2019 IEEE 35th International Conference on Data Engineering (ICDE)

# Workload-aware Subgraph Query Caching and Processing in Large Graphs

Yongjiang Liang Peixiang Zhao
*Florida State University, Tallahassee, FL* 32306
*liang@cs.fsu.edu* *zhao@cs.fsu.edu*

***Abstract*—A subgraph query $q$ that finds as output all its subgraph-isomorphic embeddings from a data graph $g$ has been core to modern declarative querying in large graphs. In this paper, we address subgraph queries with the availability of query workload information, $W = \{w_1, \ldots, w_n\}$, where $w_i \in W$ is a previously issued query with all its subgraph-isomorphic embeddings cached beforehand. We introduce a *workload-aware* subgraph querying framework, WaSQ, that leverages query workload for subgraph query rewriting, search plan refinement, partial results reusing, and false positive filtering towards facilitating the whole subgraph querying process. Experimental studies in real-world graphs demonstrate that WaSQ achieves significant and consistent performance gains in comparison with state-of-the-art, workload-oblivious solutions for large-scale subgraph querying.**

***Keywords*-graphs; subgraph queries; graph matching.**

## I. Introduction

Fundamental to virtually any graph management tasks is the querying functionality that locates user-specified graph patterns in a data graph, which, without loss of generality, is often modeled as the *subgraph query* problem. Despite NP-hard in nature, subgraph queries have been intensively studied in real-world big graphs. However, existing solutions still suffer from performance and scalability issues, due in particular to the intrinsic subgraph-isomorphism complexity and the sheer sizes of real-world graphs [1, 2, 3]. Surprisingly enough, existing methods are primarily *workload-oblivious*: subgraph queries are naively considered memoryless, irrespective of common and essential patterns that have arisen repeatedly and been addressed, either partially or completely, beforehand. As a consequence, every query has to be handled anew and independently, thus resulting in a huge amount of wasteful computation. As modern graph management systems have been exposed to diverse and dynamic query workload [4, 5, 6], we are motivated to reexamine the subgraph query problem augmented with valuable query workload information.

In this paper, we introduce a new subgraph querying framework, **WaSQ** (**W**orkload-**a**ware **S**ubgraph **Q**uerying), for *workload-aware* subgraph query caching and processing in large graphs. Given the query workload $W = \{w_1, \ldots, w_n\}$, where $w_i$ $(1 \leq i \leq n)$ is a previously issued subgraph query, all $w_i$'s subgraph-isomorphic embeddings in a data graph $g$ are cached in advance. When a new query $q$ is posed, the objective of WaSQ is to reuse the materialized query results of $W$ to facilitate query execution of $q$. We design in WaSQ a suite of new query refinement methods that identifies and explores the intrinsic structure relationships between workload queries and the new query $q$. Specifically, we rewrite $q$ *w.r.t.* $w_i \in W$ in a way that answers to $w_i$ can be reused in the query plan of $q$ in order to $(1)$ filter false-positive embeddings and $(2)$ expedite the overall query evaluation, thus resulting in significant performance gains for subgraph queries. To the best of our knowledge, WaSQ is the first that creates a new vision and complete solutions for workload-ware subgraph querying and optimization in large graphs. Our experimental studies in real-world large graphs validate the effectiveness of WaSQ, which consistently outperforms the state-of-the-art, workload-oblivious solutions, CFL-Match [1] and TurboISO [3], for subgraph querying.

## II. Related Work

**Subgraph queries in large graphs.** There have been a proliferation of practical subgraph querying solutions, which, by exploring a tree-structured search space (*a.k.a.* a *query plan*), iteratively match each vertex of $q$ to one of the feasible vertices in $g$. Here a *feasible matching* satisfies the *vertex-label equivalence* condition and the *connectivity-preserving* constraints enforced by subgraph isomorphism. The key to high-performance subgraph querying is to minimize the number of intermediate false-positive matchings from $q$ to $g$. Existing solutions have thus focused primarily on selecting different matching orders, and applying false-positive filtering strategies to expedite subgraph querying. In particular, TurboIso [3] and BoostIso [2] propose to merge together vertices with the same labels and neighborhood information in $q$ and $g$, respectively, to enhance filtering capabilities. CFL-Match [1] provides a subgraph query framework in which $q$ can be decomposed into $(1)$ a cyclic subgraph, called *core*, $(2)$ a forest, and $(3)$ leaves, such that the matching order follows a *core-forest-leaves* pattern. Note all existing solutions are workload-oblivious. In contrast, our method, WaSQ, exploits another performance-critical vertical, query workload, for subgraph querying.

**Query caching for graphs.** Caching has been a proven technique to enhance query performance in numerous query answering contexts. Existing RDF engines cache intermediate results of SPARQL queries to speedup forthcoming queries sharing identical triple patterns [5]. Materialized-view based approximation methods are also proposed for

2375-026X/19/$31.00 ©2019 IEEE
DOI 10.1109/ICDE.2019.00190

subgraph queries [7]. In GraphCache [8, 9], the authors design a cache system for subgraph/supergraph queries in a graph database. However, it cannot be generalized to search all embeddings of $q$ in a single, large graph $g$. Thus far, no existing research has used query workload for subgraph query caching and processing in large graphs, which is the prime goal of this work.

## III. Preliminaries

We consider in this paper the subgraph query problem defined upon *labeled*, *undirected* graphs. Given a graph $g = (V, E, l, \Sigma)$, $V$ is a set of vertices; $E \subseteq V \times V$ is a set of edges; $\Sigma$ is a set of vertex labels; $l : V \rightarrow \Sigma$ is a labeling function assigning for each vertex $v \in V$ a label $l(v) \in \Sigma$. A graph $g' = (V', E', l, \Sigma)$ is a *subgraph* of $g$, denoted as $g' \subseteq g$, if $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$. Consider a query graph $q$ and a large data graph $g$. If $q$ is subgraph isomorphic to $g$ *w.r.t.* an injective function $f$, denoted as $q \subseteq_f g$, we call $f(q) \subseteq g$ a *subgraph-isomorphic embedding* (or *embedding* for short) of $q$. There may exist multiple embeddings of $q$ in $g$, and we use $\mathcal{E}(q) = \{f_1(q), \ldots, f_n(q)\}$ to denote all the $n$ embeddings of $q$. The *subgraph query* problem is to find as output $\mathcal{E}(q)$ of $q$ from $g$, which is NP-hard [10].

In this paper, we reexamine the subgraph query problem augmented with the query workload information, $W = \{w_1, \ldots, w_n\}$, where $w_i (1 \leq i \leq n)$ is a query graph that has been resolved beforehand, and all its embeddings $\mathcal{E}(w_i) = \{f_{i_1}, \ldots, f_{i_m}\}$ have been explicitly cached. To this end, the *workload-aware* subgraph query problem can be formalized as follows,

**Definition 1 (Workload-aware Subgraph Query).** *Consider a query graph $q$, a data graph $g$, and query workload $W$. The workload-aware subgraph query problem is to find as output $\mathcal{E}(q)$ of $q$ in $g$, in the presence of $W$.* □

## IV. WASQ: Workload-aware Subgraph Querying

### A. Query Refinement and Execution

Given a new query $q$ and a workload query $w_i \in W$, we aim to refine $q$ *w.r.t.* $w_i$ in a way that the answers to $w_i$, $\mathcal{E}(w_i)$, can be effectively reused to speedup the query execution of $q$. It is immediate that there exist five structure relationships between $q$ and $w_i$: (1) $q$ is graph-isomorphic to $w_i$, $q = w_i$; (2) $q$ contains $w_i$ as a proper subgraph, $w_i \subset q$; (3) $q$ is contained in $w_i$, $q \subset w_i$; (4) $q$ and $w_i$ overlap with each other, $q \cap w_i \neq \emptyset$; and (5) $q$ and $w_i$ are totally disjoint, $q \cap w_i = \emptyset$. For each structure relationship, we will design graph refinement and execution methods for $q$ *w.r.t.* $w_i$.

**Case 1 : $\mathbf{q = w_i}$.** If $q = w_i$, $q$ has been issued and resolved in the form of $w_i$, and all its embeddings, $\mathcal{E}(w_i) = \{f_{i_1}, \ldots, f_{i_m}\}$, have been cached in $W$. To address $q$, it suffices to identify a graph-isomorphic mapping $f'$ from $q$ to $w_i$, where $f' : V_q \rightarrow V_{w_i}$. For any embedding $f_{i_l} \in \mathcal{E}(w_i)$ $(1 \leq l \leq m)$, we derive a corresponding embedding of $q$: $f_{i_l} \circ f'$, where $\circ$ denotes function composition that applies $f_{i_l}$ to the output of $f'$. Therefore, the query answer, $\mathcal{E}(q)$, of $q$ can be computed from $W$ *without* accesses to $g$,

$$\mathcal{E}(q) = \{f_{i_1} \circ f', \ldots, f_{i_m} \circ f'\} \quad (1)$$

**Case 2 : $\mathbf{q \supset w_i}$.** In this case, there exists a workload query $w_i \in W$ that is a proper subgraph of $q$, and the following theorem holds:

**Theorem 1.** Given $w_i \subset q$, for any embedding $f_k$ of $q$, $f_k \in \mathcal{E}(q)$, there exists at least one embedding $f_{i_l}$ of $w_i$, $f_{i_l} \in \mathcal{E}(w_i)$, such that $f_{i_l}(w_i) \subset f_k(q)$. □

Based on Theorem 1, any embedding of $q$ can be extended from an embedding of $w_i$. We thus refine $q$ *w.r.t.* $w_i$ as follows. We first find one subgraph-isomorphic mapping, $f' : V_{w_i} \rightarrow V_q$, *s.t.*, $w_i \subseteq_{f'} q$. Considering queries, including $w_i$ and $q$, are significantly smaller in size than $g$, such an embedding of $w_i$, $f'(w_i)$, can be identified efficiently from $q$. We then refine $q$ to a *summarized graph*, $q'$, that subsumes $f'(w_i)$ as a special *super-node* $u^*$ with all the embeddings of $\mathcal{E}(w_i)$ as $u^*$'s matching candidates. The super-node $u^*$ has now become a proxy of the workload query $w_i$ in the summarized graph $q'$. While evaluating the subgraph-isomorphic constraints relating to the super-node $u^*$, we unfold $u^*$ to crosscheck embeddings of $\mathcal{E}(w_i)$ accordingly.

The advantage of refining $q$ *w.r.t.* $w_i$ into $q'$ for subgraph querying is twofold: (1) After refinement, $q'$ is smaller than $q$ in size, resulting in a significant reduction of the number of subgraph-isomorphic checkings to be examined from $q'$ to $g$; (2) The workload query $w_i$, once condensed into a super-node $u^*$ in $q'$, is exploited as an integral pattern in subgraph querying, thus bringing better filtering capabilities than existing workload-oblivious solutions, where all the constituent vertices and edges of $f'(w_i)$ have to be matched or crosschecked individually.

**Case 3 : $\mathbf{q \subset w_i}$.** In this case, there exists a workload query $w_i \in W$ that contains $q$ as a proper subgraph. According to Theorem 1, all the embeddings of $w_i$, $\mathcal{E}(w_i)$, can be computed from those of $q$. However, the reverse is not true: not all the embedding of $q$ rest on those of $w_i$. Since $q \subset w_i$, we first find all the subgraph-isomorphic mappings from $q$ to $w_i$, $f' : V_q \rightarrow V_{w_i}$, *s.t.*, $f'(q) \subseteq w_i$. In order to find the complete set of embeddings of $q$, we consider *all* the binary partitionings $\mathcal{P}$ of the vertex set $V_q$ of $q$. Each partitioning leads to two collectively exhaustive, mutually exclusive sets of vertices, denoted as $\mathcal{P}(V_q) = V^w | V^{\bar{w}}$, where:

1) $V^w \in 2^{V_q}$ is a subset of $V_q$ comprising vertices whose subgraph-isomorphic matchings can be derived from $\mathcal{E}(w_i)$. Here $2^{V_q}$ represents the power set of $V_q$;
2) $V^{\bar{w}} = V_q \setminus V^w$ consists of vertices of $V_q$ whose matchings are not in $\mathcal{E}(w_i)$, and have to be searched against $g$.

For the first set, $V^w$, its induced subgraph in $q$ is $q[V^w]$, and all its embeddings, $\mathcal{E}(q[V^w])$, can be computed directly from $\mathcal{E}(w_i)$ by extracting the sub-embeddings (with duplicates eliminated) of $\mathcal{E}(w_i)$ confined by the vertex set, $f'(V^w)$ (Note that $V^w \subseteq V_q \subset V_{w_i}$). This sub-embedding extraction operation upon $\mathcal{E}(w_i)$ is denoted as $\pi_{f'(V^w)}\mathcal{E}(w_i)$, for which we abuse the notation of the *projection* operator, $\pi$, in relational algebra. For each vertex $u$ in the second set, $V^{\bar{w}}$, it cannot be mapped to any vertex in the embeddings of $w_i$. As a result, its candidate set $C(u)$ is refined to $C(u) \setminus \pi_{f'(u)}\mathcal{E}(w_i)$; that is, we will search in other regions of $g$, excluding $\mathcal{E}(w_i)$, for feasible matchings of $u$.

The advantage of refining $q$ *w.r.t.* $w_i$, where $q \subset w_i$, is that, for any partitioning of $V_q$ characterized by $V^w$ and $V^{\bar{w}}$, the embeddings of the induced subgraph, $q[V^w]$, can be computed directly from $\mathcal{E}(w_i)$ without referring to $g$. Although the embeddings of $q[V^{\bar{w}}]$ have to be retrieved from $g$, the candidate set $C(u)$ of each $u \in V^{\bar{w}}$ is refined to exclude the matchings of $u$ in $\mathcal{E}(w_i)$, thus effectively filtering false-positive embeddings relating to $u$.

**Case 4 : $\mathbf{q \cap w_i \neq \emptyset}$.** In this case, $q$ and $w_i$ share some common substructures. To address $q$ *w.r.t.* $w_i$, we first find a maximal connected common subgraph, $mcs$, from $q$ and $w_i$. Note that finding maximal common subgraphs for two given graphs is NP-hard [10]. However, we only need to identify one such subgraph from $q$ and $w_i$, both of which are small-size graphs. Therefore, $mcs$ can be computed efficiently in practice. We then find all the embeddings of $mcs$, $\mathcal{E}(mcs)$, *w.r.t.* $w_i$, because $mcs \subseteq w_i$. After all the embeddings of $mcs$ are computed from $g$, we further compute the embeddings of $q$ *w.r.t.* $mcs$, because $mcs \subseteq q$.

**Case 5 : $\mathbf{q \cap w_i = \emptyset}$.** It is still possible that the query graph $q$ is totally disjoint from the workload query $w_i$; that is, $q \cap w_i = \emptyset$, or the maximal connected common subgraph of $q$ and $w_i$, $mcs$, is too small in size. In such cases, the contribution of $w_i$ in answering $q$ becomes marginal. We then turn to workload-oblivious algorithms without referring to $w_i$ for help.

### B. Workload-aware Query Processing

Given a query graph $q$, a large data graph $g$, and query workload $W = \{w_1, \ldots, w_n\}$, our workload-aware subgraph querying framework, WaSQ, aims to select one workload query $w_i \in W (1 \leq i \leq n)$, if any, to help address $q$ against $g$. When a new query $q$ is received, it is first compared with each workload query $w_i \in W$. According to different structure relationships between $q$ and $w_i$ as discussed in Section IV-A, each $w_i$ is assigned to different sets, $\mathcal{B}_i (i = 2, 3, 4)$, representing different structure relationship cases. In particular, if we find a workload query $w_i \in W$, *s.t.* $w_i = q$ (Case 1), it will be immediately reused to answer $q$, rather than being assigned to a set first, say $\mathcal{B}_1$, and then retrieved for query refinement and execution.

After workload queries are allocated to different sets $\mathcal{B}_i$ $(i = 2, 3, 4)$, we then select one workload query $w^*$ following the order of examination from $\mathcal{B}_2$, to $\mathcal{B}_3$, and then to $\mathcal{B}_4$; that is, if $\mathcal{B}_i \neq \emptyset$, we select $w^*$ from $\mathcal{B}_i$ while ignoring workload queries in $\mathcal{B}_j$, where $2 \leq i < j \leq 4$. If there exist more than one workload queries in $\mathcal{B}_i$, we choose $w^*$ that is potentially most beneficial for query refinement and execution of $q$. In particular, we consider $w^* \in \mathcal{B}_i$ that is closest to $q$ in structure, thus incurring a minimum number of query refinements *w.r.t.* $q$. On the other hand, if no workload queries can be exploited (Case 5), we turn to the workload-oblivious approach for subgraph querying.

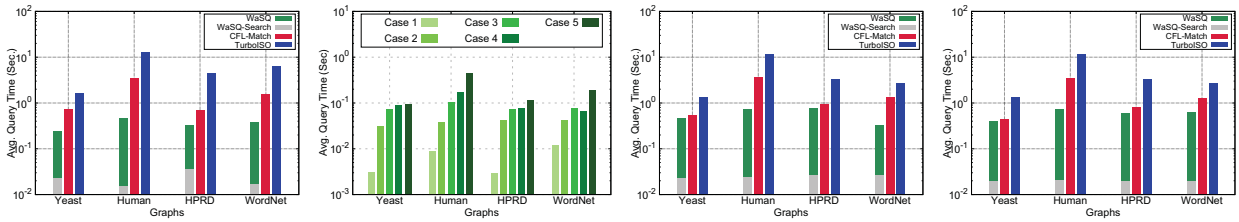
In support of workload-aware subgraph querying in a large graph $g$, we need to store past queries and their complete sets of embeddings in the cache. However, due to the space limit, we cannot maintain all subgraph queries that have been issued and resolved thus far. Instead, only up to $n$ queries are allowed in WaSQ, which constitute the query workload $W$. Furthermore, when real-world graph access patterns change over time, query workload need to be updated as well. Specifically, when the cache space becomes full, we evict some $w_i \in W$ to open space for new queries. In practice, we incorporate well-known cache-replacement schemes, such as the least frequent used (LFU) and the least recently used (LRU) policies, in WaSQ to adaptively update $W$ in the presence of dynamic and diverse query workload.

## V. Experiments

We report our experimental studies in four real-world graphs: Yeast, Human, HPRD, and WordNet, and compare WaSQ with two state-of-the-art, workload-oblivious methods, CFL-Match [1] and TurboISO [3]. Both query graphs and query workload are generated from data graphs. All algorithms were implemented in C++, and compiled by GNU g++ 4.8.4 with the -O3 flag. All experiments were carried out on a Linux server running Ubuntu 14.04 with two Intel 2.3GHz ten-core CPUs and 256GB memory.

**1. The Static Hit-Rate Setting.** In the first experiment, $W$ is saturated with 50 workload queries without replacement. We require every 10 workload queries satisfy the condition that there exists a query $q \in Q$ to account for each of the five structure relationship cases for query refinement and execution. The rationale to compose $W$ with *static hit-rates* is to illustrate in which case, $W$ will contribute most for subgraph querying. The query time results are presented in Figure 1(a). Note that WaSQ consistently outperforms CFL-Match and TurboISO in all different graphs, indicating that, once empowered with query workload, WaSQ can significantly boost subgraph query performance.

For WaSQ, we further evaluate the runtime cost for query refinement and execution (named WaSQ-Search in Figure 1(a) in gray colors). Note if there exists no such $w^* \in W$ that can be used for query refinement (namely, in




(a) Static Workload (b) Runtime for Different Cases (c) Random workload (d) Cold-start

Figure 1: **Subgraph Query performance in Different Experimental Settings.**

Case 5), this cost can be regarded as the *cache-miss penalty*. It is shown that WaSQ-Search only takes less than 10% of the overall runtime for WaSQ, indicating that finding $w^*$ from $W$ can be done efficiently, and even when a cache-miss arises, the cache-miss penalty is marginal.

We further illustrate the breakdown time of WaSQ consumed in different cases, as illustrated in Figure 1(b). Note that in Case 1 ($w_i = q$), $q$ can be answered most efficiently, as embeddings of $w_i$ can be directly employed to answer $q$ without accesses to $g$. In Case 2 ($w_i \subset q$), $q$ can be directly extended from $w_i$, so the query performance is still significantly better than the workload-oblivious method, represented by Case 5. Even in Case 3 and Case 4, our proposed query refinement and execution methods still outperforms the workload-oblivious method.

**2. The Random Workload Setting.** In the second experiment, the query workload $W$ is saturated with 50 *randomly* selected workload queries. This setting represents the real-world scenario where workload-aware subgraph querying is typically performed *without* prior knowledge of the distribution of workload queries that can be exploited in difference cases. The query time results are illustrated in Figure 1(c). Again, WaSQ consistently outperforms CFL-Match and TurboISO in all four graphs. Specifically, the performance gains of WaSQ versus TurboISO can be as large as one order-of-magnitude. In addition, the time consumed for WaSQ-Search, the upper bound of which turns out to be the cache-miss penalty, is rather marginal (within 0.1 second) in all different graphs.

**3. The Cold-start Setting.** In the third experiment, we assume the cache of WaSQ is initially empty ($|W| = 0$). When the first query $q$ is posed, we rely on the workload-oblivious method to address it. After $q$ is resolved, the query, together with all its embeddings, $\mathcal{E}(q)$, is added to $W$ as the first workload query. Subsequent queries can henceforth exploit the growing $W$ for workload-aware subgraph querying. When $W$ is full ($|W| = 50$), the workload replacement policy is enabled to keep $W$ up-to-date. Note this experimental setting resembles the cold-start and evolution process for query workload $W$ during subgraph querying in graphs. The query time results are illustrated in Figure 1(d). Again, WaSQ achieves consistent performance gains compared with CFL-Match and TurboISO in all different graphs. As a result, by exploiting the query workload information, WaSQ can always help expedite subgraph querying in real-world large graphs.

## VI. Conclusion

Subgraph querying has been a fundamental problem in graph data management. State-of-the-art solutions, however, are workload-oblivious, leaving the dynamic and diverse query workload information unexplored. In this paper, we presented a workload-aware subgraph querying framework, WaSQ, to consider the missing yet performance-critical query workload for subgraph querying. WaSQ is the first workload-aware subgraph querying framework comprising a suites of query refinement and execution methods and workload caching, organization, and maintenance techniques. Our theoretical and experimental studies have validated the necessity, applicability, and effectiveness of WaSQ, which has achieved significant and consistent performance gains for subgraph querying in real-world, large graphs.

## Acknowledgment

This work was supported by the National Science Foundation (Grant No.1743142), the Air Force Office of Scientific Research (Award No. FA95501810106), and the Army Research Office (Award No. W911NF1810395). Any opinions, findings, and conclusions in this paper are those of the author(s) and do not necessarily reflect the funding agencies.

## References

[1] F. Bi, L. Chang, X. Lin, L. Qin, and W. Zhang, "Efficient Subgraph Matching by Postponing Cartesian Products," in *SIGMOD'16*, 2016, pp. 1199–1214.

[2] X. Ren and J. Wang, "Exploiting Vertex Relationships in Speeding Up Subgraph Isomorphism over Large Graphs," *Proc. VLDB Endow.*, vol. 8, no. 5, pp. 617–628, 2015.

[3] W.-S. Han, J. Lee, and J.-H. Lee, "Turboiso: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases," in *SIGMOD'13*, 2013, pp. 337–348.

[4] G. Bagan, A. Bonifati, R. Ciucanu, G. H. L. Fletcher, A. Lemay, and N. Advokaat, "Generating Flexible Workloads for Graph Databases," *Proc. VLDB Endow.*, vol. 9, no. 13, pp. 1457–1460, 2016.

[5] N. Papailiou, D. Tsoumakos, P. Karras, and N. Koziris, "Graph-Aware, Workload-Adaptive SPARQL Query Caching," in *SIGMOD'15*, 2015, pp. 1777–1792.

[6] G. Aluç, M. T. Özsu, and K. Daudjee, "Workload Matters: Why RDF Databases Need a New Design," *Proc. VLDB Endow.*, vol. 7, no. 10, pp. 837–840, 2014.

[7] W. Fan, X. Wang, and Y. Wu, "Answering Graph Pattern Queries Using Views," in *ICDE'14*, 2014, pp. 184–195.

[8] J. Wang, N. Ntarmos, and P. Triantafillou, "GraphCache: A Caching System for Graph Queries," in *EDBT'17*, 2017, pp. 13–24.

[9] ——, "Indexing Query Graphs to Speedup Graph Query Processing," in *EDBT'16*, 2016, pp. 41–52.

[10] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.