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

[Technical Report]

## 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 *diversity* and *cognitive load*. Experimental study with real-world datasets and visual graph interfaces demonstrates the effectiveness of MIDAS compared to static GUIS.

## 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. These GUIS typically utilize *direct-manipulation* interfaces that are appealing to novices as they can learn basic functionality quickly, are easy to remember for intermittent users by retaining operational concepts, and can be rapid for frequent users [36]. Several commercial subgraph querying frameworks in a variety of domains provide such interfaces. For instance, *PubChem* provides a GUI for structure-based chemical compound search (https://pubchem.ncbi.nlm.nih.gov/edit3/index.html).

Several commercial visual graph query interfaces (*e.g., PubChem*, *Drugbank*, *eMolecules*) for querying a large collection of small- or medium-sized data graphs expose a set of *canned patterns* (*i.e.,* small subgraph patterns). These canned patterns are beneficial to visual querying in at least three possible ways. First, they can potentially decrease the time taken to visually construct a query [11, 21]. 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, a recent study [15] on visual querying systems for sequence data revealed that majority of real-world users follow a *top-down* or *bottom-up* visual search paradigm. The former refers to searching data based on user's intuition of how the desired pattern (*i.e.,* query) should look like in theory (*i.e.,* translating a pattern "in-the-head" to visual query). The latter refers to search when a user does not have upfront knowledge of what to search for. She learns the key patterns that exist in the dataset through representative objects to jumpstart queries. Observe that canned patterns facilitate both these paradigms. In particular, diverse set of patterns on a GUI can be considered as "recommendations" that may provoke further inquiries by dragging-and-dropping these patterns onto the canvas and then modifying them. 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 especially for constructing larger queries.

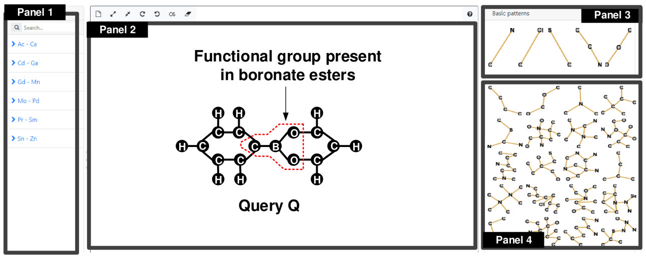


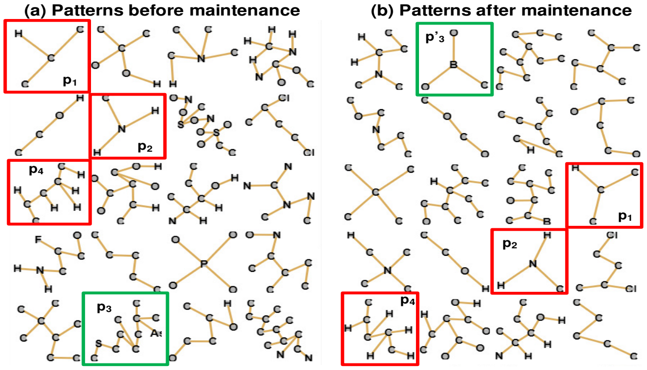
**Figure 1: A GUI.**

*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 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 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 pattern set, such bottom-up search is infeasible. ■

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 [11]. However, until recently, very little attention has been given to address this problem. CATAPULT [21] is the first effort that systematically selects canned patterns in a *data-driven* manner. Intuitively, it is desirable for canned patterns to exhibit high *coverage*, high *diversity*, and low *cognitive load* as patterns with high load may adversely impact the *visual mapping time* for selecting them during query

*Conference'17, July 2017, Washington, DC, USA*
2020. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn



(a) Patterns before maintenance

(b) Patterns after maintenance

**Figure 2: Canned pattern sets.**

formulation [21] (detailed in Section 2.2). To this end, given a graph database $D$ containing a set of small- or medium-sized data graphs and a *pattern budget* (*i.e.*, minimum and maximum size of patterns, number of patterns on the GUI), Catapult clusters $D$ based on their topological similarities. Then, it summarizes each cluster to create a *cluster summary graph* (CSG). Finally, it selects the canned patterns from these CSGs using a weighted random walk approach that seeks to maximize the coverage and diversity of the patterns. Furthermore, it preferentially selects patterns that have potentially low cognitive load on end users.

Observe that these patterns are selected from $D$ at a particular time point. However, real-world graph repositories are dynamic in nature. A study [45] reported that approximately 4,000 new structures were added daily to the *SCI finder* database (www.cas.org/products/scifinder). Consequently, these patterns may grow stale quickly over time and adversely impact efficient query formulation.

*Example 1.2.* Reconsider Example 1.1. Chemical compounds are discovered at exponential rate [27]. Suppose that the patterns are regenerated (Figure 2(b)) after *PubChem* added a new group of compounds (6375 compounds) called *boronic esters* which is characterized by the functional group outlined in Figure 1. In particular, some patterns (*e.g.*, $p_3$) have become stale and 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 $H$ and its associated edge from $p_4$; adding $3H$ vertices and 7 edges. That is, the refreshed pattern set led to more efficient formulation compared to its stale version. Also, observe that the existence of the new $p'_3$ pattern may trigger bottom-up search for boronic ester-based compounds by end users that may not be always possible if the stale GUI was used. ∎

The straightforward approach to maintain the canned pattern set by executing Catapult repeatedly from scratch as $D$ evolves can be extremely inefficient as clustering data graphs has exponential time complexity [21]. Hence, we propose Midas (**MaIntenance** 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 [21]. Specifically, it seeks to update the existing canned patterns $\mathcal{P}$ in a GUI $\mathbb{I}$ to $\mathcal{P}'$ due to evolution of underlying graph database $D$ such that $\mathcal{P}'$ continues to have high *coverage*, high *diversity* and low *cognitive load*. We assume the database changes as a *batch* of graphs insertion and deletion. This assumption is reasonable as unlike large networks (*e.g.,* social) where data may continuously appear in streaming mode, several real-world databases of small- or medium-sized data graphs are updated *periodically*. For instance, the *SCI finder* database is updated daily [45].

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 pairs of *graph extension*, there is no changes 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* which supports incremental maintenance. 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* [10] (FCT) as feature vectors for clustering. Specifically, FCTs display *closure property*, 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 to select 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 query formulation time 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). (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 Appendix A.

## 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 [21] to address the *canned pattern selection* (CPS) problem.

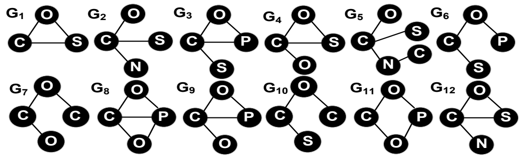


**Figure 3: A sample graph database.**

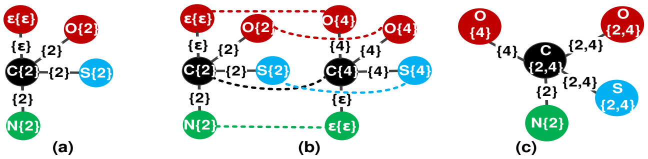


**Figure 4: (a) Extended graph of $G_2$ (Figure 3); (b) mapping of extended graphs of $G_2$ and $G_4$; (c) closure graph of $G_2$ and $G_4$.**

## 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 graph query interface $\mathbb{I}$, the number of patterns should be small and these patterns satisfy certain desirable characteristics as follows [21].

*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 $scov(p, D) = \frac{|\mathcal{G}_p|}{|D|}$ 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 $lcov(e, D) = \frac{|L(e,D)|}{|D|}$ and $L(e, D)$ is the set of graphs in $D$ containing edges having same label as $e$. Intuitively, the canned pattern set should cover a large number of data graphs in $D$ (*i.e.,* subgraph coverage) and have as many unique vertex labels as possible (*i.e.,* label coverage).

*High diversity.* In order to make efficient use of the limited display space on $\mathbb{I}$, every pattern $p$ should ideally be topologically diverse from every other pattern in $\mathcal{P}$. This will enable $\mathcal{P}$ to potentially serve a larger variety of queries. This intuition is captured using *diversity* of $p$, denoted as $div(p, \mathcal{P} \setminus p) = min\{GED(p, p_i)\}$ where $p_i \in \mathcal{P} \setminus p$ and $GED(.)$ is the graph edit distance [32]. Similarly, diversity of $\mathcal{P}$ is given as $\min_{p \in \mathcal{P}} div(p, \mathcal{P} \setminus p)$.

*Low cognitive load. Cognitive load* refers to the memory demand or mental effort required to perform a given task [23]. 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 [21, 22]. Hence, it is desirable for the patterns to impose low cognitive load on the end user. We adopt the measure in [21] 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 [23, 41]. Kai *et al.* reported the superiority of this measure to other alternatives in [21]. As we shall see later, MIDAS is orthogonal to any specific cognitive load measure of a pattern. Any superior measure can be plugged into our framework.

## 2.3 The CATAPULT Framework

CATAPULT is introduced in [21] to address the CPS problem. It comprises of the following three steps.

**Small Graph Clustering.** A 2-step clustering approach is used to partition $D$ into a set of *graph clusters* $C = \{C_1, C_2, \ldots, C_k\}$, 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 [6]. 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 $C_{fine} = \{C'_1, \cdots, C'_m\}$ where $\forall C'_i \in C_{fine}, |C'_i| \leq N$, $\omega_{\text{MCCS}}(G_m, G_n) \geq \omega_{\text{MCCS}}(G_m, G_p)$, $G_m, G_n \in C'_i$, $G_p \in C'_j$, $C'_j \in C_{fine}$ 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 $C_i \in C$ into a CSG by performing *graph closure* iteratively on pairs of data graphs in the cluster. A *closure graph* [20] is a generalized graph generated by performing a *union* on the structures of a set of graphs. It 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 $\varepsilon$ such that every vertex and edge is represented in $G^*$. Given two extended graphs $G^*_1$ and $G^*_2$ and a *mapping* $\phi$ 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 these closures may contain attribute values $\varepsilon$ corresponding to a dummy vertex or edge, which 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 the canned patterns from the 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 $lcov(e, X) = \frac{|L(e,X)|}{|X|}$. Next, CATAPULT 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 $b$), CATAPULT leverages on the statistics obtained from the walks to propose a variety of *potential candidate patterns* (PCP) from which a *final candidate pattern* (FCP) is derived. Each random walk starts with a *seed edge* (*i.e.*, edge with largest weight) and proceeds to an adjacent edge until the required number of edges is traversed or when no more edges can be added. The adjacent edge is selected from a list of *candidate adjacent edges (CAE)* where edges with higher weights are selected with higher probability. Then, the FCP of a particular size $\eta_i$ for a CSG is found by retrieving a connected subgraph of size $\eta_i$ with the most frequently traversed edges. A *pattern score* is computed for each FCP as follows.

*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 $ccov(\mathcal{P}, cw, C) = \sum_{i \in C} cw_i \times I_i$ 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 [5]. These steps are repeated until either the required number of patterns are discovered or when no new pattern can be found.

# 3 CANNED PATTERN MAINTENANCE (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 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.

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 $\Delta D$ to $D$ is a sequence of unit updates. We denote insertion of a set of data graphs in $\Delta D$ as $\Delta^+$ and deletion of a set of data graphs as $\Delta^-$. Also, $D \oplus \Delta D$ denotes a graph database after applying $\Delta D$ 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 $\eta_{min}$ (resp. $\eta_{max}$) is the minimum (resp. maximum) size of a pattern and $\gamma$ is the number of patterns to be displayed on $\mathbb{I}$, and updates $\Delta D$ to the input graph database $D$.*
- *Output: Let $f_{div}(\mathcal{P}') = \min_{p \in \mathcal{P}'} div(p, \mathcal{P}' \backslash p)$ and $f_{cog}(\mathcal{P}') = \max_{p \in \mathcal{P}'} cog(p)$ s.t. $p \subseteq \mathcal{P}'$; $p \subseteq D \oplus \Delta D$; $p_i \in \mathcal{P}' \setminus p$; $f_{scov}(\mathcal{P}')$ and $f_{lcov}(\mathcal{P}')$ be the subgraph and label coverage, respectively. Then, the output is an updated set of canned patterns $\mathcal{P}'$ on $\mathbb{I}$ for modified database $D \oplus \Delta D$ that satisfies $\max_{i \in \{scov, lcov, div\}} f_i(\mathcal{P}')$[1] and $\min f_{cog}(\mathcal{P}')$s.t. $|\mathcal{P}'| = \gamma; \eta_{min} \geq 2$, $\lceil \frac{\gamma}{\eta_{max} - \eta_{min} + 1} \rceil$ is the maximum number of patterns for each $k$-sized pattern; $k \in [\eta_{min}, \eta_{max}]$.*

**Remark.** Similar to CPS, 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 [28]. We address this by converting CPM into a single-objective optimization problem using a *multiplicative score function* [37] (Section 6). Similar to [21], here we focus on maintenance of patterns with size larger than 2. The maintenance of patterns with size 1 or 2 is given in Appendix B.

The CPM problem can be shown to be NP-hard by reducing it from the classical maximum coverage problem.

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 play a pivotal role in designing of efficient maintenance strategies [10]. 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.*

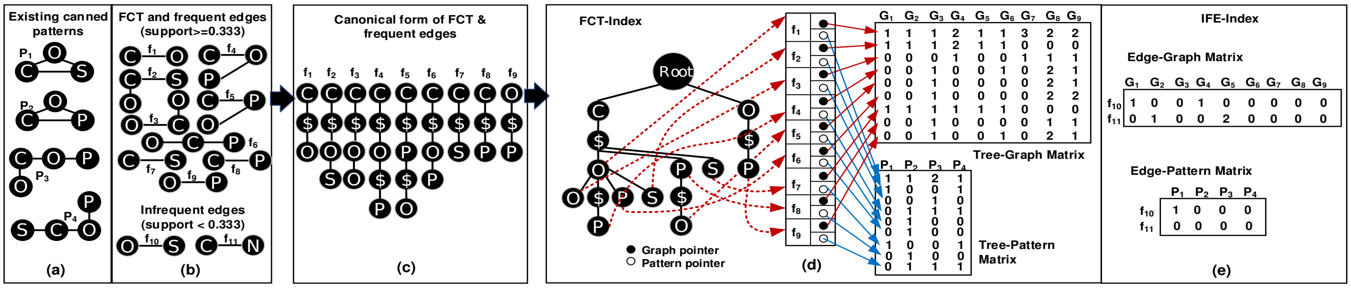
## 3.3 Scaffolding Strategy

We tackle the first challenge using *scaffolding.* Intuitively, *scaffolding* adds or modifies elements in an existing framework to support its extension. In particular, *scaffolding* is used to adapt the existing Catapult framework by replacing the frequent subtrees (FS) with *frequent closed trees* [10] (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 $f \subseteq f'$ and $sup(f') = sup(f)$.

*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 [10]. Hence, by using FCTs instead of FS, Midas heavily reduces the number of frequent structures being considered. More importantly, the *closure property* of FCT (see Section 4.1) facilitates its incremental mining as the underlying database evolves using the approach in [10]. Hence, the computational demand of maintaining graph clusters is greatly alleviated. Lastly, FCTs capture the topological characteristics of graphs where similar graphs have similar FCTs [26]. We also add two indices, namely, *frequent closed tree index* (*FCT-Index*) and *infrequent edge index* (*IFE-Index*), in Catapult to facilitate pruning of unpromising candidate patterns and

[1] We follow the formal definition convention in [30, 38] for such problems.



Tree-Graph Matrix

| $G_1$ | $G_2$ | $G_3$ | $G_4$ | $G_5$ | $G_6$ | $G_7$ | $G_8$ | $G_9$ |
|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 2 | 1 | 1 | 3 | 2 | 2 |
| 1 | 1 | 1 | 2 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 0 | 1 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 1 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 2 | 2 |
| 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 | 1 | 0 | 2 | 1 |

Tree-Pattern Matrix

| $P_1$ | $P_2$ | $P_3$ | $P_4$ |
|---|---|---|---|
| 1 | 1 | 2 | 1 |
| 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |

Edge-Graph Matrix

|  | $G_1$ | $G_2$ | $G_3$ | $G_4$ | $G_5$ | $G_6$ | $G_7$ | $G_8$ | $G_9$ |
|---|---|---|---|---|---|---|---|---|---|
| $f_{10}$ | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| $f_{11}$ | 0 | 1 | 0 | 0 | 2 | 0 | 0 | 0 | 0 |

Edge-Pattern Matrix

|  | $P_1$ | $P_2$ | $P_3$ | $P_4$ |
|---|---|---|---|---|
| $f_{10}$ | 1 | 0 | 0 | 0 |
| $f_{11}$ | 0 | 0 | 0 | 0 |

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

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]. In particular, Cannoodt *et al.* demonstrated that it is possible to optimize network topology using graphlets [12]. 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 $D$. 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 $\psi$) 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. The effect of using alternative distance measures is examined in Section 7.3.

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 3.4). 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 $f_1, f_2, \ldots, f_k$, respectively, where $f_1 \geq f_2 \geq \ldots \geq f_k$. After database modification, let the frequencies be $f'_1, f'_2, \ldots, f'_k$ in $D \oplus \Delta D$. It is indeed possible that $f_i \geq f_j$ but $f'_i < f'_j$ for $i < j$. 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 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 3.4. *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 $\epsilon$ 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 by $C^+$ 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 $G$ to that of the 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 $\Delta D$ to determine the type of modification and corresponding action. For major modification (Lines 9-12), MIDAS generates candidates patterns from CSGS of newly-generated or 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 (*e.g.,* AIDS, *PubChem*) do not make such data available due to privacy concerns. 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 $\Delta D$. We begin by introducing *closure property* of FCT and how it is utilized to maintain FCTS in CATAPULT++.

**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 $\epsilon$;
Ensure: Updated canned pattern set $\mathcal{P}'$;
$(C^+, C) \leftarrow$ AssignToCluster$(C, \Delta D)$
$(C^-, C) \leftarrow$ RemoveFromCluster$(C, \Delta D)$
$\psi_D \leftarrow$ GetGraphletDistribution$(D)$
$\psi_{D \oplus \Delta D} \leftarrow$ GetGraphletDistribution$(D \oplus \Delta D)$
$\mathcal{F} \leftarrow$ MaintainFCT$(\mathcal{F}, \Delta D, sup_{min})$
$\mathbb{S} \leftarrow$ MaintainClusterSet$(C^+, C, \mathbb{S})$
$\mathbb{S} \leftarrow$ MaintainCSGSet$(\mathbb{S}, C, C^+, C^-)$
if Distance$(\psi_D, \psi_{D \oplus \Delta D}) \geq \epsilon$ then
    $(I_{FCT}, I_{IFE}) \leftarrow$ GetIndices$(D, sup_{min})$
    $\mathcal{P}' \leftarrow$ MajorModification$(C^+, C^-, \mathbb{S}, b, \mathcal{P}, I_{FCT}, I_{IFE})$
end if
$(I_{FCT}, I_{IFE}) \leftarrow$ MaintainIndices$(D, \Delta D, \mathcal{P}, \mathcal{P}', sup_{min}, I_{FCT}, I_{IFE})$
```

## 4.1 Closure Property of FCT

According to [10], 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 $G_1 \bigcap \cdots \bigcap G_n$, 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 [10] related to *closed trees* (CT) is also applicable to FCT since the latter is essentially a subset of CT satisfying the minimum support threshold condition (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 [21] where canonical trees are first generated via normalization and then converted to canonical string. We now describe the maintenance of FCTs. We first briefly describe how they are generated in CATAPULT++ and then focus on their maintenance. We generate a set of closed tree (CT) by leveraging the TREENAT approach in [8]. 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 are 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 $f''$ violate the definition of CT [8]. 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 $\frac{sup_{min}}{2}$. Note that this avoids missing out on closed trees that may become frequent after modification to $D$ (Lemma 4.5). For $\Delta^-$, the relevant FCT $\mathcal{F}_{\Delta^-}$ is found by utilizing the TREENAT approach in [8]. $\mathcal{F}_{\Delta^-}$ is then integrated with $\mathcal{F}$ using the approach in [9] (referred to as CTMININGDELETE procedure). Briefly, the CTMININGDELETE procedure identifies the integrated set of CT by checking every CT common to $\mathcal{F}$ and $\mathcal{F}_{\Delta^-}$ in size-ascending order to determine whether its subtrees remain close after the deletion operation. Note that this can be achieved by leveraging Proposition 4.4. A similar step is taken for $\Delta^+$. The relevant CT ($\mathcal{F}_{\Delta^+}$) is integrated with $\mathcal{F}$ using the CTMININGADD procedure in [9]. Similar to CTMININGDELETE, it checks every CT common to $\mathcal{F}$ and $\mathcal{F}_{\Delta^+}$ in size-ascending order to determine if it remains closed in $\mathcal{F}$. For the CT that remains closed, its support and the support of all its subtrees are updated (Proposition 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 $\Delta D$ 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) identifying $\mathcal{F}_{\Delta^+}$ 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 $\mathcal{F}_{\Delta^+}$. Hence, they are considered further: (3) compute the support of $f_1$ and $f_2$ and their subgraphs for $D \oplus \Delta D$ and update them. Note that the support of a CT $f$ is computed as $\frac{|\mathcal{G}_f|}{|D \oplus \Delta D|}$ where $\mathcal{G}_f \subseteq D \oplus \Delta D$ is the set of graphs containing $f$. Hence, supports of $f_1$ and $f_2$ are updated to $\frac{12}{12}$ and $\frac{8}{12}$, respectively. The support of subgraphs of $f_2$ (*i.e.*, edge $(C, S)$) is 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 the remaining CTs all have $\frac{1}{2}$. Only $f_2$ is a CT common to $\mathcal{F}$ and $\mathcal{F}_{\Delta^-}$. The support of $f_2$ 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. The edge $(C, S)$ is not considered a CT as its supertree $f_2$ has the same support. $(C, O)$ which corresponds to $f_1$ continues to be a FCT after the update since none of its supertrees has the same support. ∎

## 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. *The worst case time and space complexities of maintaining the 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 ($\Delta^+$ and $\Delta^-$), Midas takes the following steps to update the CSGs.

(1) For every $G^+ = (V^+, E^+)$ in $\Delta^+$, 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 the CSGs are* $O(|E_{max}| \times (|\Delta^+| + |\Delta^-|))$ *and* $O((|\Delta^+| + |\Delta^-|)(|E_{max}| + |V_{max}|))$*, respectively, where* $G_{max} = (V_{max}, E_{max})$ *is the largest modified graph.*

## 5 CANDIDATE PATTERN GENERATION

For Type 1 modification (*i.e.,* major), Midas proceeds to generate candidate canned patterns and then replace 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 existence of specific FCTs and frequent edges in data graphs and canned patterns whereas the *IFE-Index* keeps track of infrequent edges. In particular, FCT-Index is constructed from the canonical form of the FCTs and frequent edges. Figure 5(c) depicts the canonical form of the 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 $|\mathcal{F} \bigcup E_{freq}| \times |D|$ and a $|\mathcal{F} \bigcup E_{freq}| \times |\mathcal{P}|$ matrices, 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 the example in 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 the sample graph database $D$ 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, canned 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 in these matrices 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 occurs 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 of every FCT in $D$ and $\mathcal{P}$. We use the VF2 algorithm [14] 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 [29]. 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

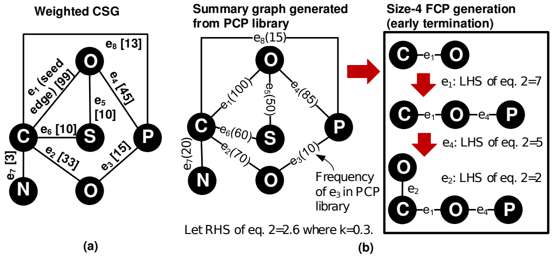


Figure 6: CCP generation.

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* ($\kappa$) sets the minimum *marginal* subgraph coverage that is desired. The value of $\kappa$ is updated based on the swap-based strategy (Section 6). We use coverage-based pruning as it is monotonic. That is, given patterns $p$ and $p'$, if $p$ contains $p'$, then the coverage of $scov(p') \geq scov(p)$. Note that the candidate patterns are further assessed w.r.t pattern score, that is derived from cognitive load and diversity, during canned pattern maintenance (Section 6). In particular, we deliberately refrain from integrating cognitive load-based pruning here as it allows our framework to be flexible by enabling incorporation of any alternative cognitive load measure in the canned pattern maintenance phase (Recall from Section 2.2). Note that such measure may not be monotonic.

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

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

In particular, we utilize the FCT-Index and IFE-Index to compute $\mathcal{G}_{cov(e)}$. If $e$ is a frequent edge, $\mathcal{G}_{cov(e)}$ 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 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 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 1. In Figure 6(b), early termination of FCP generation occurs after adding $e_2$ since it satisfies Equation 1.

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 largest CSG in set of CSGs $\mathbb{S}$ whose clusters have evolved, $x$ is number of random walk iterations, $P_{max}$ is largest pattern and $G_{max} = (V_{max}, E_{max})$ is largest data graph.*

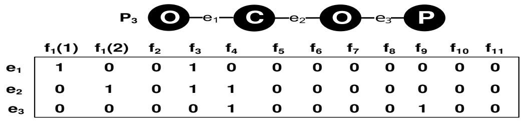
# 6 CANNED PATTERN MAINTENANCE

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

Catapult uses a pattern score $s_p$ (Def. 2.1) to assess the quality of the patterns. In Midas, this score 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 $s'_p$. We discuss these modifications in turn.

**Cluster Coverage vs Subgraph Coverage.** In the CPM problem, cluster size may change due to $\Delta D$. Since *ccov* (Def. 2.1) is sensitive to cluster weights, $s_p$ is unsuitable for comparing pattern quality effectively. Hence, we replace $ccov_p$ with $scov_p = \frac{|\mathcal{G}_p|}{|D|}$. That is, $s'_p = scov(p, D) \times lcov(p, D) \times \frac{div(p, \mathcal{P} \setminus p)}{cog(p)}$. However, *scov* computation is prohibitively expensive for large $D$. We address it by generating a sampled database $D_s \subset D$ using the *lazy sampling* technique in Catapult [21] and then computes *scov* over $D_s$. In addition, we leverage on $I_{FCT}$ 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



|  | f₁(1) | f₁(2) | f₂ | f₃ | f₄ | f₅ | f₆ | f₇ | f₈ | f₉ | f₁₀ | f₁₁ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| e₁ | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e₂ | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| e₃ | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

**Figure 7: Pattern-feature matrix.**

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 equals 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 $p_3$ to compute *scov*.

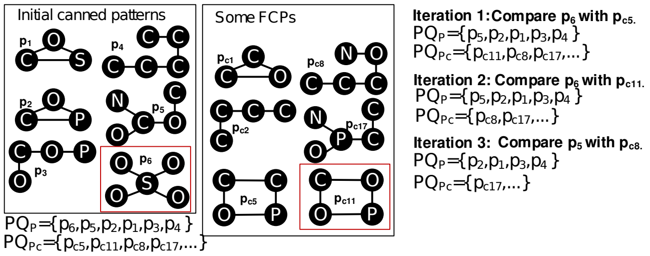
**Tighter Bound GED.** Observe that diversity of a pattern is computed using a lower bound of GED (denoted as GED$_l$) to reduce number of exact GED computation [21]. In MIDAS, we leverage a *pattern-feature matrix* (PF-matrix) to further tighten GED$_l$. Given a FCP $p_c = (V, E)$, each row of the matrix 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 correspond 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 edge $e_2$ 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 GED$'_l$ = 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 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 the 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.



$PQ_P$={$p_6$,$p_5$,$p_2$,$p_1$,$p_3$,$p_4$ }
$PQ_{Pc}$={$p_{c5}$,$p_{c11}$,$p_{c8}$,$p_{c17}$,... }

**Iteration 1:Compare $p_6$ with $p_{c5}$.**
$PQ_P$={$p_5$,$p_2$,$p_1$,$p_3$,$p_4$ }
$PQ_{Pc}$={$p_{c11}$,$p_{c8}$,$p_{c17}$,... }

**Iteration 2: Compare $p_6$ with $p_{c11}$.**
$PQ_P$={$p_5$,$p_2$,$p_1$,$p_3$,$p_4$ }
$PQ_{Pc}$={$p_{c8}$,$p_{c17}$,... }

**Iteration 3: Compare $p_5$ with $p_{c8}$.**
$PQ_P$={$p_2$,$p_1$,$p_3$,$p_4$ }
$PQ_{Pc}$={$p_{c17}$,... }

**Figure 8: Swap-based pattern maintenance.**

*Definition 6.2.* **[Loss and 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 $S_B(p_c, \mathcal{P}, D) = \sum_{p' \in \mathcal{P} \cup p_c} scov(p', D) - \sum_{p \in \mathcal{P}} scov(p, D)$.*

Note that $I_{FCT}$ and $I_{IFE}$ can be used to facilitate computation of *scov* (Section 6.1).

MIDAS swaps an existing canned pattern $p \in \mathcal{P}$ 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* (sw) 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**: $div(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq div(\mathcal{P})$
- **sw4**: $cog(\mathcal{P}) \geq cog(\mathcal{P} \setminus \{p\} \bigcup \{p_c\})$
- **sw5**: $lcov(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq lcov(\mathcal{P})$

where $\kappa$ and $\lambda$ are *swapping thresholds*. Note that $\kappa$ here is the same as that in Equation 1 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. Additional sw requirements by users such as $div(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq (1 + \alpha_1) div(\mathcal{P})$, $cog(\mathcal{P})(1 + \alpha_2) \geq cog(\mathcal{P} \setminus \{p\} \bigcup \{p_c\})$ and $lcov(\mathcal{P} \setminus \{p\} \bigcup \{p_c\}) \geq (1 + \alpha_3) lcov(\mathcal{P})$ 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.,* $PQ_{\mathcal{P}_c}$). The 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 $\kappa$. Although the following lemma in [40] is proposed for 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 $\lambda$ same as $\kappa$ for reasons discussed in Section 7.

LEMMA 6.3. *Given an initial result set $\mathcal{P}$, let $\kappa_t$ be the value of $\kappa$ used for the $t^{th}$ scan of the multi-scan swap algorithm and $\sigma_t$ 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 $\sigma_t = 0.25(\frac{1}{1-\sigma_{t-1}})$.*

According to Lemma 6.3, $\frac{cov(\mathcal{P})}{cov(\mathcal{P}_{OPT})} = 0.25(\frac{1}{1-\sigma_{t-1}})$ at the $t^{th}$ scan if $\sigma_{t-1} < 0.5$ and $\kappa_t$ is set to $1 - 2\sigma_{t-1}$.

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.* Suppose $\gamma = 6$, $\eta_{min} = 3$, $\eta_{max} = 4$, $\kappa = 0.3$, $\lambda = 0.3$ and existing $\mathcal{P}$ consists of 6 patterns as shown in Figure 8. Suppose 20 fcps (*i.e.,* $|\mathcal{P}_c| = 20$) are generated. Examples of some fcps are shown in the figure. The fcps are stored in a priority queue $PQ_{\mathcal{P}_c} = [p_{c5}, p_{c11}, p_{c8}, p_{c17}, \cdots]$. The canned patterns are stored in a priority queue $PQ_{\mathcal{P}} = [p_6, p_5, p_2, p_1, p_3, p_4]$. Observe that fcps that are ranked high in $PQ_{\mathcal{P}_c}$ (*i.e.,* $p_{c5}$ and $p_{c11}$) are structurally diverse from patterns in $\mathcal{P}$. 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 $s'_{p_5} < (1+\lambda)s'_{p_{c8}}$. This is likely due to $p_{c_8}$ being very structurally 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. 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 [3] 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 [2] 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.** This is the first work on canned pattern maintenance. Hence, we compare Midas against (1) maintenance from scratch using Catapult (denoted as Catapult), (2) maintenance from scratch using Catapult++ (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 $\mathcal{P}_X$.

**Query set.** The query set is generated by randomly selecting connected subgraphs from the dataset. Similar to [21], 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 $\Delta^+$ are represented. In particular, when $|\Delta^+| > 0$, 500 queries are derived from $\Delta^+$ 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).

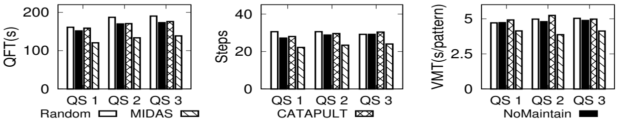


**Figure 9: User study on AIDS25K.**

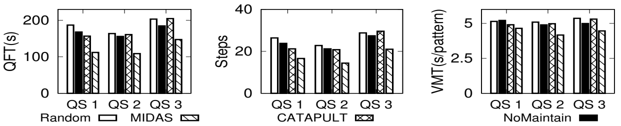


**Figure 10: User study on *eMol5K*.**

**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 [21] 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 $\mu$): 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 follow the same assumptions in [21]: (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. 25 unpaid volunteers (ages from 20 to 39) took part in accordance to hci research that recommends at least 10 participants[19, 25]. These volunteers are students or researchers within different majors including Biology, Business, and Chemical Engineering. They displayed a range of familiarity and expertise with subgraph queries according to a pre-study survey. We use the gui of Catapult [22]. 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 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

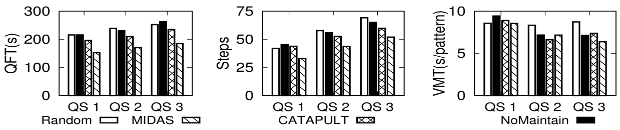


**Figure 11: User study on *PubChem*.**

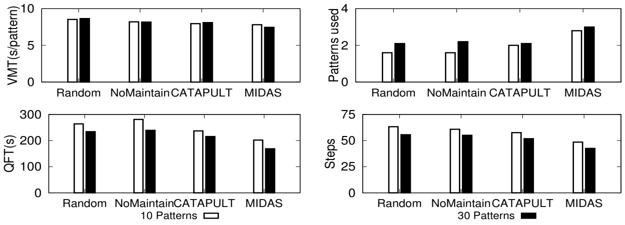


**Figure 12: Effect of varying $|P|$ on *PubChem*.**

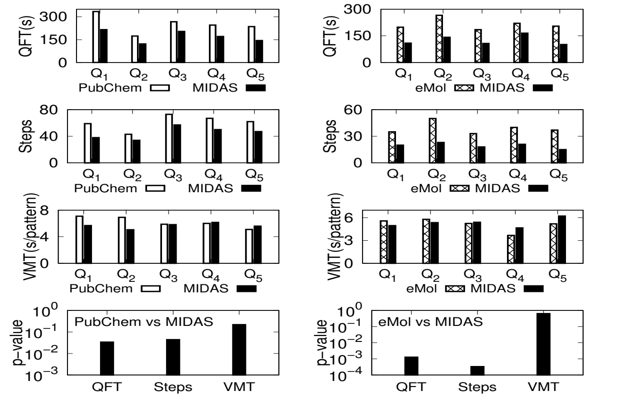


**Figure 13: Comparison with commercial GUIs.**

from $\Delta^+$; Query set 3 (*i.e.,* QS 3) consisted of 5 queries derived from $\Delta^+$. 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.

Figures 9-11 report the results. MIDAS took the least QFT and steps on average for all datasets. For AIDS25K (resp. *eMol5k*), query formulation using MIDAS is up to 20% (resp. 26%) faster and required up to 18% (resp. 27%) fewer steps compared to *NoMaintain.* VMT of all approaches vary in the range [3.9-5.2] for AIDS25K whereas VMT of MIDAS ([4.2-4.7]) is lower than other approaches ([4.9-5.4]) for *eMol5k*. For *PubChem*, query formulation using MIDAS is up to 29.5% faster and required up to 22.9% fewer steps compared to *NoMaintain* (Figure 11). VMT of MIDAS is in the range [6.4 - 8.5] and is comparable to other approaches [6.6 - 9.4].

In addition, we investigated the effect of varying the number of canned patterns ($|\mathcal{P}| = 10$ vs $|\mathcal{P}| = 30$) on GUI. Figure 12 shows a general trend of reduction in the QFT and number of steps as $|\mathcal{P}|$ increases, which is intuitive.

**Comparison with commercial GUIs**. Lastly, we compare $\mathcal{P}_{\text{MIDAS}}$ against those obtained from "static" commercial GUIs, namely, *PubChem* (pubchem.ncbi.nlm.nih.gov/edit3/index.html) and *eMol* (reaxys.emolecules.com). We extract 13 and 6 canned patterns that are of size 3 or larger from the *PubChem* and *eMol* GUIs, respectively. Note that some of these patterns contain no vertex labels (*i.e., unlabelled patterns*). We transform these unlabelled patterns into labelled ones by assigning a common label that is not found in our set of 5 queries to ensure that participants relabel the vertices to the correct ones during formulation. Further, we set $|\mathcal{P}_{\text{MIDAS}}|$ to 13 and 6 when comparing against *PubChem* and *eMol* GUIs, respectively. From Figure 13, we observe that MIDAS is faster in terms of QFT (up to 42%) and require fewer steps (up to 50%) than commercial GUIs. The superior performance is statistically significant ($p < 0.05$).

**Case Study**. We examine the use of MIDAS to maintain *PubChem* after addition of a new group of compounds (6375 compounds) called *boronic esters* which is characterized by the functional group (outlined in dotted line) shown in Figure 1. Figures 2(a) and (b) show samples of canned patterns before and after maintenance, respectively. There are 28 common patterns (*e.g.,* $p_1$ and $p_2$) between the 2 sets. In particular, some patterns (*e.g.,* $p_3$) have been swapped with candidate patterns (*e.g.,* $p'_3$) relevant to *boronic esters.* CPM took 296s and required 2914KB space. After maintenance, the canned pattern set improved in *scov* marginally (1%), maintained the same diversity and cognitive load. Suppose John, a chemist, tries to construct a query graph of *boronic acid*, he would require 20 steps (102sec) with the initial set of canned patterns. In particular, John uses $p_4$ and $p_1$; removes a $H$ and its associated edge from $p_4$; add 7 vertices ($3H$, $1C$, $1B$ and $2O$); and 10 edges. In comparison, he only require 14 steps (70sec) with the set of canned patterns that have been maintained by doing the following. He uses $p_4$, $p_1$ and $p'_3$; removes a $H$ and its associated edge from $p_4$; add $3H$ vertices and 7 edges.

## 7.3 Experimental Results

**Exp 1: Setting the values of $\epsilon$, $\kappa$ and $\lambda$.** In this set of experiments, we vary the evolution ratio and swapping thresholds on AIDS25K with batch addition of 5K graphs. Figure 14 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 $\epsilon$. 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 cluster and CSG using CATAPULT++.* In particular, we set $\epsilon$ as 0.1 since variations of *scov*, *lcov* and *div* between $\mathcal{P}_{\text{MIDAS}}$ and $\mathcal{P}_{\text{CATAPULT++}}$ is less than 1% and there is improvement of 24% in terms of *cog* (cog$\in [1.8, 2.3]$).

Next, we vary the swapping thresholds $\kappa$ and $\lambda$ in the ranges {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 $\kappa$ is similar to $\lambda$. Hence, we set $\kappa = \lambda = 0.1$.

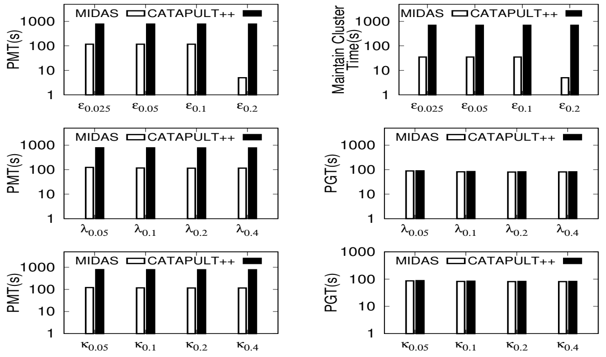


Figure 14: Effect of varying $\epsilon$, $\lambda$ and $\kappa$.

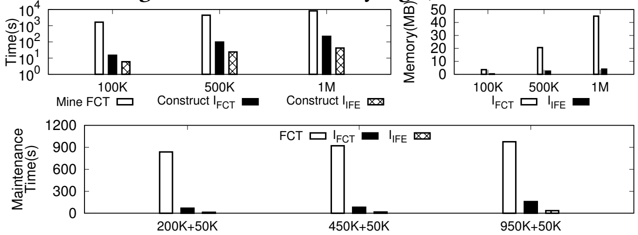


Figure 15: Cost of indices and FCT (PubChem).

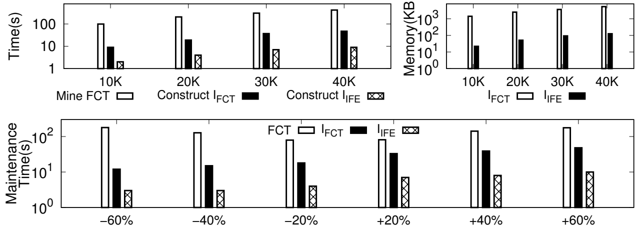


Figure 16: Cost of indices and FCT (AIDS).

**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 cost of mining FCT and index construction increases as the dataset size increases (Figures 15 and 16, 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 (Figure 15, bottom). Note that $\frac{|FCT|}{|D|}$ 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* (Figure 17). 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 $\mathcal{P}_{NoMaintain}$.

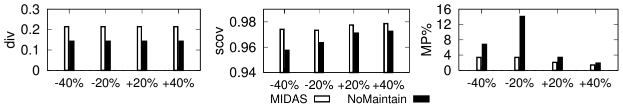


Figure 17: Comparison with no maintenance.

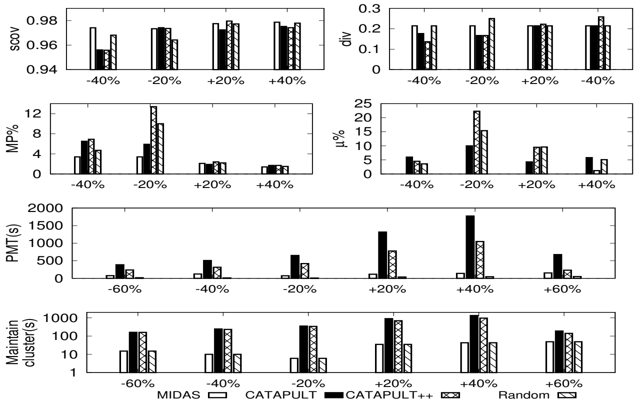


Figure 18: Baseline comparison on AIDS25K.

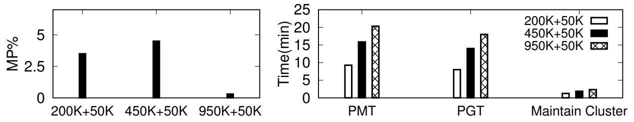
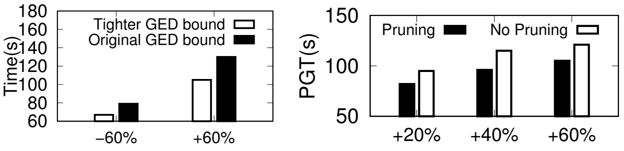


Figure 19: Scalability study on *PubChem*.

Next, we compare Midas with Catapult, Catapult++ and *Random*. In terms of execution time (Figure 18). Midas is comparable with *Random* (fastest approach) and is up to an order of magnitude faster than Catapult. Note that the time taken for maintaining cluster is similar for Midas and *Random* (Figure 18, last row). Hence, additional time taken by Midas w.r.t. *Random* for PMT is for generating the candidate patterns. In general, Midas yields canned pattern set of better quality (*scov*, *lcov*, *div*, *cog*) than Catapult and Catapult++. Note that *lcov* is approximately 1 for all approaches and the average *cog* of Midas, Catapult and Catapult++ are 2.1, 2.2 and 2.5, respectively. As expected, $\mathcal{P}_{\text{Catapult}}$ (6.5% lower) and $\mathcal{P}_{\text{Catapult++}}$ (9.4% lower) have comparatively lower $\mu$ than $\mathcal{P}_{\text{Midas}}$. Interestingly, $\mathcal{P}_{\text{Catapult}}$ (36% higher) and $\mathcal{P}_{\text{Catapult++}}$ (58% higher) 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 has smaller MP (Figure 18, second row (left), Midas vs *Random*) and lower $\mu$ (Figure 18, second row (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. The canned 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 $\mathcal{P}$ derived from DS $X$. In particular, $\mu$ is -27.7, -6.5 and -25.9 for 250K, 500K and 1M dataset, respectively. Note that negative $\mu$ indicates greater step reduction. Further, cluster maintenance of Midas is fast (~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.



(a) (b)

**Figure 20: Effect of (a) tighter GED bound and (b) coverage-based pruning on AIDS25K.**

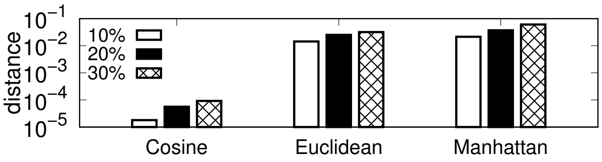


**Figure 21: Effect of alternative distance measures.**

**Exp 5: Effect of tighter GED bound.** Next, we examine the effect of using the tighter GED bound on AIDS25K by adding and removing 15K data graphs. The tighter bound reduces the PMT with average speed up of 21.5% (Figure 20a).

**Exp 6: Effect of coverage-based pruning.** Then, we study the effect of coverage-based pruning on AIDS25K by adding 5K, 10K and 15K data graphs. The pruning speeds up PGT by up to 20% (Figure 20b).

**Exp 7: Effect of alternative distance measures for graphlet frequent distribution.** Further, we examine the effect of using alternative distance measures (*i.e.,* Manhattan distance and Cosine distance) on AIDS25K(denoted as $D$). In particular, we constructed 3 new dataset $D_1$, $D_2$ and $D_3$ where 10%, 20% and 30% distinct new graphs are added to $D$, respectively. Note that the vertices of these newly added graphs contain labels that are either absent or of low occurrence in $D$. Hence, the newly added graphs are expected to be different from graphs in $D$. As a result, we have the following ground truth: $dist(D, D_3) > dist(D, D_2) > dist(D, D_1)$ where $dist(D, D_x)$ is the distance between the graphlet frequency distribution of $D$ and $D_x$. The distance measures shall be accessed based on how well it is aligned with the ground truth. All three distance measures rank the pairwise graphlet frequency distribution the same as the expected ground truth (Figure 21). The time required for execution (*i.e.,* $< 0.01s$) is similar for all three measures. Hence we randomly selected a distance measure (*i.e.,* Euclidean distance).

## 8 RELATED WORK

Most germane to this research is the canned pattern selection problem introduced in [21, 22, 44] to realize the vision of data-driven visual graph query interface construction and maintenance [11]. In particular, these approaches automatically generate canned patterns from a large collection of small- or medium-sized data graphs. These frameworks assume the underlying repository is static. In contrast, the CPM problem examines the maintenance of canned patterns in a GUI as the underlying database evolves.

There are several recent efforts on incremental graph pattern matching [16–18, 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. These patterns are maintained by ensuring the coverage and diversity are maximized and cognitive load is minimized. These issues are not relevant to these efforts.

The incremental maintenance of frequent patterns can be broadly classified into *apriori*-based (*e.g., IncGM+*[4]), *partition*-based (*e.g., PartMiner* [39]), and *concise-representation*-based (*e.g., CATS-Tree* [13]). The common thread running across these approaches is the 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 play pivotal role in enhancing usability of visual subgraph query formulation using direct-manipulation interfaces. However, these patterns in state-of-the-art academic and commercial 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 in real-world visual query interfaces. As part of future work, we plan to investigate how efficient generation and maintenance of canned patterns for GUIs designed for large networks.

# REFERENCES

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

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

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

[4] 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.

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

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

[7] A. Bonifati, M. Goodfellow, I. Manolescu, et al. Algebraic incremental maintenance of XML views. *TODS*, 38(3): 1-45,2013.

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

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

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

[11] 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.

[12] R. Cannoodt, J. Ruyssinck, J. Ramon, K. De Preter, Y. Saeys. IncGraph: Incremental graphlet counting for topology optimisation. *PloS one*, 13(4), 2018.

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

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

[15] D. J. L. Lee, J. Lee, T. Siddiqui, J. Kim, K. Karahalios, A. G. Parameswaran. You can't always sketch what you want: Understanding Sensemaking in Visual Query Systems. *IEEE Trans. Vis. Comput. Graph.*, 26(1): 1267-1277, 2020.

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

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

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

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

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

[21] 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.

[22] K. Huang, H.E. Chua, S.S. Bhowmick, B. Choi, S. Zhou. AURORA: Data-driven Construction of Visual Graph Query Interfaces for Graph Databases. *In SIGMOD*, 2020.

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

[24] R.M. Karp. Reducibility among combinatorial problems. In *Complexity of computer computations*, 85-103, 1972.

[25] J. Lazar, J.H. Feng, H. Hochheiser. Research Methods in Human-Computer Interaction. John Wiley & Sons, 2010.

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

[27] 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.

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

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

[30] M. Nouiehed, M. Sanjabi, T. Huang, J.D. Lee, M. Razaviyayn. Solving a class of non-convex min-max games using iterative first order methods. *In NIPS*, 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] R. Udwani. Multi-objective maximization of monotone submodular functions with cardinality constraint. *In NIPS*, 2018.

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

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

# A PROOFS

**Proof of Theorem 3.2 (Sketch).** The CPM is a multi-objective optimization problem which can be reformulated as a constrained single-objective optimization problem where the objective function is max $f_{scov}$ and the constraints are max($f_{lcov}, f_{div}$) and min $f_{cog}$. This reformulated problem (*i.e.,* max $f_{scov}$) can be reduced from the maximum coverage problem, which is a classical NP-hard optimization problem [24]. In particular, given a number $k$ and a collection of sets $S$, the maximum coverage problem aims to find a set $S' \subset S$ such that $|S'| \leq k$ and the number of covered elements is maximized. In CPM, the collection of sets $S$ is the set that consists of all possible subgraphs of the graph dataset $D$. The subset $S'$ is the canned pattern set and $k$ is the size of the canned pattern set. The number of covered elements corresponds to the number of covered graphs in $D$. Note that the reformulated optimization problem is at least as hard as the maximum coverage problem since optimizing the objective may result in solutions that are sub-optimal with regards to additional imposed constraints.

**Proof of Lemma 3.4 (Sketch).** Let $p_i$ denote a canned pattern with $i$ edges. Observe that the edge is the basic building block of a pattern and $p_i$ can be constructed using $i$ edges. Consider $p_3$, which has only 3 possible pattern topology (*i.e.,* triangle, 3-star, 3-path) where 3-star and 3-path are 4-node graphlets and triangle is a 3-node graphlet. When $p_4$, there are 5 possible topologies of which 2 are 4-node graphlets and the remaining are 5-node graphlets. Observe that these 5 topologies can be "grown"by adding an appropriate edge to triangle, 3-star, or 3-path. That is, $p_4$ contains subgraphs that are isomorphic to graphlets in $p_3$. Similarly, larger $p_i$ can be constructed by "growing" graphlets with edges. Hence, each canned pattern $p$ contains one or more graphlets and edges.

**Proof of Lemma 4.5 (Sketch).** Given $sup_{min}$, if $f$ is frequent in $D \oplus \Delta^+$ where $|D| = n_1$ and $|\Delta^+| = n_2$, then $freq(f, D \oplus \Delta^+) \geq (n_1 + n_2) \times sup_{min}$ where $freq(f, X)$ is the frequency of $f$ in $X$. If $f$ is not frequent in $D$, then $freq(f, D) < n_1 \times \frac{sup_{min}}{2}$. Since $freq(f, \Delta^+) = freq(f, D \oplus \Delta^+) - freq(f, D)$ and $freq(f, D \oplus \Delta^+) - freq(f, D) \geq freq(f, D \oplus \Delta^+) - n_1 \times \frac{sup_{min}}{2}$, therefore $freq(f, \Delta^+) \geq n_2 \times sup_{min} + \frac{n_1}{2} \times sup_{min}$. This can be further written as $freq(f, \Delta^+) > n_2 \times \frac{sup_{min}}{2}$. Note that the RHS implies $f$ is frequent in $\Delta^+$. Further, if $f$ is not frequent in $\Delta^+$, then similarly, $f$ is frequent in $D$. Hence, $f$ will not be missed. Alternatively, if $f$ is frequent in $D \setminus \Delta^-$, let $|D \setminus \Delta^-| = n_2$. Then, any closed tree $f$ is frequent in $D \oplus \Delta^-$ satisfies condition that $freq(f, D \oplus \Delta^-) \geq n_2 \times sup_{min}$. Since $\frac{freq(f,D)}{n_1} = \frac{freq(f,D \setminus \Delta^-)+freq(f,\Delta^-)}{n_1}$, $\frac{freq(f,D)}{n_1} \geq \frac{freq(f,D \setminus \Delta^-)}{n_1}$. Further, $\frac{freq(f,D \setminus \Delta^-)}{n_1} \geq \frac{n_2 \times sup_{min}}{n_1}$. We can assume that number of removed graphs is less than or equals to half of $|D|$ (*i.e.,* $n_1 - n_2 \leq n_2$) because if it is otherwise, MIDAS would perform FCT mining of $D \setminus \Delta^-$. Hence, $\frac{freq(f,D)}{n_1} \geq \frac{n_2 \times sup_{min}}{2 \times n_2}$. This in turn gives us, $\frac{freq(f,D)}{n_1} \geq \frac{sup_{min}}{2}$. Hence, $f$ will not be missed.

**Proof of Lemma 4.6 (Sketch).** The time complexity of FCT maintenance is due to TREENAT which requires $O(|D||E_{max}|)$ time in the worst case when every edge in every graph of $D$ is traversed and these graphs have the same number of edges (*i.e.,* $|E_{max}|$). In FCT maintenance procedure, $O(|D|)$ space is required for storing $D$. In practice, $|\mathcal{F}|$ is small. Coupled with the small size of individual FCT, the space requirement for storing $\mathcal{F}$ is expected to be much lesser compared to that of $D$. Hence, in the worst case, the space complexity of FCT maintenance is $O(|D|)$.

**Proof of Lemma 4.8 (Sketch).** The worst case time complexity of cluster maintenance is due to the fine clustering step. In the worst case, all the clusters containing newly added graphs exceed the maximum cluster size and fine clustering is performed. The worst case time complexity is $O(\sum_{i=1}^{|\Delta^+|-N} (|\Delta^+|-i) \frac{(|V_{max}|+1)!}{(|V_{max}|-|V_i|+1)!)})$ where $G_{max} = (V_{max}, E_{max})$ is the largest modified graph and $N$ is the maximum cluster size. The proof follows that in [21]. Since all modified graphs have to be stored, the worst case space complexity is $O((|C^+| + |C^-|)(|V_{max}| + |E_{max}|)$.

**Proof of Lemma 4.9 (Sketch).** In the worst case, all modified graphs are of the same maximum size. CSG maintenance requires processing of each edge in the modified graph. Hence, worst case time complexity is $O(|E_{max}| \times (|\Delta^+| + |\Delta^-|))$ where $G_{max} = (V_{max}, E_{max})$ is the largest modified graph. In terms of worst case space complexity, MIDAS has to store all modified graphs. Hence, worst case space complexity is $O((|\Delta^+| + |\Delta^-|)(|E_{max}| + |V_{max}|))$.

**Proof of Lemma 5.3 (Sketch).** The worst time complexity is due to the subgraph isomorphism checks of every FCT in $D$ and $\mathcal{P}$ and it is $O(|D| \times |\mathcal{F}| \times |V_{max}|!|V_{max}|)$ where $G_{max} = (V_{max}, E_{max})$ is the largest graph in $D$ since $\mathcal{P} \subset D$, $|D| \gg |\mathcal{P}|$ and FCT are expected to be much smaller in size than $G_{max}$. In practice, there are in fact very few FCT compared to $|D|$ (See Section 7). Hence, the worst time complexity can be represented as $O(|D| \times |V_{max}|!|V_{max}|)$. The matrices require $(|D| + |\mathcal{P}|) \times (|\mathcal{F}| + |E_{infreq}| + |E_{freq}|)$ space whereas the pointers require $|\mathcal{F}|$ space. In the worst case, $\mathcal{F}$ and $E_{freq}$ are 2 non-intersecting sets and every vertex labels in them are unique, resulting in worst case space complexity of $2 \times |E_{freq}| + |\mathcal{F}|(|V_{f_{max}}| + |E_{f_{max}}|)$ where $f_{max}$ is the largest FCT in $\mathcal{F}$. Hence, the worst case space complexity for index construction is $|D|(|\mathcal{F}| + |E_{infreq}| + |E_{freq}|)$ since $|D| \gg |\mathcal{P}|$, $|D| \gg |V_{f_{max}}|$ and $|D| \gg |E_{f_{max}}|$.

**Proof of Lemma 5.4 (Sketch).**The worst case time complexity occurs when graph modification happens. In this case, it takes $O(|D \oplus \Delta D||E_{max}|)$ time to retrieve the set of FCTs [9] where $G_{max} = (V_{max}, E_{max})$ is the largest graph of $D \oplus \Delta D$. Updating of trie requires $O(L)$ where $L$ is the length of the key. Since for large dataset, $|D \oplus \Delta D| \gg L$, worst case time complexity is $O(|D \oplus \Delta D||E_{max}|)$. The trie, the pointers and the matrices have to be stored. In the worst case, the size of the trie is $O(|F_{D \oplus \Delta D}| \times f_{max} + |E'_{freq}|)$ where the set of FCTs $F_{D \oplus \Delta D}$ and frequent edges $E'_{freq}$) are distinct and $f_{max}$ is the maximum size of a FCT. Correspondingly, the size of pointers is $O(|F_{D \oplus \Delta D}| + |E'_{freq}|)$ and TG matrix requires the largest storage of $|D \oplus \Delta D| \times (|F_{D \oplus \Delta D}| + |E'_{freq}|)$ assuming that each entry in the matrix is non-zero. Hence, worst case space complexity is $O(|D \oplus \Delta D| \times (|F_{D \oplus \Delta D}| + |E'_{freq}|))$.

**Proof of Lemma 5.7 (Sketch).** In the worst case, no FCPs are pruned. Finding weights of edges in the closure graphs require $O(|\mathbb{S}||E_{\mathbb{S}_{max}}|)$ time in the worst case where $\mathbb{S}_{max} \in \mathbb{S}$ is the largest closure graph. Generating PCP requires $O(x\eta_{max}^2|\mathbb{S}||\mathcal{P}||E_{\mathbb{S}_{max}}|)$ time where $x$ is the number random walk iterations. Similar to CATAPULT, MIDAS utilizes edge occurrence from the random walk to identify the FCP. For every PCP library, computing edge occurrence requires $O(x\eta_{max})$ time while FCP generation takes $O(\eta_{max}|E_{\mathbb{S}_{max}}|)$ time. Computing pattern score requires subgraph isomorphism test for each closure graph to find $\beta_p$ ($O(|V_{\mathbb{S}_{max}}|!|V_{\mathbb{S}_{max}}|)$ [14]) and $|\mathcal{P}'|$ times of graph edit distance computation ($O(|V_{P_{max}}|^3)$ [32] where $P_{max}$ is the largest pattern in $\mathcal{P}$) to find $\lambda_p$, yielding $O(|V_{\mathbb{S}_{max}}|!|V_{\mathbb{S}_{max}}||\mathbb{S}| + |\mathcal{P}||V_{P_{max}}|^3)$ worst case time complexity for each FCP. Updating of cluster weights and edge label occurrence require $O(|V_{\mathbb{S}_{max}}|!|V_{\mathbb{S}_{max}}||\mathbb{S}|)$ and $O(\eta_{max})$ time, respectively. Taken together, the pattern mining and selection phase has worst-time complexity of $O(|V_{\mathbb{S}_{max}}|!|V_{\mathbb{S}_{max}}||\mathbb{S}| + |\mathcal{P}|(|V_{P_{max}}|^3 + x\eta_{max}^2|\mathbb{S}||E_{\mathbb{S}_{max}}|))$.

*Space complexity:* There are $|\mathcal{P}|$ canned patterns. Since we expect canned patterns to be subgraphs of $D$, their sizes should be less than $O(|V_{max}| + |E_{max}|)$. Hence, storage space needed for the candidate patterns is $O(|\mathcal{P}|(|V_{max}| + |E_{max}|))$. In addition, MIDAS allocates weights to each closure graph and this requires $O(|\mathbb{S}||E_{\mathbb{S}_{max}}|)$ space. In the worst case, maintaining the edge label weight requires $O(|D||E_{max}|)$ space assuming that every edge in each graph in $D$ has a unique label. For each PCP library, $O(x\eta_{max})$ space is needed where $x$ is the number of random walk instances. During each iteration, there are $\eta_{max} - \eta_{min} + 1$ candidate canned patterns per closure graph. Hence, in the worst case, generating CCPs and FCPs require space complexity of $O(|\mathcal{P}|(|V_{max}| + |E_{max}|) + |\mathbb{S}||E_{\mathbb{S}_{max}}| + |D||E_{max}| + \eta_{max}^2|\mathbb{S}|)$ since $x\eta_{max} \ll |D||E_{max}|$ in typical large graph dataset. This can be further reduced to $O(|D||E_{max}| + |\mathbb{S}|(|E_{\mathbb{S}_{max}}| + \eta_{max}^2))$ since $|D| \gg |\mathcal{P}|$ and in the worst case, $G_{max}$ is a strongly connected graph where $|E_{max}| > |V_{max}|$.

**Proof of Lemma 6.1.** Given two graphs $G_A = (V_A, E_A)$ and $G_B = (V_B, E_B)$, the lower bound GED of vertices consists of the minimum number of node modifications ($M_V$) and edge modifications ($M_E$) required to transform $G_A$ to $G_B$. Observe that $M_V$ consists of (1) adding (resp. removing) $||V_A| - |V_B||$ nodes if $G_A$ contains less (resp. more) nodes than $G_B$ and (2) modifying the node labels in $G_A$ to make sure they are the same as those in $G_B$. The latter requires $Min(|V_A|, |V_B|) - |L(V_A) \bigcap L(V_B)|$ steps. In lower bound GED (denoted as $\text{GED}'_l$) as defined in [21], $M_E$ consists of only adding (resp. removing) $||E_A| - |E_B||$ edges if $G_A$ contains less (resp. more) edges than $G_B$. That is $\text{GED}'_l(G_A, G_B) = ||V_A| - |V_B|| + Min(|V_A|, |V_B|) - |L(V_A) \bigcap L(V_B)| + ||E_A| - |E_B||$. Observe that this definition of lower bound GED does not consider the "rewiring" of edges where an existing edge $(v_s, v_t)$ is modified to become $(v'_s, v'_t)$. In the tighter lower GED bound (denoted as $\text{GED}_l(G_A, G_B)$), such edges are considered. That is, by iteratively removing edges (*i.e.,* relaxed edges) which require "rewiring", $G_A$ progressively achieves a match to $G_B$. Let $n$ be the number of relaxed edges. Then, $\text{GED}_l(G_A, G_B) = \text{GED}'_l(G_A, G_B) + n$. Hence, $\text{GED}_l(G_A, G_B)$ is tighter than $\text{GED}'_l(G_A, G_B)$ when $n > 0$.

**Proof of Lemma 6.3.** First, we proof the following lemma:
*For a scan, assume that the coverage of the initial set of canned patterns $\mathcal{P}$ is lower bounded by $\sigma cov(\mathcal{P}_{OPT})$,* i.e., *$cov(\mathcal{P}) \geq \sigma cov(\mathcal{P}_{OPT})$.*

$$(2 + \frac{1}{\kappa} + \kappa) cov(\mathcal{P}_{final}) \geq (1 + \frac{\sigma}{\kappa}) cov(\mathcal{P}_{OPT}) \tag{2}$$

Similar to [40], the proof is based on the concepts of set charge and element charge in [33] and [43]. The analysis is based on tracking the coverage $cov(\mathcal{P}_{OPT})$ of an optimum solution $\mathcal{P}_{OPT}$ as canned patterns in $\mathcal{P}_{OPT}$ are examined. For each canned pattern $p \in \mathcal{P}_{OPT}$, if $p$ is not selected, a set charge is computed. If it is selected, then an element charge is computed for each vertex in $p$. This ensures that the total charge is an upper bound of $cov(\mathcal{P}_{OPT})$.

*[Set Charge]* Let $\beta = 1 + \kappa$. For a canned pattern $p_i \in \mathcal{P}_{OPT}$, let $H_i$ be the set of $k$ canned patterns when $p_i$ is examined. If $p_i$ is not selected, then $\forall f \in H_i$, $B(p_i, H_i) < \beta \times L(f, H_i)$ where $B(.)$ and $L(.)$ are the benefit and loss functions, respectively (Definition 6.2). Hence, $kB(p_i, H_i) < \beta \sum_t L(f_t, H_i)$. Since $\sum_t L(f_t, H_i) \leq cov(H_i)$, $kB(p_i, H_i) < \beta cov(H_i)$. Set charge for each pattern in $H_i$ is given by $\frac{B(p_i, H_i)}{cov(H_i)} < \frac{\beta}{k}$. Note further that $cov(\mathcal{P}_{final}) \geq cov(H_i)$. Hence, total set charge for all canned patterns in $\mathcal{P}_{OPT}$ is less than $\sum_{i=1}^{k} cov(H_i) \times \frac{\beta}{k} = \sum_{i=1}^{k} \frac{\beta cov(H_i)}{k} \leq \beta cov(\mathcal{P}_{final})$.

*[Element Charge]* Let $\mathcal{P}_i$ be the canned pattern set when $p_i$ is removed from it, where $p_i$ is the $i^{th}$ canned pattern swapped out by the algorithm. Hence, $p_i \in \mathcal{P}_i$ and $p_i \notin \mathcal{P}_{i+1}$. Let $\mathcal{P}_{final}$ be the final canned pattern set. Similar to [40], the element charge is to keep track of the coverage of vertices in canned patterns in $\mathcal{P}_{OPT}$ that have been selected, which may either appear in $\mathcal{P}_{final}$ or be swapped out. The total element charge is at most $cov(\mathcal{P}_{final}) + \sum_{i \geq 0} L(p_i, \mathcal{P}_i)$, where $p_i$ is the $i^{th}$ removed canned pattern.

$$\begin{aligned} cov(\mathcal{P}_{i+1}) - cov(\mathcal{P}_i) &\geq B(p_i, \mathcal{P}_i) - L(p_i, \mathcal{P}_i) \\ &\geq (\beta - 1) L(p_i, \mathcal{P}_i) \end{aligned} \tag{3}$$

$$\begin{aligned} \sum_{i \geq 0} L(p_i, \mathcal{P}_i) &\leq \frac{1}{\beta - 1} \sum_{i \geq 0} (cov(\mathcal{P}_{i+1}) - cov(\mathcal{P}_i)) \\ &= \frac{1}{\beta - 1} (cov(\mathcal{P}_{final}) - cov(\mathcal{P})) \end{aligned} \tag{4}$$

*[Total Charge]* Summing up the set charges and element charges, we get $\beta cov(\mathcal{P}_{final}) + cov(\mathcal{P}_{final}) + \frac{1}{\beta - 1}(cov(\mathcal{P}_{final}) - cov(\mathcal{P}))$. Observe that this sum upper bounds $cov(\mathcal{P}_{OPT})$. Since $\beta = 1 + \kappa$, $(2 + \frac{1}{\kappa} + \kappa) cov(\mathcal{P}_{final}) - \frac{1}{\kappa} cov(\mathcal{P}) \geq cov(\mathcal{P}_{OPT})$. If the coverage of the initial set of $k$ canned patterns is lower bounded by $\sigma cov(\mathcal{P}_{OPT})$, *i.e., $cov(\mathcal{P}) \geq \sigma cov(\mathcal{P}_{OPT})$*, we have $(2 + \frac{1}{\kappa} + \kappa) cov(\mathcal{P}_{final}) \geq (1 + \frac{\sigma}{\kappa}) cov(\mathcal{P}_{OPT})$.

*[Proof of Lemma 6.3]* From Inequality 2, we obtain

$$\frac{cov(\mathcal{P}_{final})}{cov(\mathcal{P}_{OPT})} \geq \frac{1 + \frac{\sigma}{\kappa}}{2 + \frac{1}{\kappa} + \kappa} = \frac{\kappa + \sigma}{(\kappa + 1)^2} \tag{5}$$

Differentiating w.r.t. $\kappa$ and setting the result to zero, we obtain an optimal value for $\kappa$ for the swapping condition.

$$\kappa = 1 - 2\sigma \tag{6}$$

From Equations 5 and 6,

$$\kappa_{t+1} = 1 - 2\sigma_t \tag{7}$$

$$\sigma_{t+1} = \frac{\kappa_{t+1} + \sigma_t}{(\kappa_{t+1} + 1)^2} = \frac{1}{4(1 - \sigma_t)} \tag{8}$$

Note that in [33], $\kappa_1$ is set to 1 which follows $\kappa_t = 1 - 2\sigma_{t-1}$ in Lemma 6.3, assuming $\sigma_0 = 0$. At the fixed point of $\sigma_t = 0.25(\frac{1}{1 - \sigma_{t-1}})$, $4\sigma(1 - \sigma) = 1$. Solving for the equation of $4\sigma^2 - 4\sigma + 1 = 0$, the fixed point value of $\sigma_\infty = 0.5$ is found. If $f(x) = \frac{1}{4(1-x)}$, $f'(x) = \frac{1}{4(1-x)^2}$, and $f'(0.5) = 1$. Thus, the sequence of $\sigma_0, \sigma_1, \cdots$ converges to the fixed point of 0.5.

**Proof of Lemma 6.4 (Sketch).** Computing pattern score requires $\gamma|D_s|$ times of subgraph isomorphism tests to find *scov* ($O(|V_{max}|!|V_{max}|)$ [14]) where $G_{max} = (V_{max}, E_{max})$ is the largest data graph. In addition, $|\mathcal{P}|$ times of graph edit distance computation ($O(|V_{P_{max}}|^3)$ [32]) where $P_{max} = (V_{P_{max}}, E_{P_{max}})$ is the largest canned pattern. Hence, worst case time complexity is $O(|V_{max}|!|V_{max}||D_S|\gamma + |\mathcal{P}||V_{P_{max}}|^3)$. Swap-based pattern maintenance has to store the existing canned pattern set, the set of FCPs and indices $I_{FCT}$ and $I_{IFE}$. In the worst case, storing canned patterns require $O(\gamma(|V_{P_{max}}| + |E_{P_{max}}|))$ space. Storage of PCPs takes $O(|C_\psi| \sum_{i=\eta_{min}}^{\eta_{max}} \times i)$ space in the worst case when each cluster in $C_\psi$ yields a FCP for the size range $[\eta_{min} - \eta_{max}]$. The largest storage of the indices is due to TG matrix $(|D \oplus \Delta D| \times (|F_{D \oplus \Delta D}| + |E'_{freq}|))$ assuming that each entry in the matrix is non-zero. Hence, worst case space complexity is $O(\gamma(|V_{P_{max}}| + |E_{P_{max}}|) + |D \oplus \Delta D| \times (|F_{D \oplus \Delta D}| + |E'_{freq}|) + (\eta_{min} + \eta_{max})|C_\psi| \frac{\eta_{max} - \eta_{min} + 1}{2})$.

## B MAINTENANCE OF BASIC PATTERNS

The CATAPULT GUI described in [21] contains edges and 2-edges as *basic patterns* (denoted as $\mathcal{B}$). In this section, we shall describe their maintenance. In particular, MIDAS shall store the support level of each edge and 2-edge in the dataset. Note that the 2-edge $(e_1(u, v), e_2(v, w))$ support values can be represented as a matrix where the row and column represents $e_1$ and $e_2$, respectively. Similar to $I_{IFE}$ and $I_{FCT}$, MIDAS only store the non-zero entries in the matrix to reduce space consumption.

The basic edge patterns can be easily updated by taking the following step:

(1) Update support level of each edge $e$ following graph modification. That is, $\forall G^+ = (V^+, E^+) \in \Delta^+$ (resp. $G^- = (V^-, E^-) \in \Delta^-$), increment (resp. decrement) support of $e$ by 1 if $e \in E^+$ (resp. $e \in E^-$).
(2) Rank the edges in decreasing level of support in the dataset. The ranked list is denoted as $E_r$.

(3) Compute the number of steps ($step_e(e_1)$) required to draw $e_1$ on Panel 2 using edge-at-a-time approach where $e_1 \in E_r$ is the edge with the highest level of support.

(4) Compute the minimum number of steps ($minStep_p(e_1)$) required to draw $e_1$ on Panel 2 using pattern-at-a-time approach. Note that patterns in Panels 3 and 4 are utilized here and edge deletion is allowed.

(5) Select $e_1$ as a basic pattern if $step_e(e_1) < minStep_p(e_1)$ and $|\mathcal{B}_{edge}| < \alpha$ where $\mathcal{B}_{edge}$ is a set of basic edge pattern and $\alpha$ is the maximum number of such patterns allowed in Panel 3. Remove $e_1$ from $E_r$.

(6) Repeat Steps 2 to 4 until $|\mathcal{B}_{edge}| = \alpha$.

Steps for updating the basic 2-edge patterns are the same.