes the condition for FCT by using a lower minimum support threshold $sup_{min}/2$. Note that this avoids missing out on closed trees that may become frequent after modification to D (Lemma 4.5). For Δ^- (resp. Δ^+), the relevant FCT $\mathcal{F}_{\Delta^-}$ (resp. $\mathcal{F}_{\Delta^+}$) is found by utilizing the TreeNat approach in [9]. $\mathcal{F}_{\Delta^-}$ (resp. $\mathcal{F}_{\Delta^+}$) is then integrated with $\mathcal{F}$ using the approach in [10] (referred to as CTMiningDelete (resp. CTMiningAdd) procedure). Briefly, the CTMiningDelete (resp. CTMiningAdd) procedure identifies the integrated set of CT by checking every CT common to $\mathcal{F}$ and $\mathcal{F}_{\Delta^-}$ (resp. $\mathcal{F}_{\Delta^+}$) in size-ascending order to determine whether its subtrees remain closed after the deletion (resp. addition) operation by leveraging Prop. 4.4. For the CT that remains closed, its support and the support of all its subtrees are updated (Prop. 4.1). In addition, those subtrees that are closed in $\mathcal{F}_{\Delta^+}$ but not in $\mathcal{F}$ are added to the set of CT in accordance to Corollary 4.3. Finally, the threshold sup_{min} is restored to its original value and CT t in $\mathcal{F}$ with $sup(t) < sup_{min}$ are pruned to obtain the final set of FCT.

Lemma 4.5. *Halfing the min_{sup} prevents missing out of frequent closed trees after modification to D.*

Lemma 4.6. *The worst case time and space complexities of FCT maintenance are $O(|D||E_{max}|)$ and $O(|D|)$, respectively, where $G_{max} = (V_{max}, E_{max})$ is the largest graph in D.*

Example 4.7. Consider the graph database D in Example 3.3. Figure 5(b) shows the FCTs (f_1 to f_5). Suppose ΔD involves addition of G_{10} to G_{12} (Figure 3) to D. The FCTs are maintained as follows: (1) relax sup_{min} to 0.17; (2) identify $\mathcal{F}_{\Delta^+}$ which consists of f_1, f_2, G_{10} and five other CTs. The supports for f_1, f_2 and G_{10} are $\frac{3}{3}$, $\frac{2}{3}$ and $\frac{1}{3}$, respectively. For the remaining CTs, they are all $\frac{1}{3}$. Observe that only f_1 and f_2 are CTs common to $\mathcal{F}$ and $\mathcal{F}_{\Delta^+}$; (3) compute the support of f_1 and f_2 and their subgraphs for $D \oplus \Delta D$ (*i.e.,* updated to $\frac{12}{12}$ and $\frac{8}{12}$, respectively). The support of subgraphs of f_2 (*i.e.,* edge (C, S)) are updated to $\frac{8}{12}$ as well. The edge (C, S) is not considered a CT as f_2, its supertree has the same support. After the update, there is no change in the FCT set. However, (C, S) is now a frequent edge.

Now consider a new batch update involving deletion of G_4 and G_6. $\mathcal{F}_{\Delta^-}$ is found to be f_2, G_6 and three other CTs. f_2 and G_6 have support of $\frac{2}{2}$ and $\frac{1}{2}$, respectively, whereas that of the remaining CTs are all $\frac{1}{2}$. Only f_2 is a CT common to $\mathcal{F}$ and $\mathcal{F}_{\Delta^-}$. Hence, its support is updated to $\frac{6}{10}$ whereas those of its subgraphs (C, O) and (C, S) are updated to $\frac{10}{10}$ and $\frac{6}{10}$, respectively. In particular, (C, O) which corresponds to f_1 continues to be a FCT after the update. ∎

4.3 Maintenance of Graph Clusters

The clusters are maintained as follows using Algorithm 1: (1) Assign each newly added graph to an appropriate cluster (Line 1). (2) Remove graphs marked for deletion from existing clusters (Line 2). (3) Perform fine clustering (Section 2.3) on clusters that exceed the maximum cluster size (Line 6). Observe that fine clustering results in new clusters. In major modification, numerous graph additions and removals on a given cluster C may yield a CSG that is distinct from a CSG derived from the original C. These CSGs in turn may yield new candidate patterns and should be considered during candidate pattern generation (Section 5).

Lemma 4.8. *Worst case time and space complexities of maintaining clusters are* $O(\sum_{i=1}^{|\Delta^+|-N}(|\Delta^+| - i)\frac{(|V_{max}|+1)!}{(|V_{max}|-|V_i|+1)!)})$ *and* $O((|C^+| + |C^-|)(|V_{max}| + |E_{max}|)$*, respectively, where* $G_{max} = (V_{max}, E_{max})$ *is the largest modified graph and* N *is the maximum cluster size.*

4.4 Maintenance of CSG Set

Given graph insertions and deletions (Δ^+ and Δ^-), MIDAS takes the following steps to update the CSGs.

(1) For every $G^+ = (V^+, E^+)$ in Δ^+, retrieve the CSG $S = (V_S, E_S)$ associated with the cluster that G^+ is assigned to and update S by adding the ID of G^+ to the labels of all edges $e \in E^+ \bigcap E_S$. Further, $\forall e \in E^+ \setminus E_S$, the edge e together with its label l is added to E_S where $l(e)$ is the ID of G^+.
(2) For $G^- = (V^-, E^-)$, retrieve the CSG $S = (V_S, E_S)$ associated with the cluster that G^- is removed from. If frequency of edge $e \in E^-$ in the graph cluster associated with S is 1, update S by removing e. Otherwise, update $l(e)$ by removing the ID of G^-.

Lemma 4.9. *The worst case time and space complexities of maintaining CSGs are* $O(|E_{max}| \times (|\Delta^+| + |\Delta^-|))$ *and* $O((|\Delta^+| + |\Delta^-|)(|E_{max}| + |V_{max}|))$*, respectively.*

5 CANDIDATE PATTERN GENERATION

For Type 1 modification (*i.e.,* major), MIDAS proceeds to generate candidate canned patterns and then replaces existing "stale" patterns in $\mathcal{P}$ with these candidate patterns according to a *swap-based strategy*. These two steps are encompassed by the MAJORMODIFICATION procedure in Algorithm 1 (Line 10). In this section, we elaborate on the candidate pattern generation process. In Section 6, we shall elaborate on the swap-based strategy. We begin by introducing two indexes, *frequent closed tree index* (*FCT-Index*) and *infrequent edge index* (*IFE-Index*), to facilitate these steps.

5.1 FCT-Index and IFE-Index

Intuitively, the *FCT-Index* enables us to efficiently keep track of the existence of specific FCTs and frequent edges in data graphs and canned patterns whereas the *IFE-Index* keeps track of infrequent edges. In particular, the FCT-Index is constructed from the canonical forms of FCTs and frequent edges. Figure 5(c) depicts the canonical forms of FCTs and frequent edges in Figure 5(b). The canonical string is obtained by performing a top-down level-by-level breadth-first scan of the canonical tree. Note that the symbol $ is used to separate families of siblings (*e.g.,* O and S in f_2).

Definition 5.1. **[FCT-Index]** *Given a set of FCTs* $\mathcal{F}$ *and a set of frequent edges* E_{freq} *in D, the* ***FCT-Index*** I_{FCT} *constructed on* $\mathcal{F} \bigcup E_{freq}$ *consists of the following components:*

- *A Trie* $T = (V_T, E_T)$ *where* $v \in V_T$ *corresponds to a token of the canonical string of the FCTs and frequent edges. An edge* $e = (u, v) \in E_T$ *exists if the corresponding tokens of* $u \in V_T$ *and* $v \in V_T$ *are adjacent in the canonical strings.*
- $\forall v^\dagger \in V_T$ *where* $v^\dagger$ *is the terminating token in a canonical string, there exists a graph pointer and a pattern pointer. The graph pointer (resp. pattern pointer) of* $v^\dagger$ *points to an array containing the number of embeddings of FCTs and frequent edges in each data graph (resp. pattern) over D (resp.* $\mathcal{P}$*).*

Observe that the two array structures can be represented by a $|\mathcal{F} \bigcup E_{freq}| \times |D|$ matrix and a $|\mathcal{F} \bigcup E_{freq}| \times |\mathcal{P}|$ matrix, respectively. We refer to the former as *trie-graph matrix* (TG-matrix) and the latter as *trie-pattern matrix* (TP-matrix).

We illustrate the construction of the FCT-Index using Figure 5. First, the FCT set $\mathcal{F} = \{f_1, f_2, f_3, f_4, f_5, f_6\}$ and frequent edges $E_{freq} = \{f_7, f_8, f_9\}$ are selected from D in Figure 3. Then, the canonical strings of every FCT and frequent edge are inserted into a trie as shown in Figure 5(d). Finally, for every node in the trie representing the terminating token, a graph pointer (resp. pattern pointer) pointing to the row in the TG-matrix (resp. TP-matrix) is created. For instance, in the TP-matrix, pattern P_3 in Figure 5(a) has two embeddings of f_1, and one embedding each of f_3, f_4 and f_9.

Definition 5.2. **[IFE-Index]** *Given D containing a set of infrequent edges* E_{inf}*,* ***IFE-Index*** I_{IFE} *constructed on* E_{inf} *consists of* $|E_{inf}| \times |D|$ *edge-graph matrix (EG-matrix) and* $|E_{inf}| \times |\mathcal{P}|$ *edge-pattern matrix (EP-matrix) that store the number of embeddings for all infrequent edges over D and over canned patterns* $\mathcal{P}$*, respectively.*

An example of IFE-Index is given in Figure 5(e) where the infrequent edge $f_{11} = (C, N)$ is found in G_2 and G_5.

Observe that the aforementioned matrices are sparse. Hence, MIDAS stores only non-zero entries to reduce space usage. That is, given a sparse matrix, let $x_{(i,j)}$ be the value of the entry in the i^{th} row and j^{th} column. $\forall x_{(i,j)} > 0$, MIDAS stores i, j and $x_{(i,j)}$ in vectors a_{row}, a_{column} and a_{value}, respectively. Note that insertion and deletion occur as a tuple $(i, j, x_{(i,j)})$.

Lemma 5.3. *The time and space complexities for index construction are* $O(|D| \times |V_{max}|!|V_{max}|)$ *and* $O(|D|(|\mathcal{F}| + |E_{infreq}| + |E_{freq}|) + (n \times m))$*, respectively, where* $G_{max} = (V_{max}, E_{max})$ *is the largest graph in D, m is the maximum depth of the trie and n is the number of unique vertices in the trie.*

Remark. The exponential time complexity is due to the subgraph isomorphism checks for FCTs in D and $\mathcal{P}$. We use the VF2 algorithm [17] to this end. In practice, as we shall see in Section 7, the cost is low due to small size of FCTs. This also applies to subsequent Lemmas 5.7 and 6.4.

Index Maintenance. Given an updated set of FCT and frequent edges, the trie is updated by inserting new vertices and edges and removing deleted vertices and edges [30]. For all new FCT and frequent edges, a corresponding graph and pattern pointers are added and set to null initially. The matrices in FCT and IFE indices are maintained as follows: (1) When new FCTs or frequent edges (resp. infrequent edges) are added, new rows are added to TG- and TP- (resp. EG- and EP-) matrices. (2) When existing FCTs or frequent edges (resp. infrequent edges) are removed, corresponding rows are removed from TG- and TP- (resp. EG- and EP-) matrices. (3) When new graphs (resp. patterns) are added, new columns are added to TG- (resp. TP-) and EG- (resp. EP-) matrices. (4) When existing graphs (resp. patterns) are removed, corresponding columns are removed from TG- (resp. TP-) and EG- (resp. EP-) matrices.

Note that the indices are maintained after database modification as well as when the canned pattern set is updated.

LEMMA 5.4. *The worst case time and space complexities of maintaining the indices are* $O(|D \oplus \Delta D||E_{max}|)$ *and* $O(|D \oplus \Delta D| \times (|F_{D \oplus \Delta D}| + |E'_{freq}|))$*, respectively, where* $G_{max} = (V_{max}, E_{max})$ *is the largest graph of* $D \oplus \Delta D$.

5.2 Pruning-based Candidate Generation

The candidate generation step in CATAPULT does not exploit any pruning technique to filter unsuitable candidates early (Section 2.3). Since in the CPM problem we can exploit the knowledge of existing canned pattern set $\mathcal{P}$, *can we eliminate "unpromising" candidates early?* To this end, MIDAS exploits a novel *coverage-based pruning strategy* to guide the FCP generation process towards candidates that are deemed to have greater potential of replacing some existing patterns in $\mathcal{P}$ (referred to as *pattern swapping*).

Intuitively, a new pattern p' is a *promising* FCP if it covers a large number of data graphs that is not covered by $\mathcal{P}$ (*i.e.,* high *marginal* subgraph coverage), since p' is likely to improve upon the *pattern score.* A *swapping threshold* (κ) sets the minimum *marginal* subgraph coverage that is desired. The value of κ is updated based on the swap-based strategy. We use coverage-based pruning as it is monotonic. That is, given patterns p and p', if p contains p', then $scov(p') \geq scov(p)$. Note during canned pattern maintenance (Section 6), the candidate patterns are further assessed w.r.t pattern score that is derived from cognitive load and diversity. In particular, we deliberately refrain from integrating cognitive load-based pruning here as it allows us the flexibility to incorporate any alternative cognitive load measure in the pattern maintenance phase. Note that such measure may not be monotonic.

Definition 5.5. **[Promising FCP]** *Given* $D, \mathcal{P}$ *and swapping threshold* κ, p_c *is a* ***promising FCP*** *if:* $\exists p \in \mathcal{P}$, $|\mathcal{G}_{scov(p_c)} \setminus \bigcup_{p \in \mathcal{P}} \mathcal{G}_{scov(p)}| \geq (1+\kappa)|\mathcal{G}_{scov(p)} \setminus \bigcup_{p' \in \mathcal{P}, p' \neq p} \mathcal{G}_{scov(p')}|$ *where* $\kappa \in [0, 1]$*;* $\mathcal{G}_{scov(x)} \subseteq D$ *is a set of graphs containing* x.

MIDAS seeks to generate promising FCP efficiently by terminating the generation process early if p_c is unlikely to have high subgraph coverage. Since the FCP is constructed iteratively by adding the most

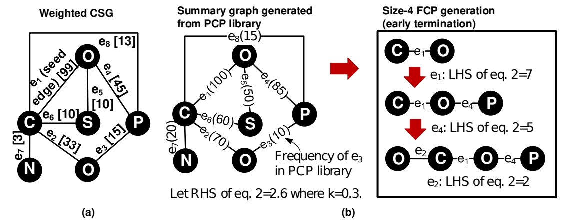

Figure 6: CCP generation.

frequently traversed edge that is connected to the partially constructed FCP (denoted as p'_c), MIDAS can perform early termination by considering the *marginal* subgraph coverage of the next edge e that is to be added to p'_c. It terminates FCP generation if e satisfies the following criteria (*i.e.,* low *marginal* subgraph coverage):

$$|\mathcal{G}_{scov(e)} \setminus \bigcup_{p \in \mathcal{P}} \mathcal{G}_{scov(p)}| < (1+\kappa) \min_{p \in \mathcal{P}} (|\mathcal{G}_{scov(p)} \setminus \bigcup_{p' \in \mathcal{P}, p' \neq p} \mathcal{G}_{scov(p')}|) \quad (2)$$

In particular, we utilize the FCT-Index and IFE-Index to compute $\mathcal{G}_{scov(e)}$. If e is a frequent edge, $\mathcal{G}_{scov(e)}$ is computed using the TG-matrix of FCT-Index. Otherwise, it can be computed using the EG-matrix of IFE-Index. The subsequent generation of CCP and FCP is similar to the CATAPULT framework (Section 2.3).

Example 5.6. Reconsider Example 3.3. Suppose $|D \oplus \Delta D| = 1000$, $\gamma = 9$, $\eta_{min} = 3$ and $\eta_{max} = 5$. Let $C_1 = \{G_1, G_2, G_6, G_8, G_9, G_{12}\}$ be a cluster. The weighted CSG of C_1 (Figure 6(a)) is generated by computing the weight w_e for each CSG. Then, MIDAS generates a library of PCPs for each pattern size by performing random walks on the weighted CSGs. Next, it identifies the FCPs from the PCP library. Figure 6(b) depicts generation of a size-4 FCP from S_{C_1}. Construction of the FCP starts from (C, O), the most frequent edge (based on 100 random walks). At each step, MIDAS checks if the current FCP ought to be pruned by considering the condition imposed by Equation 2. In Figure 6(b), early termination of FCP generation occurs after adding e_2 since it satisfies Equation 2.

LEMMA 5.7. *The worst case time and space complexities of finding CCPs and FCPs are* $O(|V_{S_{max}}|!|V_{S_{max}}||\mathbb{S}| + |\mathcal{P}|(|V_{P_{max}}|^3 + x\eta^2_{max}|\mathbb{S}||E_{S_{max}}|))$ *and* $O(|\mathbb{S}|(|E_{S_{max}}| + \eta^2_{max}) + |D||E_{max}|)$*, respectively, where* S_{max} *is the largest CSG in the set of CSGs* $\mathbb{S}$ *whose clusters have evolved,* x *is the number of random walk iterations,* P_{max} *is the largest pattern and* $G_{max} = (V_{max}, E_{max})$ *is the largest data graph.*

6 CANNED PATTERN MAINTENANCE

In this section, we present the algorithm for maintaining the canned pattern set. We begin by *adapting* the pattern score utilized in CATAPULT to suit the CPM problem.

6.1 Pattern Score

The pattern score s_p (Def. 2.1) of CATAPULT is modified by (1) replacing *cluster coverage* (*ccov*) with *subgraph coverage* (*scov*) and (2) using a tighter bound GED in computing diversity $div(p, \mathcal{P} \setminus p)$. We denote the modified score as $s'_{\mathcal{P}}$.

Cluster Coverage vs Subgraph Coverage. In the CPM problem, cluster size may change due to ΔD. Since *ccov* (Def. 2.1) is sensitive to cluster weights, we replace $ccov_p$ with $scov_p = |\mathcal{G}_p|/|D|$. That is, $s'_{\mathcal{P}} = f_{scov}(\mathcal{P}) \times f_{lcov}(\mathcal{P}) \times \frac{f_{div}(\mathcal{P})}{f_{cog}(\mathcal{P})}$. Similarly, $s'_p = scov(p, D) \times lcov(p, D) \times \frac{div(p, \mathcal{P} \setminus p)}{cog(p)}$ where $p \in \mathcal{P}$. Note that *scov* computation can be prohibitively expensive for large D. We address it by generating a sampled database $D_s \subset D$ using the *lazy sampling* technique in [23] and then computing *scov* over D_s. In addition, we leverage I_{FCT} and I_{IFE} for computing *scov*. Observe that if a pattern p is contained in a graph G, then the corresponding column entries for p in TP-matrix must be smaller than or equal to that of G in TG-matrix. Hence, the pairs (p, G), where p may be contained in G, can be found by utilizing it. In Figure 5(d), p_3 contains 2 f_1, 1 f_2, 1 f_3 and 1 f_9 (TP-matrix). From TG-matrix, G_8 and G_9 have corresponding cell entries that are greater than or equal to that of p_3. Hence, only 2 (*i.e.,* (p_3, G_8), (p_3, G_9)) instead of 9 subgraph isomorphism checks are performed for p_3 .

Tighter Bound GED. Observe that diversity of a pattern is computed using a lower bound of GED (denoted as GED_l) to reduce the number of exact GED computation. In MIDAS, we leverage a *pattern-feature matrix* (PF-matrix) to further tighten GED_l. Given a FCP $p_c = (V, E)$, each row of the matrix represents an edge $e \in E$ whereas each column represents a *subtree feature instance* (*i.e.,* FCT, frequent and infrequent edge). Since a FCP may contain multiple embeddings of a subtree feature f, these embeddings are presented as multiple columns in the PF-matrix. This is in contrast to the EG-matrix and EP-matrix where every column corresponds to a graph or a pattern instead of their embeddings. Hence, an entry $x_{(i,j)}$ in PF-matrix is 1 if G contains the j^{th} feature (denoted as $f_j = (V_f, E_f)$) and $e_i \in V \bigcap V_f$. Otherwise, it is 0. The PF-matrix of canned pattern p_3 in Figure 5(a) is given in Figure 7. p_3 contains two embeddings of f_1 (*i.e.,* $f_1(1)$ and $f_1(2)$ in the PF-matrix) and one embedding each of f_3, f_4 and f_9. We denote the embedding set of p_3 as B_{P_3}. For example, $x_{(2,5)}$ and $x_{(3,5)}$ are 1 as p_3 contains an embedding of f_4 and edges e_2 and e_3 are in f_4.

Observe that if a graph G_1 contains p_3, then $B_{p_3} \subseteq B_{G_1}$. Consider the case of another graph G_2 with one embedding of f_1 and one embedding each of f_3, f_4 and f_9. G_2 does not contain p_3 since $B_{p_3} \nsubseteq B_{G_2}$. Suppose an edge e_1 of p_3 is "relaxed" (*i.e.,* e_1 is not taken into consideration when p_3 is being matched to another graph), then the relaxed embedding set $B'_{p_3} \subseteq B_{G_2}$. That is, G_2 contains p_3 when e_1 is "relaxed". In general, when matching two given graphs $G_i = (V_i, E_i)$ and $G_j = (V_j, E_j)$ where $|E_i \bigcap E_j| > 0$ and $|E_j| > |E_i|$, G_i can be matched to G_j by progressively relaxing more and more edges. The upper bound for the number of matching edges is $|E_i| - n$ where n is the number of relaxed edges. Hence, GED_l can be tightened further as $\text{GED}'_l = \text{GED}_l + n$.

LEMMA 6.1. ***[Tighter lower bound for GED]*** *Given two graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$, the* ***tighter lower bound GED*** *is given as $\text{GED}'_l(G_A, G_B) = |V| + |E|$ where $L(V_A)$ is the set of labels of vertices in V_A, $|V| = ||V_A| - |V_B|| + Min(|V_A|, |V_B|) - |L(V_A) \bigcap L(V_B)|$, $|E| = ||E_A| - |E_B|| + n$ and n is the number of relaxed edges.*

6.2 Swap-based Pattern Maintenance

Observe that maximum coverage (MC) problem is a sub-problem of the CPM problem. However, greedy solutions typically find the maximum cover from scratch and hence cannot be effectively exploited in our problem setting. Recent works [33, 43] that address the MC problem in the context of streaming scenario use *swap-based updating techniques* instead. Specifically, they ensure that with each swap, the new cover set can outperform the cover set prior to the swap. However, these techniques are oblivious to diversity and cognitive load. Hence, we cannot adopt them directly. MIDAS realizes a *multi-scan swapping strategy* which allows progressive gain of coverage without sacrificing diversity and cognitive load. We begin by introducing the *loss* and *benefit scores* to facilitate exposition.

	$f_1(1)$	$f_1(2)$	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}
e_1	1	0	0	1	0	0	0	0	0	0	0	0
e_2	0	1	0	1	1	0	0	0	0	0	0	0
e_3	0	0	0	0	1	0	0	0	0	1	0	0

Figure 7: Pattern-feature matrix.

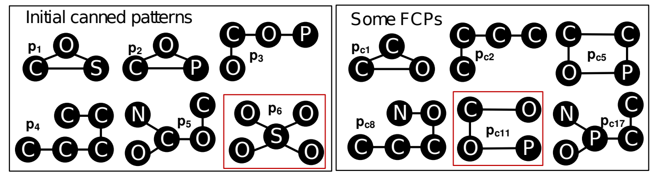

Figure 8: Swap-based pattern maintenance.

Definition 6.2. **[Loss & Benefit Scores]** *Given $\mathcal{P}$ and D, the* ***loss score*** *of a pattern $p \in \mathcal{P}$ is defined as $S_L(p, \mathcal{P}, D) = \sum_{p \in \mathcal{P}} scov(p, D) - \sum_{p' \in \mathcal{P} \setminus p} scov(p', D)$. The* ***benefit score*** *of a pattern $p_c \notin \mathcal{P}$ is defined as $S_B(p_c, \mathcal{P}, D) = \sum_{p' \in \mathcal{P} \cup p_c} scov(p', D) - \sum_{p \in \mathcal{P}} scov(p, D)$.*

MIDAS swaps an existing $p \in \mathcal{P}$ with a proposed FCP p_c if there is no significant change for pattern size distribution of $\mathcal{P}$ and $\mathcal{P} \setminus \{p\} \bigcup \{p_c\}$ and the following *swapping criteria* are satisfied:

- **sw1**: $S_B(p_c, \mathcal{P}, D) \geq (1 + \kappa) S_L(p, \mathcal{P}, D)$
- **sw2**: $s'_{p_c} \geq (1 + \lambda) s'_p$
- **sw3**: $f_{div}(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq f_{div}(\mathcal{P})$
- **sw4**: $f_{cog}(\mathcal{P}) \geq f_{cog}(\mathcal{P} \setminus \{p\} \bigcup \{p_c\})$
- **sw5**: $f_{lcov}(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq f_{lcov}(\mathcal{P})$

where κ and λ are *swapping thresholds*. Note that κ here is the same as that in Equation 2 and we use *Kolmogorov-Smirnov* test to assess if pattern size distributions are similar. sw3-sw5 are to maintain quality of the updated canned pattern set (*i.e.,* ensure optimization of $s'_{\mathcal{P}}$). Additional requirements by users such as $f_{div}(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq (1 + \alpha_1) f_{div}(\mathcal{P})$, $f_{cog}(\mathcal{P})(1 + \alpha_2) \geq f_{cog}(\mathcal{P} \setminus \{p\} \bigcup \{p_c\})$ and $f_{lcov}(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq (1 + \alpha_3) f_{lcov}(\mathcal{P})$ where $\alpha_i > 0$ can be easily handled.

MIDAS ranks all the FCPs in decreasing s'_p and stores them in a priority queue (*i.e.,* $PQ_{\mathcal{P}_c}$). Existing canned patterns are ranked in increasing s'_p and stored in another priority queue (*i.e.,* $PQ_{\mathcal{P}}$). Then, it pops the FCP with the highest s'_p and compares it with a pattern with lowest s'_p in $PQ_{\mathcal{P}}$. A swap occurs only if the swapping criteria are met. Swapping is repeated until either the $PQ_{\mathcal{P}_c}$ becomes empty or when the second swapping criterion (sw2) is not met. Observe that comparison based on sw2 can be used to terminate the swapping process. Finally, the swapped patterns are displayed on the GUI in a single update. MIDAS leverages the state-of-the-art $SWAP_\alpha$ approach in [40] for setting κ. Although the following lemma in [40] is proposed for a different problem (*i.e.,* diversified top-k subgraph matching), we can exploit it as each canned pattern can be cast as an embedding of a query graph. Lastly, we set λ same as κ for reasons discussed in Section 7.

LEMMA 6.3. *Given an initial result set $\mathcal{P}$, let κ_t be the value of κ used for the t^{th} scan of the multi-scan swap algorithm and σ_t be the* *lower bound for the approximation ratio of the result set in the t^{th} scan. At the t^{th} scanning of $SWAP_\alpha$, if $\sigma_{t-1} < 0.5$, then by setting $\kappa_t = 1 - 2\sigma_{t-1}$, the approximation ratio of the result set after the scanning is lower bounded by $\sigma_t = 0.25(\frac{1}{1-\sigma_{t-1}})$.*

Remark. Lemma 6.3 dictates $\frac{f_{scov}(\mathcal{P})}{f_{scov}(\mathcal{P}_{OPT})} = 0.25(\frac{1}{1-\sigma_{t-1}})$ at the t^{th} scan if $\sigma_{t-1} < 0.5$ and $\kappa_t = 1-2\sigma_{t-1}$. That is, the coverage of $\mathcal{P}$ is lower bounded by 0.25 times the subgraph coverage of the optimal pattern set and this coverage tends towards $0.5 f_{scov}(\mathcal{P}_{OPT})$. The diversity, cognitive load and label coverage of $\mathcal{P}$ are at least as good as the original pattern set due to **sw1** - **sw5**.

Lemma 6.4. *The worst case time and space complexities of swap-based pattern maintenance are $O(\gamma|D_s||V_{max}|!|V_{max}|+|\mathcal{P}||V_{P_{max}}|^3)$ and $O(\gamma(|V_{P_{max}}| + |E_{P_{max}}|) + |D \oplus \Delta D| \times (|F_{D\oplus\Delta D}| + |E'_{freq}|) + (\eta_{min} + \eta_{max})|C_\psi|^{\frac{\eta_{max}-\eta_{min}+1}{2}})$, respectively, where $G_{max} = (V_{max}, E_{max})$ is the largest data graph, $P_{max} = (V_{P_{max}}, E_{P_{max}})$ is the largest canned pattern and $D_s \subseteq D$.*

Example 6.5. Let $\gamma = 6$, $\eta_{min} = 3$, $\eta_{max} = 4$ and $\kappa = \lambda = 0.3$. Suppose $\mathcal{P}$ has 6 patterns (Figure 8) and 20 fcps (*i.e.,* $|\mathcal{P}_c| = 20$) are generated. Note that the fcps are stored in a priority queue $PQ_{\mathcal{P}_c} = [p_{c5}, p_{c11}, p_{c8}, p_{c17}, \cdots]$ while the canned patterns are stored in a priority queue $PQ_{\mathcal{P}} = [p_6, p_5, p_2, p_1, p_3, p_4]$. Suppose S_B, S_L, s'_{p_6} and $s'_{p_{c5}}$ are found to be 0.8, 0.7, 0.61, 0.85, respectively. Since $S_B < (1+\kappa)S_L$ and $s'_{p_{c5}} > (1+\lambda)s'_{p_6}$, p_6 is not swapped with p_{c5}. Next S_B, S_L and $s'_{p_{c11}}$ are found to be 0.8, 0.6, 0.79, respectively. Hence, $S_B > (1+\kappa)S_L$ and $s'_{p_6} > (1+\lambda)s'_{p_{c11}}$. Midas swaps p_6 with p_{c11}. In the next iteration, S_B, S_L, s'_{p_5} and $s'_{p_{c8}}$ are found to be 0.7, 0.65, 0.63, 0.73, respectively. Since $S_B < (1+\kappa)S_L$ and $s'_{p_5} < (1+\lambda)s'_{p_{c8}}$, p_5 is not swapped with p_{c8}. The scan is also terminated since $s'_{p_5} < (1+\lambda)s'_{p_{c8}}$ (p_{c_8} is similar to p_4). Consequently, the set of canned patterns after maintenance is $\{p_1, p_2, p_3, p_4, p_5, p_{c11}\}$.

7 PERFORMANCE STUDY

Midas is implemented with Java (JDK1.8). In this section, we investigate the performance of Midas and report the key findings. Additional results are discussed in [24]. All experiments are performed on a 64-bit Windows desktop with Intel(R) Core(TM) i7-4790K CPU (4GHz) and 32GB of main memory.

7.1 Experimental Setup

Datasets. We use the following datasets: (a) The aids antiviral dataset [1] with 40,000 (40K) data graphs. (b) The *PubChem* dataset [4] containing chemical compound graphs. Unless otherwise stated, *PubChem* refers to the 23K dataset. Other variants used are 250K, 500K and 1 million. (c) *eMolecule* dataset [3] consisting of 10K chemical compounds (*i.e., eMol*). We use variants of various datasets and they are denoted as <Y><X> where Y and X refer to the dataset name and the number of graphs used, respectively (*e.g.,* aids25k refers to aids dataset with 25K data graphs).

Baselines. We compare Midas against (1) maintenance from scratch using Catapult (denoted as Catapult), (2) maintenance from scratch using Catapult++ (denoted as Catapult++), (3) random swapping instead of multi-scan swap (denoted as *Random*), and (4) canned pattern set from Catapult with no maintenance (denoted as *NoMaintain*). Canned pattern set derived by an approach X is denoted as $\mathcal{P}_X$.

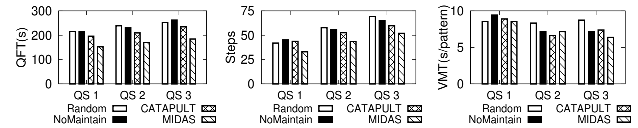

Figure 9: User study on *PubChem*.

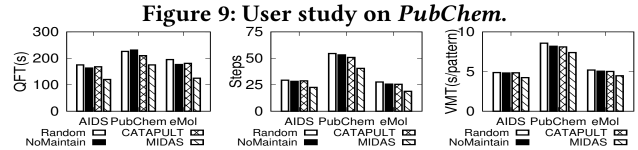

Figure 10: User study with user-specified queries.

Query set. The query set is generated by randomly selecting connected subgraphs from the dataset. Similar to [23], for each dataset, **1000 subgraph queries** with sizes in the range of [4–40] are generated. We balance the query set such that queries from Δ^+ are represented. In particular, when $|\Delta^+| > 0$, 500 queries are derived from Δ^+ and the rest from $D \setminus \Delta^-$. Otherwise, all queries are obtained from $D \oplus \Delta D$. We denote a batch addition (resp. deletion) of graphs as $+Y\%$ (resp. $-Y\%$) where $Y = \frac{|M|}{|D|} \times 100\%$ and M is the number of graphs randomly added (resp. removed).

Parameter settings. Unless specified otherwise, we set $\tau = \frac{10}{|D|}$, $\eta_{min} = 3$, $\eta_{max} = 12$, $|\mathcal{P}| = \gamma = 30$, $sup_{min} = 0.5$, $\epsilon = 0.1$, $\kappa = \lambda = 0.1$. We use the default settings in [23] for Catapult.

Performance measures. We use the following measure to assess the performance of Midas: (a) *Pattern maintenance time (pmt):* Time taken to maintain canned pattern set $\mathcal{P}$ (Algorithm 1). (2) *Missed percentage (mp):* Percentage of query set containing no canned patterns. $MP = \frac{|Q_M|}{|Q|} \times 100\%$ where Q is the query set and $Q_M \subseteq Q$ does not contain subgraphs that are isomorphic to any $p \in \mathcal{P}$. (3) *Reduction ratio* (denoted as μ): Given a subgraph query Q, $\mu = \frac{step_X - step_{\text{Midas}}}{step_X}$ where $step_X$ and $step_{\text{Midas}}$ are the *minimum* number of steps required to construct Q when $\mathcal{P}$ derived from approach X and Midas are used, respectively. Note that $\mu > 0$ implies that $\mathcal{P}$ derived from X required more steps compared to Midas. For simplicity in automated performance study, we assume: (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.

7.2 User Study

The most pertinent question related to Midas is *whether canned pattern maintenance expedites visual query formulation?* We perform a user study to address it. Note that we focus the study on this question. Additional issues and a case study are reported in [24]. 25 unpaid volunteers (ages from 20 to 39) took part in accordance to hci research that recommends at least 10 participants [21, 26]. These volunteers are students or researchers within different majors. They displayed a range of familiarity and expertise with subgraph queries according to a pre-study survey. We use the gui of Catapult [13]. We first presented a 10-min scripted tutorial of the gui describing how to visually formulate queries. We then allowed the subjects to play with the gui for 15 min.

For *PubChem23K*, aids25k and *eMol5K*, we added 6K, 10K and 3K data graphs, respectively. Then, for each dataset, 3 sets of 5

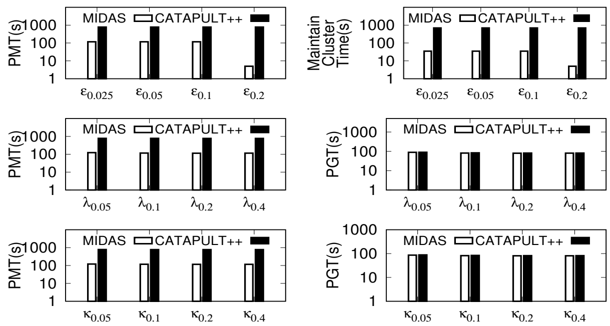

Figure 11: Effect of varying ϵ, λ and κ.

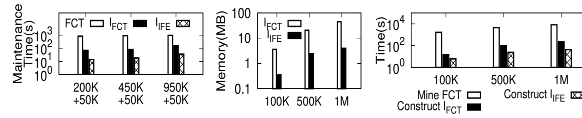

Figure 12: Cost of indices and FCT (PubChem).

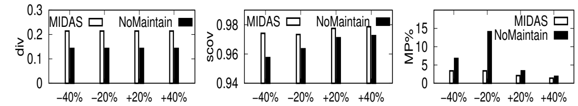

Figure 13: Comparison with no maintenance on AIDS25K.

subgraph queries with size in the range [19-45] are selected. Query set 1 (*i.e.,* QS 1) consisted of 5 queries derived from D; Query set 2 (*i.e.,* QS 2) consisted of 2 queries derived from D and 3 derived from Δ^+; Query set 3 (*i.e.,* QS 3) consisted of 5 queries derived from Δ^+. We measured the query formulation time (QFT), the number of steps required to formulate a query, and also the *visual mapping time* (VMT) which is the time required to browse and select a canned pattern for use. Unless specified otherwise, we set $|\mathcal{P}| = 30$.

To describe the queries to the users, 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 mitigate effects of learning and fatigue.

Figure 9 reports the results for *PubChem*. Query formulation using MIDAS is up to 29.5% faster and required up to 22.9% fewer steps compared to *NoMaintain* (Figure 9). VMT of MIDAS is in the range [6.4 - 8.5] and is comparable to other approaches [6.6 - 9.4]. Results on other data sets are qualitatively similar [24].

Lastly, we let users come up with their own queries. Specifically, they can formulate queries of any size and topology. On average, each user constructed 5 queries from each dataset with query size in the range of [18-42]. Figure 10 reports the results. As expected, MIDAS took the least QFT, steps and VMT on average for all datasets.

It is interesting to observe that MIDAS is superior to CATAPULT. In the latter, at each iteration, "best" pattern is added greedily to the pattern set. The order in which patterns are added impact the overall quality of the pattern set as CATAPULT does not guarantee that at each iteration, the best candidate is the optimal one. Unlike MIDAS, there is no requirement that a candidate is added only if the resultant pattern set has better quality than the old one.

7.3 Experimental Results

Exp 1: Setting the values of ϵ, κ and λ. In this set of experiments, we vary the evolution ratio and swapping thresholds on AIDS25K with batch addition of 5K graphs. Figure 11 plots the results. PMT and clustering time of MIDAS remain relatively constant when $\epsilon \leq 0.1$. A dip in these times when $\epsilon = 0.2$ is due to fewer clusters requiring maintenance compared to smaller values of ϵ. Importantly, compared to CATAPULT++, MIDAS is up to two orders of magnitude faster in terms of PMT due to shorter time required to maintain the clusters. *This highlights the efficiency of cluster and CSG maintenance using MIDAS vs regeneration of clusters and CSGs using CATAPULT++.* In particular, we set ϵ as 0.1 since variations of *scov*, *lcov* and *div* between $\mathcal{P}_{\text{MIDAS}}$ and $\mathcal{P}_{\text{CATAPULT++}}$ are less than 1% and there is an improvement of 24% in terms of *cog* (cog$\in$ [1.8, 2.3]).

Next, we vary the swapping thresholds (*i.e.,* $\kappa, \lambda \in \{0.05, 0.1, 0.2, 0.4\}$). We assess the performance based on PMT and *pattern generation time* (PGT) ,which is the time required to generate candidate patterns and swap with existing patterns. In particular, MIDAS is almost one order of magnitude faster than CATAPULT++ due to more efficient cluster and CSG maintenance since its PGT is similar to that of CATAPULT++. Observe that the effect of varying κ is similar to λ. Hence, we set $\kappa = \lambda = 0.1$.

Exp 2: Cost of indices and FCT. Next, we examine the cost of using FCT and the indices I_{FCT} and I_{IFE}. As expected, the costs of mining FCT and index construction increase as the dataset size increases (Figure 12, top left). In particular, I_{FCT} requires longer construction time and more memory than I_{IFE} due to additional data structures. The total memory requirement for the indices is 49MB for *PubChem1M* and is well within the limits of any modern machine. The maintenance time of the indices increases with the dataset size. In comparison, the FCT maintenance time increases as the size of the graph modification increases. In particular, for *PubChem1M*, maintenance of indices and FCT require around 3 and 16 minutes, respectively. Note that $|FCT|/|D|$ of *PubChem100K*, *PubChem500K* and *PubChem1M* are 0.01%, 0.001% and 0.0001%, respectively. The results are qualitatively similar for other datasets. Hence, *constructing and maintaining the FCT and indices are fast and consume a small amount of memory.*

Exp 3: Comparison with baselines. We first compare MIDAS with *NoMaintain* on AIDS25K (Figure 13). Observe that MP of $\mathcal{P}_{\text{MIDAS}}$ outperforms $\mathcal{P}_{NoMaintain}$ by 61% on average. Further, $\mathcal{P}_{\text{MIDAS}}$ exhibits greater diversity of patterns and *scov* than $\mathcal{P}_{NoMaintain}$.

Next, we compare MIDAS with CATAPULT, CATAPULT++ and *Random* on AIDS25K (Figure 14) and *Pubchem15K* (Figure 15). In terms of execution time, MIDAS is comparable with *Random* (fastest approach) and is up to an order of magnitude faster than CATAPULT. In general, MIDAS yields canned pattern set of comparable or better (*div*, *scov*, *lcov*, *cog*) quality than CATAPULT and CATAPULT++. Note that *lcov* on AIDS25K (resp. *Pubchem15K*) is approximately 1 (resp. 0.97) for all approaches and the average *cog* of MIDAS, CATAPULT and CATAPULT++ are 2.1, 2.2 and 2.5 (resp. 1.8, 2.3 and 2.6), respectively. As for μ, $\mathcal{P}_{\text{MIDAS}}$ outperforms $\mathcal{P}_{\text{CATAPULT}}$, $\mathcal{P}_{\text{CATAPULT++}}$ and $\mathcal{P}_{\text{RANDOM}}$. Furthermore, $\mathcal{P}_{\text{CATAPULT}}$ and $\mathcal{P}_{\text{CATAPULT++}}$ have higher average MP compared to $\mathcal{P}_{\text{MIDAS}}$. This highlights that *MIDAS can efficiently maintain a set of canned patterns to ensure its relevance (lowest average MP, highest average scov) across a range of graph modifications without significant loss in pattern set quality.* In comparison with random swapping, MIDAS's multi-scan swap approach

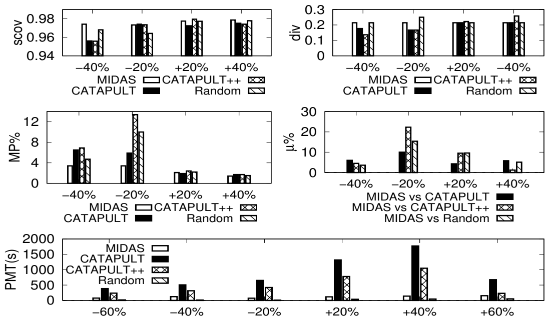

Figure 14: Baseline comparison on AIDS25K.

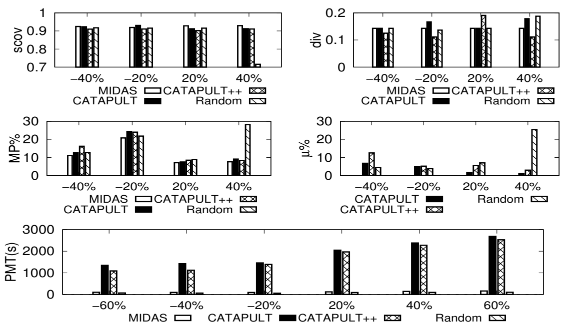

Figure 15: Baseline comparison on Pubchem15K.

has smaller MP (Figure 14, middle left, MIDAS vs *Random*) and lower μ (Figure 14, middle right, MIDAS vs *Random*). This justifies the multi-scan swap approach of MIDAS.

Exp 4: Scalability. We examine the scalability of MIDAS on *PubChem* with the following dataset DS={200K,450K, 950K} where 50K data graphs are added to each (Figure 16). The pattern quality varies in the range of [0.94-0.98], [0.94-0.97], [0.13-0.21] and [1.8-3.3] for *scov*, *lcov*, *div* and *cog*, respectively. As expected, PMT and PGT increase as dataset size increases. In this set of experiments, we defined $\mu = \frac{step_X - step_{200K}}{step_X}$ where $step_X$ is the minimum number of steps required to construct Q when $\mathcal{P}$ is derived from DS X. In particular, μ is -27.7, -6.5 and -25.9 for 250K, 500K and 1M dataset, respectively. Note that negative μ indicates greater step reduction. Further, cluster maintenance of MIDAS is faster (~2.3 min) compared to generation of cluster from scratch using CATAPULT (25 hours) for *PubChem* 1M dataset (*i.e.,* 642X). Similarly, there is a speed up of 83X in terms of PMT for MIDAS (18 min) compared to CATAPULT.

8 RELATED WORK

Most germane to this research is the canned pattern selection problem introduced in [13, 23, 44]. These approaches generate canned patterns from a large *static* collection of small- or medium-sized data graphs. In contrast, the CPM problem examines the maintenance of canned patterns in a GUI as the underlying database evolves.

There are prior work on closed frequent tree mining [9] and maintenance [10, 11]. The former focuses on mining ordered and unordered frequent closed trees whereas the latter examines the issue of mining them on evolving data streams. Our work leverages [9] and [10, 11] for mining and maintaining FCTs, respectively. In particular, MIDAS differs from these work in the following ways: (1) MIDAS involves key steps such as graph cluster and CSG set maintenance, index-based candidate pattern generation, and swap-based pattern maintenance that are not addressed by these work; (2) these efforts do not deploy any indexing schemes.

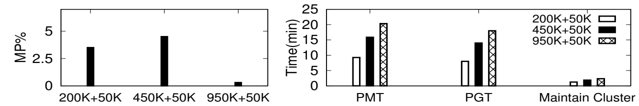

Figure 16: Scalability study on *PubChem*.

Frequent paths, trees and graphs have been utilized as indexing features to facilitate graph database search [14, 15, 39, 45, 46]. *GraphGrepSX* [14] uses a path-based index stored in a suffix tree. In contrast, *C-Trees* [45] and *Tree+*Δ [46] are tree-based and tree-and-cycle-based indices, respectively. In *C-Trees*, subtrees are extracted using frequent tree mining and a subset is selected and transformed into canonical forms and stored in a prefix tree. *Tree+*Δ also uses frequent subtrees up to a predetermined size and infrequent edges as features stored in a hash table. *gIndex* [39] and FG-index [15] are frequent subgraph-based indices. Specifically, FG-index utilizes *closed* frequent subgraphs and infrequent edges and stores them in an inverted-graph-index consisting of graph and edge arrays.

The frequent tree-based approaches lacks closure property. Hence, they are unsuitable for efficient maintenance of clusters. In contrast to FG-index, our FCT-Index and IFE-Index leverage frequent closed subtrees [11], which is more efficient to extract compared to frequent subgraphs [45]. Furthermore, FCT-Index consists of a trie with pointers to matrices instead of inverted-graph-index. IFE-Index consists of two matrices associated with data graphs and canned patterns whereas the *edge-index* in [15] is a single matrix.

There are several recent efforts on incremental graph pattern matching [18–20, 34]. These approaches examine the problem of maintaining a set of subgraphs that are results of a given query graph for an evolving data graph. Yuan *et al.* [42, 43] examined the issue of updating subgraph features that are used for indexing graph databases. The CPM problem is orthogonal to these work as we maintain a set of canned patterns.

The incremental maintenance of frequent patterns (*e.g.,* [6, 16, 38]) focus on performing incremental mining of frequent subgraphs that maximizes support (*single* objective). Hence, they cannot be applied to maintain canned patterns as they have to satisfy very different *multiple* objectives specified in Definition 3.1.

9 CONCLUSIONS

Canned patterns enhance usability of visual subgraph query formulation in direct-manipulation interfaces. However, these patterns in existing interfaces are rarely updated when the underlying graph repositories evolve. In this work, we show that the lack of maintenance of patterns adversely impact efficient visual query formulation. To alleviate this problem, we present MIDAS, which takes a data-driven approach to automatically and opportunely maintain the canned patterns of a GUI. Our maintenance strategy ensures that the updated patterns enjoy high coverage and diversity without imposing high cognitive load on the users. Our experimental study emphasize the benefits of maintaining canned patterns.

Acknowledgments. Kai Huang (partially), Sourav S Bhowmick, and Huey-Eng Chua are supported by the AcRF Tier-2 Grant MOE2015-T2-1-040. Byron Choi is supported by HKRGC GRF HKBU12201518 and IRCMS/19-20/H01. Kai Huang (partially) and Shuigeng Zhou were supported by National Natural Science Foundation of China (U1636205).

REFERENCES

[1] AIDS dataset. https://wiki.nci.nih.gov/display/NCIDTPdata/AIDS+Antiviral+Screen+Data.

[2] DrugBank interface. https://go.drugbank.com/structures/search/small_molecule_drugs/structure.

[3] eMolecules dataset. https://www.emolecules.com/info/plus/download-database.

[4] PubChem dataset. ftp://ftp.ncbi.nlm.nih.gov/pubchem/Compound/CURRENT-Full/SDF/.

[5] PubChem interface. https://pubchem.ncbi.nlm.nih.gov//edit3/index.html.

[6] E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y.C. Chang, P. Kalnis. Incremental frequent subgraph mining on large evolving graphs. *IEEE T. Knowl. Data En.* 29(12):2710-2723, 2017.

[7] S. Arora, E. Hazan, S. Kale. The multiplicative weights update method: a meta-algorithm and applications. *Theory Comput.* 8(1), 2012.

[8] D. Arthur, S. Vassilvitskii. K-means++: The advantages of careful seeding. *In SIAM*, 2007.

[9] J.L. Balcázar, A. Bifet, A. Lozano. Mining frequent closed rooted trees. *Machine Learning*, 78(1-2):1, 2010.

[10] A. Bifet, R. Gavaldà. Mining adaptively frequent closed unlabeled rooted trees in data streams. *In SIGKDD*, 2008.

[11] A. Bifet, R. Gavaldà. Mining frequent closed trees in evolving data streams. *Intell. Data Anal.*, 15(1):29-48, 2011.

[12] S.S. Bhowmick, B. Choi, C.E. Dyreson. Data-driven visual graph query interface construction and maintenance: challenges and opportunities. *PVLDB*, 9(12):984-992, 2016.

[13] S. S. Bhowmick, K. Huang, et al. AURORA: data-driven construction of visual graph query interfaces for graph databases. *In SIGMOD*, 2020.

[14] V. Bonnici, A. Ferro, R. Giugno, A. Pulvirenti, D. Shasha. Enhancing graph database indexing by suffix tree structure. *In IAPR PRIB*, 2010.

[15] J. Cheng, Y. Ke, W. Ng, et al. Fg-index: towards verification-free query processing on graph databases. *In SIGMOD*, 2007.

[16] W. Cheung, O.R. Zaiane. Incremental mining of frequent patterns without candidate generation or support constraint. *In IDEAS*, 2003.

[17] L.P. Cordella, P. Foggia, et al. A (sub)graph isomorphism algorithm for matching large graphs. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(10):1367-1372, 2004.

[18] W. Fan, J. Li, S. Ma, N. Tang, Y. Wu and Y. Wu. Graph pattern matching: from intractability to polynomial time. *In PVLDB*, 2010.

[19] W. Fan, J. Li, J. Luo, Z. Tan, X. Wang, Y. Wu. Incremental graph pattern matching. *In SIGMOD*, 2011.

[20] W. Fan, C. Hu, C. Tian. Incremental graph computations: doable and undoable. *In SIGMOD*, 2017.

[21] L.L. Faulkner. Beyond the five-user assumption: Benefits of increased sample sizes in usability testing. *Behavior Research Methods, Instruments, & Computers*, 35(3), 2003.

[22] H. He, A.K. Singh. Closure-tree: An index structure for graph queries. *In ICDE*, 2006.

[23] K. Huang, H.E. Chua, S.S. Bhowmick, B. Choi, S. Zhou. CATAPULT: data-driven selection of canned patterns for efficient visual graph query formulation. *In SIGMOD*, 2019.

[24] K. Huang, H.E. Chua, S.S. Bhowmick, B. Choi, S. Zhou. MIDAS: towards efficient and effective maintenance of canned patterns in visual graph query interfaces. *Technical Report.* Available at: https://github.com/MIDAS2020/Midas/blob/master/sigmod-midas_TR.pdf.

[25] W. Huang, P. Eades, S.H. Hong. Measuring effectiveness of graph visualizations: A cognitive load perspective. *Inf. Vis.*, 8(3):139-152, 2009.

[26] J. Lazar, J.H. Feng, H. Hochheiser. Research methods in human-computer interaction. John Wiley & Sons, 2010.

[27] G. Li, M. Semerci, B. Yener, M.J. Zaki. Graph classification via topological and label attributes. *In MLG*, 2011.

[28] E.J. Llanos, J. Leal, W. Luu, D.H. Jost, P.F., Stadler, G. Restrepo. Exploration of the chemical space and its three historical regimes. *PNAS*, 116(26):12660-12665, 2019.

[29] R.T. Marler, J.S. Arora. The weighted sum method for multi-objective optimization: new insights. *Struct. Multidiscip. O.*, 41(6):853-862, 2010.

[30] F. Morina. The trie data structure in Java. Available at *https://www.baeldung.com/trie-java.* Accessed on 30 September 2019.

[31] N. Pržulj. Biological network comparison using graphlet degree distribution. *Bioinformatics*, 23(2):e177-e183, 2007.

[32] K. Riesen, M. Neuhaus, H. Bunke. Bipartite graph matching for computing the edit distance of graphs. *In GbRPR*, 2007.

[33] B. Saha, L. Getoor. On maximum coverage in the streaming model & application to multi-topic blog-watch. *In SDM*, 2009.

[34] D. Saha. An incremental bisimulation algorithm. *In FSTTCS*, 2007.

[35] H. Shang, X. Lin, Y. Zhang, J.X. Yu, W. Wang. Connected substructure similarity search. *In SIGMOD*, 2010.

[36] B. Shneiderman, C. Plaisant. Desigining the user interface: strategies for effective human-computer interaction. 5th Ed., Addison-Wesley, 2010.

[37] C. Tofallis. Add or multiply? A tutorial on ranking and choosing with multiple criteria. *INFORMS Trans. on Education*, 14(3): 109-119, 2014.

[38] J. Wang, W. Hsu, M.L. Lee, C. Sheng. A partition-based approach to graph mining. *In ICDE*, 2006.

[39] X. Yan, P. S. Yu, J. Han. Graph indexing: a frequent structure-based approach. *In SIGMOD*, 2004.

[40] Z. Yang, A.W.C. Fu, R. Liu. Diversified top-k subgraph querying in a large graph. *In SIGMOD*, 2016.

[41] V. Yoghourdjian, D. W. Archambault, et al. Exploring the limits of complexity: A survey of empirical studies on graph visualisation. *Visual Informatics* ,2(4): 264-282, 2018.

[42] D. Yuan, P. Mitra, H. Yu, C.L. Giles. Iterative graph feature mining for graph indexing. *In ICDE*, 2012.

[43] D. Yuan, P. Mitra, H. Yu, C.L. Giles. Updating graph indices with a one-pass algorithm. *In SIGMOD*, 2015.

[44] J. Zhang, S.S. Bhowmick, H.H. Nguyen, B. Choi, F. Zhu. DaVinci: Data-driven visual interface construction for subgraph search in graph databases. *In ICDE*, 2015.

[45] S. Zhang, M. Hu, J. Yang. Treepi: A novel graph indexing method. *In ICDE*, 2007.

[46] P. Zhao, J.X. Yu, P.S. Yu. Graph indexing: tree+ delta<= graph. *In VLDB*, 2007.

[47] L. Zou, L. Chen, J.X. Yu, Y. Lu. A novel spectral coding in a large graph database. *In EDBT*, 181-192, 2008.