2023 IEEE 30th International Conference on High Performance Computing, Data, and Analytics (HiPC) | 979-8-3503-8322-5/23/$31.00 ©2023 IEEE | DOI: 10.1109/HIPC58850.2023.00040

Graph Pattern Mining Paradigms: Consolidation and Renewed Bearing

Vinicius Dias[‡], Samuel Ferraz[*†], Aditya Vadlamani[§], Mahdi Erfanian[§], Carlos H. C. Teixeira[*]
Dorgival Guedes[*], Wagner Meira Jr.[*], Srinivasan Parthasarathy[§]
[‡] *Departamento de Ciência da Computação, Universidade Federal de Lavras (UFLA) – Lavras, Brazil*
Email: viniciusdias@ufla.br
[*] *Departamento de Ciência da Computação, Universidade Federal de Minas Gerais (UFMG) – Belo Horizonte, Brazil*
Email: {samuel.ferraz, carlos, dorgival, meira}@dcc.ufmg.br
[†] *Faculdade de Computação, Universidade Federal de Mato Grosso do Sul (UFMS) – Campo Grande, Brazil*
Email: samuel.ferraz@ufms.br
[§] *Department of Computer Science and Engineering, The Ohio State University (OSU) – Columbus, USA*
Email: {vadlamani.12, erfaniandabbaq.1}@osu.edu, srini@cse.ohio-state.edu

***Abstract*—Graph Pattern Mining (GPM) refers to a class of problems involving the processing of subgraphs extracted from larger graphs. Applications to GPM algorithms include querying subgraphs, identifying motif structures in biological networks, characterizing social media, among others. GPM algorithms are challenging to develop due to subroutines that include non-trivial graph theory concepts and methods such as isomorphism. General-purpose GPM systems have emerged as a solution to improve the user experience with such algorithms. However, existing general-purpose GPM systems are heterogeneous in terms of implementation details, hardware environment and algorithmic paradigms for subgraph exploration and thus, observations taken from the experimental results alone may not clearly identify when a particular paradigm prevails over another. In this work we present an experimentation analysis of popular paradigms used in existing GPM systems. In order to provide a fair and comprehensive evaluation of various algorithmic paradigms we implement all of them within a single GPM framework. Our results show that no single paradigm is best for every application scenario, and we believe that our findings may guide practitioner towards more optimized GPM systems in the future.**

***Index Terms*—graph pattern mining, experimental evaluation**

I. Introduction

Graph pattern mining (GPM) refers to a class of problems marked by the processing of subgraphs extracted from larger graphs. The relevance of GPM computation includes applications such as motif extraction from biological networks [1], frequent subgraph mining [2], [3], [4], subgraph searching over semantic data (e.g., RDF) [5], social media network characterization [6], [7], community discovery [8], periodic community discovery [9], temporal hotspot identification [10], identification of surprising dense subgraphs in social networks [11], link spam detection [12], financial fraud detection [13], recommendation systems [14], graph learning [15], among others. GPM algorithms are complex to design and to deploy since they usually involve handling an exponential number of subgraph candidates (combinatorial explosion) and non-trivial routines for grouping or enumerating subgraphs based on their equivalence classes (isomorphisms). As a result, general-purpose GPM systems have emerged as a viable solution that facilitates programmer productivity and efficient performance.

The space of existing general-purpose GPM systems is diverse. In one dimension, some systems are designed for multi-threaded or distributed settings [16], [17], [18], [19], others are designed for emerging GPU architectures [20], [21], [22], and yet others for hardware accelerators [23], [24]. In a second dimension, two main alternative GPM paradigms are adopted by GPM systems and are responsible for ensuring an efficient search-space exploration of subgraphs: (i) the *pattern-oblivious subgraph enumeration* paradigm (POSE) in which no pattern information is given to guide subgraph enumeration and (ii) the *pattern-aware subgraph enumeration* paradigm (PASE) in which subgraph enumeration is performed by matching candidate patterns (templates) against the input graph. Figure 1 shows subgraphs with 3 vertices being explored using each paradigm. While POSE relies on some *canonical filter* [18] function to enumerate the distinct subgraphs within a graph, PASE accomplishes the same task by generating patterns of interest (templates) first and matching each against the graph using some strategy derived from Ullmann's algorithm [25] and accompanied with *symmetry breaking* [26] conditions to ensure the same set of distinct subgraphs. Such a range of implementations in multiple architectures (first dimension) spanning various algorithmic paradigms (second dimension) leads to a lack of understanding on the source of the performance gains, and it also complicates the evaluation of new and existing optimizations. To partially redress this situation we present a wide evaluation of existing GPM paradigms and application scenarios over a single distributed framework model (i.e. fixing the first dimension) to both consolidate the collective knowledge about existing efforts and to identify promising directions for future work, especially along the second dimension listed above.

Two main challenges complicate the task of evaluating GPM paradigms. First, because GPM algorithms built over general-purpose systems often include user-specific semantics, multiple algorithms or implementations to solve a task may

2640-0316/23/$31.00 ©2023 IEEE
DOI 10.1109/HiPC58850.2023.00040

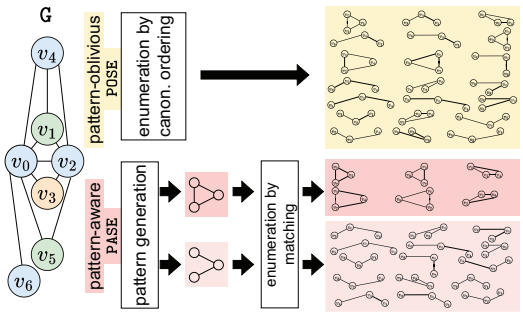


Fig. 1. Alternative paradigms used in GPM systems.

exist. Specifically, this means that our experimental methodology must accommodate both pattern-aware and pattern-oblivious paradigms within the same system implementation and underlying application model. This is non-trivial since most existing GPM systems do not support both paradigms. Most existing GPM systems adopt either pattern-oblivious (POSE) [18], [19] or pattern-aware (PASE) [27], [28], [24] for their enumeration engine, and the few that support both paradigms fail to provide a clean way to combine paradigms to understand the impacts of task specification (usually via a domain specific programming language), architecture and underlying optimizations [29]. Hence, existing GPM systems are not ideal for a systematic evaluation since a direct comparison may not distinguish whether the performance differences are explained by the GPM paradigm or the implementation details.

Second, in order to identify the trade-offs between alternative GPM paradigms, one must consider a diverse set of applications and scenarios. Moreover, an experimental evaluation with these goals must not be limited by reporting comparative performance measurements, instead it must be able to provide a comprehensive and insightful diagnostic. Existing works on general-purpose GPM provide only a narrow perspective of possible application scenarios [17], [18], [16], [20], [21] and lack a deep understanding on the sources of performance discrepancies making it difficult to understand when one paradigm may be preferred over the other.

In this work we propose a performance study of general-purpose GPM applications and existing paradigms. Our evaluation allows identifying inherent trade-offs between GPM paradigms and also unveils the opportunities for performance optimization. Our contributions are summarized as follows:

1. A modular and extensible model for representing and evaluating GPM applications: We propose a formal description of general-purpose GPM algorithms, unveiling important building blocks for standardized application design. We also present a rigorous taxonomy of GPM applications to enable a more consistent reasoning about the system performance on different types of tasks and applications, allowing new emerging performance-driven optimizations and existing GPM paradigms to be more effective. Our model is implemented as an extension to the existing GPM system Fractal [17].

2. Experimental evaluation of GPM paradigms: We provide an extensive experimental evaluation of GPM workloads, including a wide range of application scenarios, algorithms and real-world datasets. Our evaluation exposes the trade-offs between standard GPM paradigms (pattern-oblivious and pattern-aware) and show how these trade-offs may be exploited for choosing the most adequate paradigm for a given workload scenario. To our knowledge this is the first work to extensively evaluate the various use case scenarios for GPM workloads and to present a comparative performance diagnosis of paradigm alternatives on said scenarios.

3. Opportunities for optimization of GPM algorithms: Our experimental setting and general-purpose modeling expose key opportunities for deploying existing and new optimization strategies for both existing and future GPM systems. We also provide a discussion on how to identify opportunities of optimization given the characteristics of specific workloads.

II. Background

Without loss of generality we adopt in this work an undirected input graph G with edges and vertices which may have multiple labels (Definition 1).

Definition 1: **(Graph)** A graph G is represented by a set of vertices $V(G)$, edges $E(G)$, vertex labels $L(G)$, and one map function f_L. Each edge $e = (u, v) \in E(G)$ connects a pair of vertices u and $v \in V(G)$. The edges are not directed and there are no self-loops in G. Formally, $(v_i, v_j) = (v_j, v_i)$ and $(v_i, v_i) \notin E(G)$. The labels of a vertex are defined according the function $f_L : V(G) \rightarrow \mathbb{P}(L(G))$ (power set).

A **subgraph** is represented by a set of vertices and edges embedded in the input graph G and in this work we are interested only in *connected subgraphs*. Some GPM applications may also be interested in enumerating **induced subgraphs**. The edges of an induced subgraph S comprises all existing edges from G among vertices in S.

Subgraphs extracted from G may exhibit the same structure and labeling information. We say that such subgraphs belong to the same equivalence class and that they are *isomorphic* to each other. Graph isomorphism is the problem of verifying whether two (sub)graphs have an identical structure (topology) and labeling information. Thereby, each subgraph can be mapped to a direct representation of its structural and labeling information, referred simply as **pattern** (Definition 2). Thus, a pattern $\rho(S)$ is a template for a subgraph S and, thus, a subgraph is an instance of its pattern.

Definition 2: **(Pattern)** Given a subgraph S of graph G, the pattern of S is a set of pattern edges $\rho(S)$ such that $(u, v) \in E(S)$ iff $(\pi(u), f_L(u), \pi(v), f_L(v)) \in \rho(S)$, where π is an isomorphism between S and $\rho(S)$.

In fact, two (sub)graphs G and H in the same equivalence class have the same **canonical pattern**, a unique representation for each pattern. In this work, we adopt the widely accepted Bliss algorithm [30] to determine the canonical labeling of a labeled (sub)graph S, denoted by $\rho_c(S)$. The enumeration of subgraph instances (not patterns) is also related to isomorphisms. In fact, any permutation of vertices and edges

represents an enumeration ordered code for the *same* subgraph instance in G. Consequently, GPM systems strict themselves to enumerating only a single **canonical representative code** for each subgraph to prevent redundant work.

A. Graph pattern mining problems

Let G be input graph and $\mathbb{S} = \{S_1, \ldots, S_n\}$ be the set of all distinct subgraphs in G. We define the problems solved by general-purpose GPM systems as the aggregation of subgraphs of interest enumerated from G according to a predicate C and a given size k (depending on the context, the size of a subgraph can be its number of vertices or edges). The predicate can be defined over any property that may be obtained from subgraphs in G such as labeling, density, structure, to cite a few. The aggregation can be any function that generates an output given the set of subgraphs of interest (e.g. counting, listing, averaging, verifying frequency). This work focuses on identifying the impact of GPM paradigms (PASE and POSE) on performance and most problems described next assumes *listing/counting* as standard aggregation routine.

1. Pattern querying (ρ-PQ): Querying a subgraph pattern is useful to graph databases and as core operation of pattern-aware GPM systems [16], [27]. The task is to list and count all the subgraphs in an input graph G isomorphic to a user-defined pattern ρ, i.e., the predicate $C(S)$ is satisfied iff $\rho(S) = \rho$.

2. Frequent subgraph mining (k-FSM-α): A Frequent Subgraph Mining (FSM) kernel seeks to obtain all frequent not induced patterns and subgraphs from a labeled input graph G. A pattern P is frequent if it has a support $s(P) \in \mathbb{S}$ above a threshold α, *i.e.*, if $s(P) \geq \alpha$. We adopt the *minimum image-based support* [31] as the support function $s(\cdot)$ to leverage the anti-monotonic property: larger frequent patterns can only be obtained from smaller also frequent patterns. We compute the not induced frequent patterns and subgraphs with k edges ($\mathcal{F}_k$) from frequent patterns with $k-1$ edges ($\mathcal{F}_{k-1}$), i.e., the predicate $C(S)$ is satisfied iff $p \in \mathcal{F}_{k-1}$ and $\rho_c(S) \supset p$.

3. Quasi-cliques (k-QC-α): Dense subgraph extraction can assist in fraud detection for social networks [32], in unveiling structural correlations for attributed graphs [33], among others. The problem seek to list and count α-quasi-cliques in a graph. A α-quasi-clique of size k is an induced subgraph that has k vertices and each of them is connected to a fraction of the vertices in the subgraph, i.e., the predicate $C(S)$ is satisfied iff for each $v \in V(S)$, $degree(v) \geq \lceil \alpha * (|V(S)| - 1) \rceil$.

4. Query specialization (ρ-QS): Given a pattern query ρ the goal of query specialization [34], [35] is to unveil new queries that are specializations of ρ, i.e., larger queries containing ρ. The major procedure in unveiling query specializations is to list and to count not induced subgraphs that are isomorphic to specializations of query ρ. We consider specializations containing pattern ρ and an additional edge, i.e., the predicate $C(S)$ is satisfied only for a subgraph S in which pattern contains ρ and has $|E(\rho)|+1$ edges ($\rho \subset \rho(S)$).

5. Label search (k-LS-$\mathcal{L}$): Graph databases (e.g. Neo4j) are often represented by entities (vertices) that are related among themselves (edges). Vertices may carry labeled semantics representing roles or types in the database schema. The goal of label-based subgraph search is to extract relevant induced subgraphs with k vertices from a larger input graph according to labels of interest ($\mathcal{L}$), i.e., the predicate $C(S)$ is satisfied only for subgraphs S in which every vertex label exists in the input label set ($\forall u \in V(S), L(u) \subseteq \mathcal{L}$).

III. Modeling GPM algorithms

In this work we extend a primitive-based model for general-purpose GPM [17] and adopt the same model to represent each algorithm evaluated in our experimental analysis, allowing a fair and comprehensive comparison among strategies and paradigms. Specifically, our alternative algorithms for solving the GPM problems are modeled through four primitives: *extension* (E), *filtering* (F), *aggregation* (A), and *mapping* (M). The first two primitives allow default implementation of the standard GPM paradigms (PASE and POSE). The third is straightforward since is inherent of each application scenario and unrelated to enumeration paradigms and thus, we omit its discussion in this work. The last primitive is an extension proposed in this work to enable the combination of both paradigms in the same algorithm (detailed in Section III). Figure 2 illustrates the operation of these primitives.

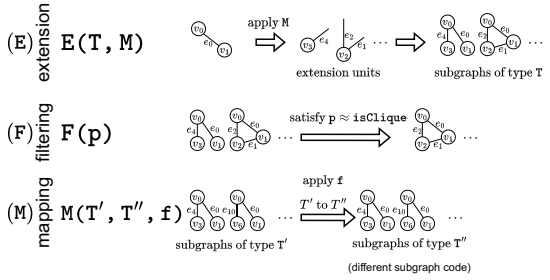


Fig. 2. Modeling GPM algorithms via primitives. Primitives (E, F, A) can be sequenced (regex GE(E + F + M)*A) to represent an application step. Aggregation (not shown) can be any output function over subgraphs.

Extension primitive: This primitive generates the search space of subgraphs by receiving a set of subgraphs as input, extending each by using their own neighborhood in G, and producing a set of larger subgraphs (Figure 2). Naturally, it is possible to extend subgraphs using different approaches, depending on the type of subgraphs targeted and the algorithm used for extending them. We denote an extension primitive by **E(T,M)**, where T is the *type* of subgraphs (i.e. the subgraph code definition) used during enumeration while M is the *method* (or algorithm) that produces extension units from a subgraph. Next we describe how an extension primitive may be configured to represent pattern-oblivious (adopted by GPM systems such as Arabesque [18]) and pattern-aware (adopted by GPM systems such as Peregrine [16] and Automine [27]).

- $\texttt{E}(\texttt{T}_\texttt{V}, \texttt{M}_\texttt{C})$*:* extension step towards the enumeration of all unique induced subgraphs ($\texttt{M}_\texttt{C}$) vertex-by-vertex ($\texttt{T}_\texttt{V}$). Subgraph codes are ordered vertices of no particular pattern and

thus, this process uses a *pattern-oblivious paradigm* (POSE). Figure 2(E) shows an example of this primitive's operation.

• $\texttt{E}(\texttt{T}_\texttt{E}, \texttt{M}_\texttt{C})$*:* extension step towards the enumeration of all unique subgraphs ($\texttt{M}_\texttt{C}$) edge-by-edge ($\texttt{T}_\texttt{E}$). Subgraph codes in this case are ordered edges of no particular pattern and thus, this process also uses *pattern-oblivious paradigm* (POSE).

• $\texttt{E}(\texttt{T}_\texttt{P}(\rho), \texttt{M}_\texttt{P}(\rho))$*:* extension step towards the enumeration of all unique subgraphs that match to a specific pattern ρ ($\texttt{T}_\texttt{P}(\rho)$) using some method derived from Ullmann's algorithm [25] and thus, using a *pattern-aware paradigm* (PASE).

Filtering primitive: The filtering primitive is used to prune subgraphs that do not meet a few application criteria, which are user-defined. In this work we denote a filtering primitive by **F(p)**, where p represents a predicate used to determine whether a subgraph is considered valid and apt for downstream processing. Figure 2(F) shows an example of filtering out subgraphs that do not represent a clique.

Mapping primitive: The *mapping primitive*, denoted by $\texttt{M}(\texttt{T}', \texttt{T}'', \texttt{f})$, allows GPM algorithms to accommodate different extension types in the same subgraph exploration task. We may accomplish this by converting a code associated to some extension type T' into another code that represents the same subgraph, but is associated to an alternative extension type T'' via function f. In As we may see in Section III-A (and Figure 4), this enables hybrid GPM algorithms to combine GPM paradigms (PASE and POSE). In the example depicted in Figure 2, two pattern-oriented ($\texttt{T}_\texttt{P}(\rho)$) subgraphs are mapped to their edge-oriented ($\texttt{T}_\texttt{E}$) canonical representation.

A. GPM algorithm design

Our model comes down to specifying sequences of primitives along with their respective parameters that represent the algorithm semantics. A GPM *application step* is denoted by a string of primitives applied to a given enumeration graph that starts with an extension primitive, proceeds with any sequence composed of extension, filtering, or mapping primitives, finishing with an aggregation primitive that produces the output (regex $GE(E+F+M)^*A$). The GPM algorithms described next may be composed of one or many application steps denoted as $G \cdots A$, meaning that within such steps subgraphs are enumerated from G using some extension/filtering strategy and aggregated (A) to produce some output. This design enable a more coarse-grained view of GPM algorithms and allow us to focus on the fundamentals of the paradigms, i.e., how they map enumeration work into application steps. For clarity, we also omit the primitive parameters. We believe this is the first attempt to model GPM algorithms in a way that is independent of system implementation, that is concise in terms of the operators needed to represent different strategies, and that is modular in terms of how easily the operators can be combined to solve complex graph analytics routines.

Table I summarizes how the algorithms evaluated in this work are categorized. *Single-pattern* represent GPM routines in which the subgraphs of interest share a same structural pattern whereas *multi-pattern* represent GPM routines in which subgraphs of interest may span multiple structural patterns. Therefore, among multi-pattern algorithms we consider whether the subgraphs of interest are selected based on a *pattern-driven filter* (e.g., finding subgraphs meeting a density threshold) or based on a *label-driven filter* (e.g., searching for subgraphs containing a few labels of interest).

Throughout these categories and for each problem we consider a few algorithm variants, two of which represent the paradigms evaluated in this study (pattern-oblivious and pattern-aware) and others representing alternatives used to capture promising directions. As we may see in Section V, existing works are limited in the amount of applications considered and most importantly, do not provide a multi-paradigm perspective to their experimental study. Next we present each algorithm variant evaluated in this work.

TABLE I
PROBLEMS, CATEGORIES AND ALGORITHMS EVALUATED.

pattern querying (ρ-PQ)	Single pattern	POSE, PASE
freq. sub. mi. (k-FSM-α) quasi cliques (k-QC-α) query spec. (ρ-QS)	Multi-pattern, Pattern-driven filter	POSE, PASE, PASE+POSE
label search (k-LS-$\mathcal{L}$)	Multi-pattern, Label-driven filter	POSE, PASE, POSE+GF

Key notations:
E: holds an extension primitive with parameters – $\texttt{E}(\texttt{T}, \texttt{M})$
F: holds a filter primitive with parameters – $\texttt{F}(\texttt{p})$
M: holds a mapping primitive with parameters – $\texttt{M}(\texttt{T}', \texttt{T}'', \texttt{f})$
$GE \cdots A$: app. step over graph G (modeled via primitives)
VPATTERNS-IND(k): induced patterns with k vertices

(a) *POSE* alg. for k-FSM-α

```
1: E ← E(T_E, M_C)
2: P' ← GEA
3: P_f ← FREQ-PATTERNS(P')
4: output P_f
5: for i ← 2 to k do
6:   F ← F(patternIsFreq(α, P_f))
7:   P' ← GE_1F_1 ··· E_{i-1}F_{i-1}E_iA
8:   P_f ← FREQ-PATTERNS(P')
9:   if P_f = ∅ then
10:    break
11:  output P_f
```

(b) *PASE* alg. for k-QC-α

```
1: P ← VPATTERNS-IND(k)
2: P' ← {ρ ∈ P | DENSTY(ρ) ≥ α}
3: for ρ in P' do
4:   E ← E(T_P(ρ), M_P(ρ))
5:   output GE_1 ··· E_kA
```

(c) *PASE+POSE* alg. for ρ-QS

```
1: E' ← E(T_P(ρ), M_P(ρ))
2: E'' ← E(T_E, M'_C) // unique edges
3: M ← M(T_P(ρ), T_E, f)
4: output GE'_1 ··· E'_{|V(ρ)|}ME''A
```

(d) *POSE+GF* alg. for k-LS-$\mathcal{L}$

```
1: E ← E(T_V, M_C)
2: F ← F(labelsSubsetOf(L))
3: G' ← labelsSubsetOf(L)(G)
4: output G'E_1F_1 ··· E_kF_kA
```

Fig. 3. Algorithm design examples.

POSE: This algorithm variant represents standard pattern-oblivious design found in systems such as Arabesque [18]. The central property of such algorithms is that they gather in a *single application step* the enumeration of subgraphs representing *different patterns*, which tends to produce more coarse-grained execution tasks. Figure 3a shows an example of POSE algorithm for FSM. On each iteration of the algorithm, responsible for mining larger frequent patterns,

a single step is submitted (line 7) to enumerate subgraphs of various patterns. This is accomplished by maintaining a pool of frequent patterns used to prune the search space via filtering primitives to discover even larger frequent patterns.

PASE: This algorithm variant represents standard pattern-aware design found in systems such as Peregrine [16] and Automine [27]. The central property of such algorithms is that they submit *one application step per pattern*, meaning that each step is responsible for only a particular subset of subgraphs. Unlike POSE, PASE produces more fine-grained execution tasks. Because the number of patterns grows exponentially with the subgraph size, this may be a problem for larger patterns. Figure 3b shows an example of PASE algorithm for quasi cliques, with one step per pattern (line 5).

PASE+POSE: If on one hand PASE produces an exponential number of steps, its subgraph exploration allows more efficient implementations because we may rely on fast set intersection routines to speedup the generation of extension candidates [16]. POSE is the opposite: we generate fewer application steps but at the cost of a less effective subgraph enumeration. Thereby, we explore in this work a hybrid design that could capture the best features of both: fewer steps plus more efficient subgraph enumeration. We explore this idea for multi-pattern algorithms with pattern-driven filter as they admit reasoning about different ways one could reach subgraphs of various patterns. In particular, a GPM algorithm may ensure a pattern-aware subgraph exploration in the first extension primitives and switch to a pattern-oblivious on the fly, which allows an unconditional exploration of larger subgraphs. The main challenge concerning this hybrid approach is that canonical codes for subgraphs differ depending on the GPM paradigm: symmetry breaking [26] is used for pattern-aware paradigm while canonical filtering [18] is used for pattern-oblivious paradigm. Thus, function `f` denotes a mapping process that transforms a subgraph represented through a pattern-aware code (obtained by $\texttt{M}_\texttt{P}(\rho)$ method) into the same subgraph but represented through an equivalent pattern-oblivious representation (obtained by $\texttt{M}_\texttt{C}$ method). In Figure 4 we show how to enumerate subgraphs with four edges that contain a tailed triangle ρ by (i) matching a triangle subgraph isomorphic to ρ and (ii) adding the fourth edge to the matched subgraph resulting in multiple tailed triangles. In this case, up to three vertices, the subgraph is being enumerated using a PASE framework with extension type $T_P(\rho)$ and, afterwards, the last edge is included using a POSE framework with extension type T_E. This is only correct because of the mapping primitive that includes function f responsible for *translating* a $T_P(\rho)$ subgraph code into a T_E subgraph code, which introduces consistency between canonical subgraph codes. Figure 3c shows an example for query specialization: the algorithm matches the query pattern ρ using pattern-aware ($\texttt{E}'$), switches to pattern-oblivious (primitive `M`), and finishes enumeration by adding an additional edge to the subgraph ($\texttt{E}''$) – producing subgraphs containing pattern ρ.

POSE+GF: Label-driven filter conditions defined in terms of subgraphs can be pushed down to the data source via

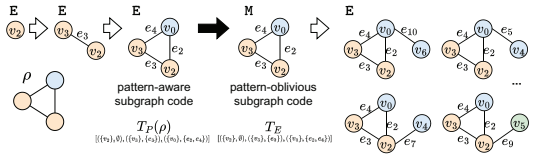


Fig. 4. Hybrid PASE+POSE via mapping primitive.

graph filtering (GF) [17] whenever predicate `p` of primitive $\texttt{F}(\texttt{p})$ is *anti-monotonic*: if `p` is false for some subgraph S, then it also must be false for any subgraph S' extended from S. Instead of repeatedly generating invalid subgraphs and filtering them during enumeration, the algorithm applies this filtering directly to the input graph once as a previous step. We highlight that this strategy can not be treated as a pre-processing step since different filtering conditions imply in different filtered graphs, leading to nontrivial data management issues for large graphs. Figure 3d shows an example of such algorithm for label search: application steps are submitted over a reduced version of the input graph ($\texttt{G}'$ in line 4).

IV. Experimental evaluation

All experiments, unless otherwise specified, were run on a cluster with 5 machines, each one having two CPU Intel Xeon E5-2695v2 Ivy Bridge 2,4GHZ (12 cores each, 24 per machine), 64GB RAM, running RedHat Linux 7.6. The machines are connected by Infiniband FDR (56Gb/s). All the algorithms were implemented within the Fractal [17] framework. We extended Fractal to support the novel mapping primitive that allows combining paradigms into the same application step. The source code and data are made available at https://github.com/dccspeed/fractal.

Datasets (Table II): The datasets used in our evaluation have been widely used previously to evaluate graph mining algorithms and systems [18], [16], [36], [27]. Mico [2] is a co-authorship network, Patents [37] models the citations of patents published in the US, LiveJournal and Orkut [38] model friendship in social networks, and Youtube [39] models posted videos and how they are related. In our experiments all graphs are loaded into the memory of workers using a compressed sparse row graph representation (CSR).

TABLE II
Real-world datasets used in the experiments.

	$\lvert V(G)\rvert$	$\lvert E(G)\rvert$	max.deg.	avg.deg.	labels
Mico (MI) [2]	100K	1M	1.3K	22.3	29
Patents (PA) [37]	2.7M	13.9M	789	10.1	37
LiveJournal (LJ) [38]	3.9M	34.6M	14.8K	17.3	-
Youtube (YO) [39]	4.5M	43.9M	2.5K	19.1	108
Orkut (OR) [38]	3.07M	117.1M	33.3K	76.2	-

Performance evaluation measures: We consider a time budget of 5 hours for each execution, which allows us to study larger and more diverse GPM paradigms. Within each execution, to be fair in our comparison, we divide the time budget amongst application steps in a way that lighter steps (concerning more dense and infrequent patterns in scale-free

networks) are scheduled first so more time budget is left for heavier steps. We consider two evaluation metrics: (1) the *runtime* whenever it is not trivial to measure the application throughput (particularly, for FSM); and (2) the *normalized throughput* of aggregated subgraphs – in this case, we indicate whether an execution reached the time limit with an asterisk ($*$). Throughput allows us to determine the most efficient strategy given a time budget. These metrics are widely used to evaluate the performance of subgraph querying systems under time constraints [40] and also in the context of streaming analytics where aggregated results are continuously consumed via a publish-subscribe framework [19].

Queries and matching order (Figure 5): For GPM problems such as query specialization (ρ-QS), we generate input patterns for each dataset based on their density. Specifically, we implement a widely known unbiased sampling method for subgraphs [41] to extract representative patterns of a given size k: (1) Sk, sparse pattern with k vertices; and (2) Dk, dense pattern with k vertices. Additionally, we adopt the following heuristic for determining the order in which patterns are matched when using the pattern-aware paradigm. We match vertices having infrequent labels first, if there is a tie match vertices with more symmetry breaking conditions [26]. Finally, if there is still a tie, we match vertices with more backward edges in the pattern so that denser subpatterns are matched first. In our experiments this heuristic is sufficient to ensure that no arbitrarily low quality matching order penalizes the pattern-aware approach. An in-depth study on various subgraph matching approaches including orders can be found at Sun et al. [42] and is out of the scope of this work. For problems such as label search and minimal keyword search, we take a similar approach to produce label sets: (1) Ik, representing k infrequent labels that rarely occur together within a subgraph; (2) Fk, representing k labels that most frequently appearing together within a subgraph.

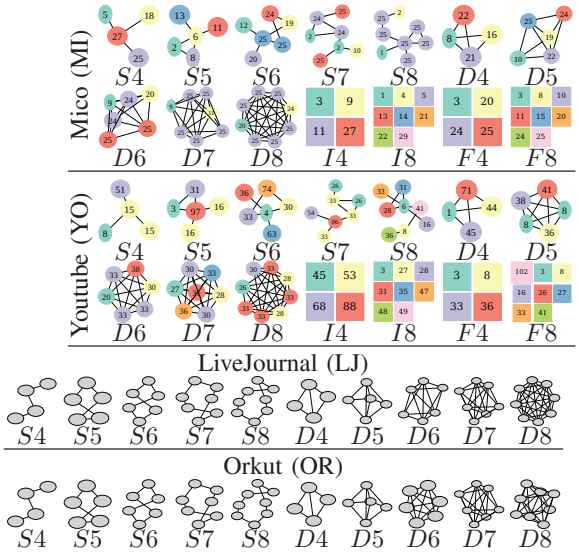
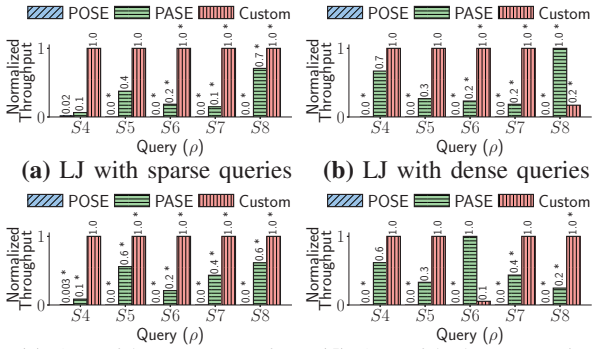


Fig. 5. Queries: both colors and numbers refer to unique vertex labels.

A. Single-pattern algorithms

Pattern querying (ρ-PQ, Figure 6): Overall the POSE strategy is inferior for all configurations considered, with many executions that do not even find one single output subgraph within the time limit. This confirms the natural hypothesis that pattern-aware computation is best whenever the target patterns are known and generated apriori. We also compare default algorithms with optimized custom algorithm MCVC [43], [16] (not to be confused with PASE+POSE in Section III-A), that matches a pattern based on its minimum connected vertex cover, to see how much more efficiency can be obtained from pattern-awareness. Regarding the two remaining alternatives - PASE and Custom/MCVC - that leverage pattern information during subgraph enumeration, we observe that the latter outperforms the former in almost all configurations. The exceptions to this rule happens in $D8$ for LiveJournal, and in $D6$ for Orkut. A careful examination of these results offers the nuanced but interesting finding that the frequency and density of the target pattern alone is insufficient to explain the best alternative. In fact, both of these exceptions represent k-clique patterns and in this case, the minimum vertex cover is maximal ($k-1$ vertices). This makes the custom strategy least effective because matching the cover is tantamount to matching the entire pattern.



(a) LJ with sparse queries **(b)** LJ with dense queries

(c) OR with sparse queries **(d)** OR with dense queries

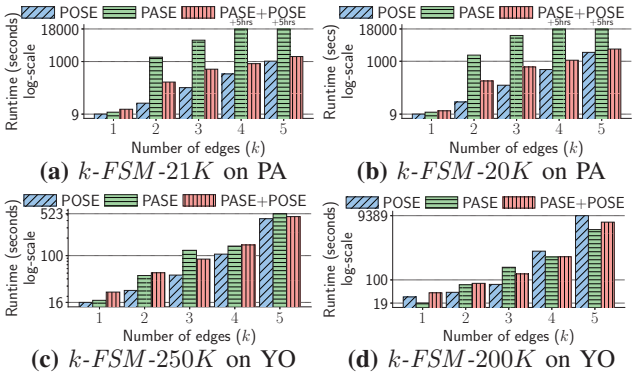
Fig. 6. Throughput: Pattern querying (ρ-PQ).

Opportunities: We see opportunities for the development of adaptable search strategies within GPM systems capable of understanding and learning the most appropriate subgraph exploration paradigm conditioned on pattern input and other related contextual features of the graph.

B. Multi-pattern algorithms with pattern-driven filter

Frequent subgraph mining (k-FSM-α, Figure 7): As the output of FSM is the set of frequent patterns and supports, no trivial measure of execution progress exist and we report the runtime of configurations in which at least one of the alternative algorithms finished within the time limit. Overall, POSE is more effective in the majority of configurations, especially for Patents dataset. An exception occurs in k-FSM-$200K$ on Youtube in which PASE exhibits better performance (Figure 7f). This is directly related to the number of steps required by

the PASE strategy, on those scenarios. For instance, while 3-FSM-20K and 4-FSM-20K on Patents (Figure 7d) require 1529 and 4463 steps, respectively, 3-FSM-200K and 4-FSM-200K on Youtube require much less: 34 and 115 respectively. Not unexpectedly, PASE+POSE represents a middle ground in terms of the number of steps required and its performance lies between the two baseline strategies.

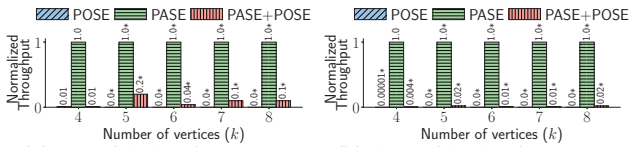


(a) k-FSM-21K on PA **(b)** k-FSM-20K on PA

(c) k-FSM-250K on YO **(d)** k-FSM-200K on YO

Fig. 7. Runtime: Frequent subgraph mining (k-FSM-α).

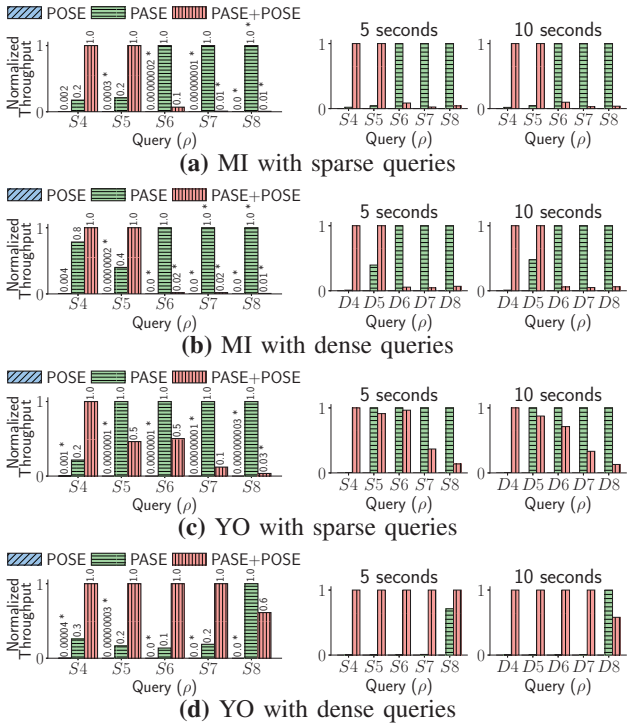
Quasi cliques (k-QC-α, Figure 8): We see a vast superiority of the PASE strategy. Compared to POSE and PASE+POSE, the PASE approach is more efficient in these settings since most queried patterns exist in the input graph and the density threshold can be used to prune application steps concerning sparse patterns, which makes the overhead PASE incurs, payoff. However, as the size of subgraphs increases and the number of pattern candidates in PASE increases exponentially (increasing overhead), it starts evaluating patterns that are not found in the input graph. For instance, in Orkut, PASE submits 5 application steps with no output for 7-QC-0.5 and 41 for 8-QC-0.5, indicating that this quantity tends to become greater and incurs in higher overhead for PASE, even for unlabeled applications such as quasi clique finding.



(a) LJ with density $\alpha = 0.5$ **(b)** OR with density $\alpha = 0.5$

Fig. 8. Throughput: Quasi cliques (k-QC-α).

Query specialization (ρ-QS, Figure 9): The POSE strategy - which relies on an expensive visit-filter operations in order to ensure that the input pattern is contained in the enumerated subgraph - performs poorly on this task. The PASE approach is superior in almost all the scenarios, especially in Mico, which is a very dense dataset with overrepresented labels and patterns. Drilling into the comparative analysis of PASE vs. PASE+POSE (hybrid variant), let us consider the Youtube results for $D7$-QS and $D8$-QS, respectively. In the former, the hybrid variant is more efficient and we observe that 261 of 648 ($\approx 40\%$) steps in PASE finishes with zero output. In the latter, PASE is slightly more efficient and 119 of 542 ($\approx 22\%$) steps in PASE finishes with zero output. A similar behavior can be observed for the other scenarios, which leads us to conclude that there is a correlation between spurious steps (the ones that return zero output) and PASE performance. The hybrid approach, on the other hand, is not prone to this because it matches pattern *ρ first* (i.e. the subpattern), and then extends it with one additional edge, which prevents the runtime from spurious querying steps. Because this behavior depends on the occurrence of each pattern/query in the underlying graph, it is not trivial to determine prior to runtime which algorithm will perform best. Given this challenge, the right hand side of Figure 9 shows how throughput is perceived when the search space exploration of subgraphs is randomized using a single execution thread for 5 and 10 seconds. For the majority of cases, with a small effort we are able to determine which algorithm is prone to exhibit better performance overall. An interesting behavior happens for $D8$-QS on Youtube in which a 5 seconds throughput estimation is not enough to unveil the real dominance of PASE over PASE+POSE, indicating that we may need to sample the search space longer (in this example, 10 seconds) in case of long running application scenarios.



(a) MI with sparse queries

(b) MI with dense queries

(c) YO with sparse queries

(d) YO with dense queries

Fig. 9. Throughput: Query specialization (ρ-QS).

Opportunities: We see significant potential in hybrid subgraph exploration paradigms (such as PASE+POSE) towards mitigating PASE overheads of spurious querying of patterns, especially for larger spaces where the number of patterns becomes exponentially larger. The main insight is

that although hybrid algorithms are no silver bullet for GPM tasks, many application scenarios could really benefit from it, opportunity that most GPM systems supporting either POSE or PASE fail to exploit. It remains a challenge to determine prior to execution which alternative may be the best on each case and black-box heuristic approaches capable of estimating the throughput of algorithms (such as the one presented in Figure 9) may be an effective and lower-cost performance optimization strategy for several application scenarios. We also anticipate that GPM systems would benefit from pragmatic (contextual) knowledge of the input data in order to automatically determine the most appropriate paradigm (or combination of them) for a particular task.

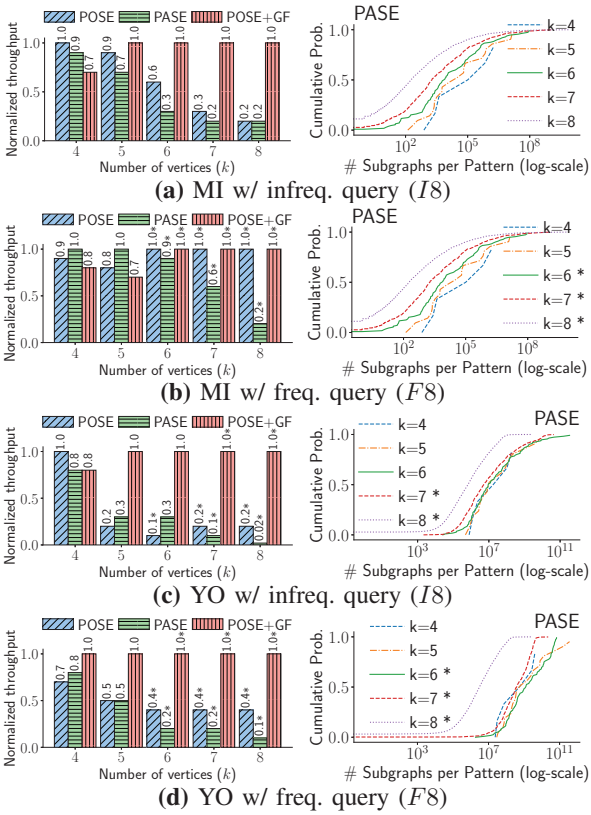
C. Multi-pattern algorithms with label-driven filter

Label search (k-LS-$\mathcal{L}$, Figure 10): For Mico, the PASE strategy exhibits larger overhead for small subgraph sizes since it is modeled as a multi-step application, especially when the amount of work is not substantial (e.g. 4-LS-$I8$). For larger values of k this overhead starts to payoff and the performance becomes roughly equivalent to the POSE approach. PASE also tends to become more inefficient for larger subgraph sizes (e.g. [4-8]-LS-$F8$ in Youtube). Such behavior is explained by the skewness of patterns w.r.t. their frequency – in particular, the algorithm spends much of the assigned time enumerating subgraphs of an individual pattern that represents a very small portion of the total output, affecting the rate at which output is generated. To support this claim we show in the right hand side of Figure 10 the number of subgraphs (output) per pattern found in PASE: greater skew as subgraph size k increases indicates that many small application steps sum up increasingly larger submission overhead and consequently, more substantial drops in performance for larger subgraph sizes. Finally, the hybrid (POSE+GF) approach exhibit substantial improvement over the alternatives. This algorithm is most effective for frequent labels ($F8$), since a larger valid subgraph space (output) tends to exacerbate the overhead and redundancy of filtering routines, applied on every enumeration level. On Youtube - a larger dataset - graph reduction produces a very interesting effect of reducing the working set at runtime, improving memory efficiency and caching.

Opportunities: We see room for several novel strategies. First scaling PASE for larger subgraph sizes. Second, taming the exponential growth of pattern search through improved scheduling and/or hybrid approaches combining PASE and POSE to reduce the number of submitted tasks and their skewness. Third, we believe that understanding the relationships between label-driven filters and subgraph patterns may be used to develop hybrid strategies to improve pruning efficiency by preventing the submission of spurious application steps.

V. Related work

Understanding graph mining computation: GraphMineSuite [44] is a benchmark for evaluating graph mining algorithms. The system provides a set of tools and methodologies for evaluating and tuning properties and behaviors of specific



(a) MI w/ infreq. query ($I8$)

(b) MI w/ freq. query ($F8$)

(c) YO w/ infreq. query ($I8$)

(d) YO w/ freq. query ($F8$)

Fig. 10. Throughput: Label search (k-LS-$\mathcal{L}$).

graph mining algorithms (i.e., not necessarily related to pattern mining) implemented over a standard set algebra. The authors provide use cases for *multi-pattern algorithms with pattern-driven filter* only but not from a multi-paradigm perspective. Indeed, the scope is not on abstractions and programming expressiveness for graph pattern mining systems. Besides that, the specificity of each algorithm and the lack of standard strategies for searching the space of subgraphs makes the task of reasoning about different paradigms not trivial, which justifies our approach of evaluating GPM algorithms.

Subgraph enumeration for general-purpose GPM systems has been explored in detail in the context of pattern-aware frameworks. Pattern analysis [28] can be used as an effective tool for optimizing exploration plans of querying patterns. The pattern information about subgraphs of interest is used to reuse computation and to reduce the depth of enumeration. Dryadic [45] is a graph pattern mining system that proposes an intermediate state representation to pattern matching, which allows a more systematic reasoning about optimized exploration plans. While these systems focus on how to optimize exploration plans for pattern-aware computations, our work takes a step back and handles a more fundamental challenge concerning the trade-offs between pattern-aware and pattern-oblivious paradigms on real problems. Also, they only consider a subset of application scenarios in their experimental evaluation, missing *multi-pattern algorithms with label-driven filter*,

so important for graph databases (e.g. Neo4j).

Pattern-oblivious GPM systems: Arabesque [18] represents the first generation of GPM systems that operate on a subgraph-centric model for graph processing with focus on programming productivity. On the other end of the spectrum, G-Miner [36] is a MPI-based C++ framework for graph pattern mining and subgraph exploration. Unlike the above, G-Miner takes a fine-tuning approach to system design and provide a low-level interface for GPM algorithms at the cost of productivity. Tesseract [19] is a distributed system for incremental graph pattern mining where the input graph may change during computation. While Tesseract's design, based on two user-defined functions *match* and *filter*, is simple enough to accommodate many applications (including static graphs), it lacks the flexibility to provide multi-paradigm algorithm design. Rstream [46] is a join-based single-machine GPM system that leverages out-of-core environments to store intermediate data, which for combinatorial GPM algorithms quickly becomes infeasible to manage in terms of I/O. Pangolin [20] is a GPM system that leverages a breadth-first subgraph exploration strategy to facilitate load balancing and improve data access on GPUs. The materialization of subgraphs in-memory may incur in high memory demand, especially for scarce memory architectures such as GPU. Each of these systems adopt its very particular application model for a very specific execution environment, which prevents a fair comparison among algorithms. In this work we handle this limitation by adopting a more general model for representing GPM algorithms.

Pattern-aware GPM systems: Recently pattern-awareness has been assumed to be standard for efficient GPM computation. Automine [27] is single-machine system for general purpose graph pattern mining programming. The system generates C++ optimized code for subgraph enumeration, given template patterns to be enumerated. Peregrine [16] is a single-machine multi-threaded system for graph pattern mining that uses an execution model centered on the subgraph patterns. In particular, Peregrine expresses any GPM computation as multiple pattern querying routine – in many senses, similar to what Automine [27] proposes. G2Miner [21] is a pattern-aware system optimized with a parallelization strategy and memory allocation schemes specifically designed for GPU architectures and thus, not particularly adequate for evaluation of multiple paradigms on shared-memory distributed systems. This work puts pattern-awareness into perspective and shows that it can not be considered the state-of-the-art design for every application scenario, which may guide practitioners to better understand trade-offs involved in the design of GPM systems.

VI. Conclusions

This work offers a comprehensive analysis of paradigms for subgraph enumeration in the context of general-purpose GPM systems. Our methodology categorizes state-of-the-art GPM strategies, tasks and optimizations in a concise and expressive model for algorithms, allowing a fair and comprehensive comparison among different search and enumeration paradigms. Our wide range of application scenarios considered reveals that there is no silver bullet when it comes to choosing subgraph enumeration paradigms, be pattern-aware or pattern-oblivious. In particular, we confirm that pattern-oblivious is inferior whenever the search space is pruned based on pattern structure, but, on the other hand, pattern-aware suffers from an exponentially increasing overhead of querying individual patterns one at a time, especially in distributed environments. Because these bottlenecks are closely related to increased communication and load imbalance, this can only be exacerbated in larger scale settings. Also, our findings go beyond performance comparison and show promising directions in exploring hybrid multi-paradigm approaches, in proposing scheduling routines specifically designed for the pattern-aware setting, and in leveraging latent pruning conditions that can be applied directly to the input graph or inferred based on pattern and/or labeling information.

Acknowledgment

This work was partially funded by Fapemig, CNPq, CAPES, and by projects CNPq/AWS (440.088/2020-8), INCTCyber (MCT/CNPq 465714/2014-5) and MASWeb (FAPEMIGPRONEX APQ-01400-14). The authors acknowledge the National Laboratory for Scientific Computing (LNCC/MCTI, Brazil) for providing HPC resources of the SDumont supercomputer. SP was partially supported by the NSF AI-Edge grant CNS-2112471, and would like to acknowledge the NSF MRI OAC-2018627 grant for additional research experimentation resources. Any opinions, findings, and conclusions in this material are those of the author(s) and may not reflect the views of the respective funding agency.

References

[1] M. Agrawal, M. Zitnik, and J. Leskovec, "Large-scale analysis of disease pathways in the human interactome," *Pacific Symposium on Biocomputing. Pacific Symposium on Biocomputing*, vol. 23, pp. 111–122, 2018.

[2] M. Elseidy, E. Abdelhamid, S. Skiadopoulos, and P. Kalnis, "Grami: Frequent subgraph and pattern mining in a single large graph," *Proc. VLDB Endow.*, vol. 7, no. 7, pp. 517–528, Mar. 2014.

[3] G. Buehrer, S. Parthasarathy, and Y.-K. Chen, "Adaptive parallel graph mining for cmp architectures," in *Proceedings of the Sixth International Conference on Data Mining*, ser. ICDM '06. Washington, DC, USA: IEEE Computer Society, 2006, pp. 97–106.

[4] G. Buehrer, S. Parthasarathy, and M. Goyder, "Data mining on the cell broadband engine," in *Proceedings of the 22nd Annual International Conference on Supercomputing, ICS*, P. Zhou, Ed., 2008.

[5] S. Elbassuoni and R. Blanco, "Keyword search over rdf graphs," in *Proceedings of the 20th ACM International Conference on Information and Knowledge Management*, ser. CIKM '11. New York, NY, USA: ACM, 2011, pp. 237–242.

[6] L. Kovanen, M. Karsai, K. Kaski, J. Kertész, and J. Saramäki, "Temporal motifs in time-dependent networks," *Journal of Statistical Mechanics: Theory and Experiment*, vol. 2011, no. 11, p. P11005, nov 2011.

[7] X. Yang, S. Parthasarathy, and P. Sadayappan, "Fast sparse matrix-vector multiplication on gpus: Implications for graph mining," *Proc. VLDB Endow.*, vol. 4, no. 4, pp. 231–242, 2011.

[8] A. R. Benson, D. F. Gleich, and J. Leskovec, "Higher-order organization of complex networks," *Science*, 2016.

[9] H. Qin, R.-H. Li, G. Wang, L. Qin, Y. Cheng, and Y. Yuan, "Mining periodic cliques in temporal networks," in *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, 2019, pp. 1130–1141.

[10] Y. Yang, D. Yan, H. Wu, J. Cheng, S. Zhou, and J. C. Lui, "Diversified temporal subgraph pattern mining," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1965–1974.

[11] B. Hooi, K. Shin, H. Lamba, and C. Faloutsos, "Telltail: Fast scoring and detection of dense subgraphs," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 34, no. 04, pp. 4150–4157, Apr. 2020.

[12] Y. I. Leon-Suematsu, K. Inui, S. Kurohashi, and Y. Kidawara, "Web Spam Detection by Exploring Densely Connected Subgraphs," in *2011 IEEE/WIC/ACM International Conferences on Web Intelligence and Intelligent Agent Technology*, vol. 1, Aug. 2011, pp. 124–129.

[13] F. Hoffman and D. Krasle, "Fraud detection using network analysis," sep 2015, patent No. EP2884418A1, Filed September 1st., 2014, Issued June 17th., 2015.

[14] H. Zhao, Y. Zhou, Y. Song, and D. L. Lee, "Motif enhanced recommendation over heterogeneous information network," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, ser. CIKM '19. New York, NY, USA: ACM, 2019, pp. 2189–2192.

[15] C. Meng, S. C. Mouli, B. Ribeiro, and J. Neville, "Subgraph pattern neural networks for high-order graph evolution prediction," 2018.

[16] K. Jamshidi, R. Mahadasa, and K. Vora, "Peregrine: A pattern-aware graph mining system," in *Proceedings of the Fifteenth European Conference on Computer Systems*, ser. EuroSys '20. New York, NY, USA: Association for Computing Machinery, 2020.

[17] V. Dias, C. H. C. Teixeira, D. Guedes, W. Meira Jr., and S. Parthasarathy, "Fractal: A general-purpose graph pattern mining system," in *Proceedings of the 2019 International Conference on Management of Data (SIGMOD)*, 2019.

[18] C. H. C. Teixeira, A. J. Fonseca, M. Serafini, G. Siganos, M. J. Zaki, and A. Aboulnaga, "Arabesque: A system for distributed graph mining," in *Proceedings of the 25th Symposium on Operating Systems Principles*, ser. SOSP '15, 2015, pp. 425–440.

[19] L. Bindschaedler, J. Malicevic, B. Lepers, A. Goel, and W. Zwaenepoel, *Tesseract: Distributed, General Graph Pattern Mining on Evolving Graphs*. New York, NY, USA: Association for Computing Machinery, 2021, p. 458–473.

[20] X. Chen, R. Dathathri, G. Gill, and K. Pingali, "Pangolin: An efficient and flexible graph mining system on cpu and gpu," *Proc. VLDB Endow.*, vol. 13, no. 8, p. 1190–1205, Apr. 2020.

[21] X. Chen and Arvind, "Efficient and scalable graph pattern mining on GPUs," in *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. Carlsbad, CA: USENIX Association, Jul. 2022, pp. 857–877.

[22] S. Ferraz, V. Dias, C. H. Teixeira, G. Teodoro, and W. Meira, "Efficient strategies for graph pattern mining algorithms on gpus," in *2022 IEEE 34th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, 2022, pp. 110–119.

[23] P. Yao, L. Zheng, Z. Zeng, Y. Huang, C. Gui, X. Liao, H. Jin, and J. Xue, "A locality-aware energy-efficient accelerator for graph mining applications," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 895–907.

[24] X. Chen, T. Huang, S. Xu, T. Bourgeat, C. Chung, and A. Arvind, "Flexminer: A pattern-aware accelerator for graph pattern mining," in *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, 2021, pp. 581–594.

[25] J. R. Ullmann, "An algorithm for subgraph isomorphism," *J. ACM*, vol. 23, no. 1, p. 31–42, jan 1976.

[26] J. A. Grochow and M. Kellis, "Network motif discovery using subgraph enumeration and symmetry-breaking," in *Proceedings of the 11th Annual International Conference on Research in Computational Molecular Biology*, ser. RECOMB'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 92–106.

[27] D. Mawhirter and B. Wu, "Automine: Harmonizing high-level abstraction and high performance for graph mining," in *Proceedings of the 27th ACM Symposium on Operating Systems Principles*, ser. SOSP '19. New York, NY, USA: ACM, 2019, pp. 509–523.

[28] K. Jamshidi and K. Vora, "A deeper dive into pattern-aware subgraph exploration with peregrine," *SIGOPS Oper. Syst. Rev.*, vol. 55, no. 1, p. 1–10, Jun. 2021.

[29] X. Chen, R. Dathathri, G. Gill, L. Hoang, and K. Pingali, "Sandslash: A two-level framework for efficient graph pattern mining," in *Proceedings of the ACM International Conference on Supercomputing*, ser. ICS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 378–391.

[30] T. Junttila and P. Kaski, "Engineering an efficient canonical labeling tool for large and sparse graphs," in *Proceedings of the Meeting on Algorithm Engineering & Expermiments*. Philadelphia, PA, USA: Society for Industrial and Applied Mathematics, 2007, pp. 135–149.

[31] B. Bringmann and S. Nijssen, "What is frequent in a single graph?" in *Proceedings of the 12th Pacific-Asia Conference on Advances in Knowledge Discovery and Data Mining*, ser. PAKDD'08. Berlin, Heidelberg: Springer-Verlag, 2008, pp. 858–863.

[32] B. Hooi, H. A. Song, A. Beutel, N. Shah, K. Shin, and C. Faloutsos, "Fraudar: Bounding graph fraud in the face of camouflage," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 895–904.

[33] A. Silva, W. Meira, and M. J. Zaki, "Mining attribute-structure correlated patterns in large attributed graphs," *Proc. VLDB Endow.*, vol. 5, no. 5, p. 466–477, Jan. 2012.

[34] D. Mottin, F. Bonchi, and F. Gullo, "Graph query reformulation with diversity," in *Proceedings of the 21th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, ser. KDD '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 825–834.

[35] P. Yi, B. Choi, S. S. Bhowmick, and J. Xu, "Autog: a visual query autocompletion framework for graph databases," *The VLDB Journal*, vol. 26, no. 3, pp. 347–372, 2017.

[36] H. Chen, M. Liu, Y. Zhao, X. Yan, D. Yan, and J. Cheng, "G-miner: An efficient task-oriented graph mining system," in *Proceedings of the Thirteenth EuroSys Conference*, ser. EuroSys '18, 2018.

[37] B. H. Hall, A. B. Jaffe, and M. Trajtenberg, "The nber patent citation data file: Lessons, insights and methodological tools," National Bureau of Economic Research, Working Paper 8498, October 2001.

[38] J. Yang and J. Leskovec, "Defining and evaluating network communities based on ground-truth," *Knowl. Inf. Syst.*, vol. 42, no. 1, pp. 181–213, Jan. 2015.

[39] X. Cheng, C. Dale, and J. Liu, "Dataset for "statistics and social network of youtube videos"," http://netsg.cs.sfu.ca/youtubedata/, 2008.

[40] M. Han, H. Kim, G. Gu, K. Park, and W.-S. Han, "Efficient subgraph matching: Harmonizing dynamic programming, adaptive matching order, and failing set together," in *Proceedings of the 2019 International Conference on Management of Data*, ser. SIGMOD '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 1429–1446.

[41] S. Wernicke, "A faster algorithm for detecting network motifs," in *Algorithms in Bioinformatics*, R. Casadio and G. Myers, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 165–177.

[42] S. Sun and Q. Luo, "In-memory subgraph matching: An in-depth study," in *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, ser. SIGMOD '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1083–1098.

[43] H. Kim, J. Lee, S. S. Bhowmick, W.-S. Han, J. Lee, S. Ko, and M. H. Jarrah, "Dualsim: Parallel subgraph enumeration in a massive graph on a single machine," in *Proceedings of the 2016 International Conference on Management of Data*, ser. SIGMOD '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1231–1245.

[44] M. Besta, Z. Vonarburg-Shmaria, Y. Schaffner, L. Schwarz, G. Kwasniewski, L.Gianinazzi, J. Beránek, K. Janda, T. Holenstein, S. Leisinger, P. Tatkowski, E. Ozdemir, A. Balla, M. Copik, P. Lindenberger, P. Kalvoda, M. Konieczny, O. Mutlu, and T. Hoefler, "Graphminesuite: Enabling high-performance and programmable graph mining algorithms with set algebra," in *Proceedings of the 47th International Conference on Very Large Data Bases*, 2021.

[45] D. Mawhirter, S. Reinehr, W. Han, N. Fields, M. Claver, C. Holmes, J. McClurg, T. Liu, and B. W. 0002, "Dryadic: Flexible and fast graph pattern matching at scale," in *30th International Conference on Parallel Architectures and Compilation Techniques, PACT 2021, Atlanta, GA, USA, September 26-29, 2021*, J. Lee and A. Cohen, Eds. IEEE, 2021, pp. 289–303.

[46] K. Wang, Z. Zuo, J. Thorpe, T. Q. Nguyen, and G. H. Xu, "Rstream: Marrying relational algebra with streaming for efficient graph mining on a single machine," in *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'18. Berkeley, CA, USA: USENIX Association, 2018, pp. 763–782.