Pivoted Subgraph Isomorphism: The Optimist, the Pessimist and the Realist

Ehab Abdelhamid
Datometry Inc.
ehab@datometry.com

Ibrahim Abdelaziz
IBM Research
ibrahim.abdelaziz1@ibm.com

Zuhair Khayyat
Lucidya LLC
zuhair@lucidya.com

Panos Kalnis
KAUST
panos.kalnis@kaust.edu.sa

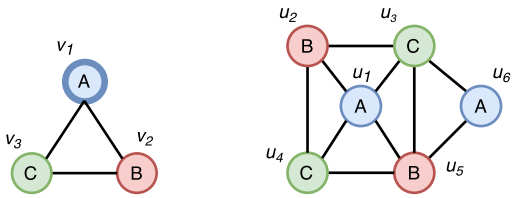
ABSTRACT

Given query graph Q with pivot node v, Pivoted Subgraph Isomorphism (*PSI*) finds all distinct nodes in an input graph G that correspond to v in all matches of Q in G. PSI is a core operation in many applications such as frequent subgraph mining, protein functionality prediction and in-network node similarity. Existing applications implement PSI as an instance of the general subgraph isomorphism algorithm, which is expensive and under-optimized. As a result, these applications perform poorly and do not scale to large graphs. In this paper, we propose SmartPSI; a system to efficiently evaluate PSI queries. We develop two algorithms, called optimistic and pessimistic, each tailored for different instances of the problem. Based on machine learning, SmartPSI builds on-the-fly a classifier to decide which of the two algorithms is appropriate for evaluating each graph node. SmartPSI also implements a machine learning-based optimizer to generate low-cost execution plans. Our experimental evaluation with large-scale real graphs shows that SmartPSI outperforms existing approaches by up to two orders of magnitude and is able to process significantly larger graphs. Moreover, SmartPSI is shown to achieve up to 6 times performance improvement when it replaces standard subgraph isomorphism in the state-of-the-art distributed frequent subgraph mining system.

1 INTRODUCTION

Graphs are widely used to model information in a variety of real world applications such as social networks [31, 32], chemical compounds [20] and protein-protein interactions [25]. Finding all valid bindings of a particular query node is an important step in various application domains such as frequent subgraph mining [4, 37], protein functionality prediction [12], neighborhood pattern mining [15], query recommendation [16] and in-network node similarity [39]. This step is called Pivoted Subgraph Isomorphism (*PSI*); for query Q with pivot node v, PSI finds the set of distinct matches of v in an input graph G. For example, PSI query S ($\underline{v_1}, v_2, v_3$) in Figure 1 has two matches of the pivot node v_1 in G: u_1 and u_6.

Existing applications [4, 13, 15, 16, 25] depend on subgraph isomorphism [9, 17, 30] to evaluate PSI-like queries. They use subgraph isomorphism to find all matches of the query and then project distinct nodes that correspond to the pivot node. In the example of Figure 1, subgraph isomorphism generates 5 intermediate results: ($\underline{u_1}, u_2, u_3$), ($\underline{u_1}, u_2, u_4$), ($\underline{u_1}, u_5, u_4$), ($\underline{u_1}, u_5, u_3$) and ($\underline{u_6}, u_5, u_3$)), in order to project u_1 and u_6 as results for the PSI query. We show in Table 1 the number of PSI results compared to the number of intermediate isomorphic subgraphs for various real graphs and query sizes (see Section 5 for details). Observe that the number of isomorphic subgraphs grows exponentially with the query size even for small and sparse datasets like Yeast, whereas the actual PSI results are significantly fewer. Consequently, the performance of applications that depend on subgraph isomorphism degrades rapidly with complex queries and larger graphs.



(a) PSI query S (b) Input graph G

Figure 1: (a) A PSI query with v_1 as pivot, (b) input graph. The bindings of v_1 are u_1 and u_6

A possible approach to reduce the exponential number of matches in subgraph ismorphism-based solutions is to stop the search once a match of the pivot node is found for each candidate graph node. We demonstrate the effect of this optimization by proposing TurboIso$^+$ (Section 5.2) as a modified version of TurboIso [17] which is a highly optimized subgraph isomorphism solution. We show in Table 2 that TurboIso$^+$ is significantly faster than TurboIso for all query sizes. However, it remains computationally expensive since it cannot avoid the overhead of the sophisticated data structures used to compile all occurrences of the given query. As such, using subgraph isomorphism to solve PSI queries is unnecessarily expensive. Moreover, to the best of our knowledge, there is no existing research that has been devoted to efficiently answer PSI queries.

In this paper, we formalize the pivoted subgraph isomorphism problem and introduce the *optimistic* and *pessimistic* approaches for evaluating PSI queries efficiently. The optimistic approach is highly optimized to confirm that a graph node matches the pivot node in the query. It uses a greedy depth-first guided search to minimize the number of graph explorations required to prove that a particular node is indeed a matching node. If no match exists, it would have unnecessarily wasted resources to discover non-matching nodes. The pessimistic approach, on the other hand, is highly optimized to verify that a graph node does not match the given query. It is based on a random, non-guided, search that prunes adjacent nodes early depending on their neighbourhood.

Table 1: Number of subgraph query matches using PSI vs. standard subgraph isomorphism

Dataset	Query	Query Size						
		4	5	6	7	8	9	10
Yeast	PSI	7×10^4	6×10^4	5×10^4	5×10^4	4×10^4	3×10^4	3×10^4
	Subgraph Iso.	1.3×10^7	1.2×10^8	1.1×10^9	7.4×10^9	5.8×10^{10}	6.0×10^{11}	2.8×10^{12}
Cora	PSI	2.1×10^5	1.9×10^5	1.4×10^5	1.2×10^5	1.1×10^5	0.9×10^5	0.8×10^5
	Subgraph Iso.	1.6×10^8	1.2×10^{10}	1.1×10^{12}	8.4×10^{13}	1.5×10^{15}	8.9×10^{16}	NA
Human	PSI	1.4×10^5	1.3×10^5	1.2×10^5	1.1×10^5	1.1×10^5	1.0×10^5	1.1×10^5
	Subgraph Iso.	2.6×10^{10}	7.6×10^{12}	NA	NA	NA	NA	NA

Early pruning helps the pessimistic algorithm to avoid unnecessary graph traversals. It detects non-matching nodes significantly faster than the optimistic approach, however, it has a higher cost when used for matching nodes.

To maximize the gain from these two methods, the optimistic method has to be used to validate nodes that match the query while the pessimistic method should be used to confirm other non-matching nodes. Unfortunately, we do not know which graph node matches the given PSI query prior to query evaluation, while it is counter-intuitive to use the proposed methods randomly on the input graph. To solve this problem, we propose SmartPSI, a solution that relies on machine learning to predict which method is best to evaluate each graph node. In SmartPSI, a classification model is trained to predict the type of each graph node (i.e., whether a node does or does not match the given query). Based on the prediction result, the appropriate method is picked accordingly; the optimistic method is used for matching graph nodes (predicted valid nodes) and the pessimistic is used for non-matching nodes (predicted invalid nodes). Notice that SmartPSI is an exact solution since the correctness of the result is guaranteed even when using the opposite evaluation, this is due to the fact that both algorithms traverse the whole search space in the worst-case scenario. SmartPSI is also coupled with a novel query optimizer, that recommends an efficient search order plan for evaluating each graph node. The last row of table 2 gives a hint on how the combined ideas implemented in SmartPSI, which are solely designed for PSI queries, perform compared to the more general solutions of subgraph isomorphism.

In summary, our contributions are:

- We introduce two methods for evaluating PSI queries, each method is optimized for a particular type of the input graph nodes.
- We propose a solution based on machine learning that, for each graph node, predicts the node type, picks the corresponding optimized evaluation method, and selects an efficient plan for query evaluation.
- We also propose a preemptive query processor that employs a detection and recovery technique at run-time to reduce the impact of incorrect predictions.
- We experimentally compare the performance of SmartPSI against the state-of-the-art subgraph isomorphism techniques. We show that SmartPSI is orders of magnitude faster and it scales to much larger queries and input graphs. Furthermore, we empirically show that our optimizations significantly improve the performance of a cutting edge frequent subgraph mining system.

The rest of the paper is organized as follows: Section 2 formalizes the problem and presents various PSI applications. Section 3 introduces two novel PSI evaluation methods. Section 4 presents a two-threaded baseline solution and describes machine learning based optimizations that overcome the limitations of the baseline. Section 5 presents the experimental evaluation. Section 6 surveys the related work while Section 7 concludes.

Table 2: Performance of PSI solutions on Human dataset (details in Section 5). SmartPSI is our proposed approach.

Query size	4	5	6	7
TurboIso	5.4 hrs	>24 hrs	>24 hrs	>24 hrs
TurboIso$^+$	14 min	30 min	45 min	2.4 hrs
SmartPSI	27 sec	47 sec	2 min	4.3 min

2 PIVOTED SUBGRAPH ISOMORPHISM

2.1 Definition

A labeled graph $G = (V_G, E_G, L_G)$ consists of a set of nodes V_G, a set of edges E_G and a labeling function L_G that assigns labels to nodes and edges. A graph $S = (V_S, E_S, L_S)$ is a *subgraph* of a graph G iff $V_S \subseteq V_G$, $E_S \subseteq E_G$ and $L_S(v) = L_G(v)$ for all $v \in V_S \cup E_S$. Subgraph isomorphism is the task of finding all occurrences of S inside G. It is a computationally expensive problem, known to be NP-Complete [14]; it needs to validate a huge number of intermediate and final results. PSI, which is also NP-Complete [15], relaxes subgraph isomorphism by focusing on finding all node matches of a particular query node (v_p), which is called a *pivot* node, inside the input graph G.

Definition 2.1. A pivoted graph is a tuple $\eth$={G, v_p}, where G is a labeled graph and $v_p \in V_G$ is called the pivot node of the pivoted graph $\eth$.

The pivot node (v_p) is the node of interest, and it is usually set by the user who creates the query.

Definition 2.2. A *pivoted subgraph isomorphism* of $\eth_S$=(S, v_p) to $\eth_G$={G, u_p} where $S = (V_S, E_S, L_S)$, $G = (V_G, E_G, L_G)$, $v_p \in V_S$ and $u_p \in V_G$, is an injective function $M : V_S \rightarrow V_G$ satisfying (*i*) $L_S(v) = L_G(M(v))$ for all nodes $v \in V_S$, (*ii*) $(M(u), M(v)) \in E_G$ and $L_S(u, v) = L_G(M(u), M(v))$ for all edges $(u, v) \in E_S$, and (*iii*) $M(v_p) = u_p$.

Rather than finding all matches in subgraph isomorphism, PSI finds at most one occurrence per graph node u_p, which significantly reduces computation and memory overheads. The input graph node u_p is called a *valid* node for query $\eth_S$ if it matches v_p in at least one occurrence of S in G, otherwise it is called an *invalid* node. PSI has an important role in several applications [4, 13, 15, 16, 25]. In the following discussion, we highlight some of these applications.

2.2 PSI Applications

Frequent Subgraph Mining (FSM). FSM is an important operation for graph analytics and knowledge discovery [4, 5, 13]. Given an input graph and a support threshold, FSM evaluates the frequencies of a large number of candidate subgraphs to determine which one is frequent. Typically, this evaluation relies on finding the distinct input graph nodes that match their corresponding candidate subgraph nodes [11]. Current solutions [4, 13] employ subgraph isomorphism to conduct this step. Instead of using the expensive subgraph isomorphism algorithms, PSI can be employed to efficiently evaluate the frequencies. Using PSI instead of subgraph isomorphism results in a significant improvement in the overall efficiency of FSM techniques (see Section 5.5).

Function prediction in PPI networks. In the domain of protein-protein-interaction (PPI) networks, it is common to discover new proteins with unknown functions [12]. Knowledge about their functionality is essential to analyze these networks and discover valuable insights. In order to predict these functions, a set of significant patterns are extracted from the PPI network. Each protein with unknown function is then matched against the extracted patterns. A label that is matched to the node with unknown function represents a predicted function for this node. Each significant pattern is represented by a different pivoted subgraph query, and the matching task resembles a PSI query evaluation.

Discovering Pattern Queries by Sample Answers. Knowledge bases are effective in answering questions. Nevertheless, it is challenging for a user to issue a query that follows the schema of the knowledge base. Han et al. [16] proposed a query discovery approach to assist users by recommending candidate queries. This approach is based on having a sample answer set (i.e., a set of nodes which the user thinks they match his query). The first step finds a set of queries that match the neighborhood around the given nodes. These queries represent recommended candidate queries. This step is conducted by a series of PSI operations which tries to filter out all queries that do not match any of the given answer nodes. Queries that pass the first step are then ranked and the top ones are recommended for the user.

Neighborhood Pattern Mining. Mining frequent neighborhood patterns [15] finds the set of frequent patterns that each originates from graph nodes with a particular label. Similar to FSM, candidate patterns are generated and evaluated to determine whether they are frequent. Though, the evaluation step is different. Given a specific label, each candidate pattern is evaluated by PSI to know the number of graph nodes that satisfy this pattern. Based on this process, interesting knowledge can be obtained regarding the common connectivity patterns found around each node label.

In-network Node Similarity. In many applications, it is very important to measure the similarity among graph nodes. These applications include role discovery [28], objects similarity [21] and node clustering [19]. Two nodes are similar if they have similar neighborhoods. There are several techniques for calculating nodes similarity. A recent approach [39] proposed a unified framework for measuring the similarity between two nodes. One of the proposed metrics is the maximum common pivoted subgraph that exists around the two nodes. This metric is extended into a more general similarity metric, which is used to compare the common pivoted subgraphs occurring in the neighborhoods of any two nodes. Such approach is a direct application of PSI.

3 THE OPTIMIST AND THE PESSIMIST

In this section, we propose an optimist and a pessimist evaluation mechanisms to improve PSI evaluation. The *optimistic* method is optimized to confirm that a candidate node is valid, i.e., matches the query pivot node. On the other hand, the *pessimistic* method is effective in proving that a candidate node is invalid. Since both methods require access to a neighborhood signature, we will first describe what a neighborhood signature is and how it is calculated (Sections 3.1 and 3.2). Then, we introduce each PSI method in Sections 3.3 and 3.4.

3.1 Neighborhood Signature

The *neighborhood signature* of a graph node represents how this node interacts with its neighbors. A good neighborhood signature should be concise and effective in pruning/matching of a graph node. The literature introduces several signature definitions [7, 23, 42, 43]. We propose to use a label propagation technique inspired by the work of proximity pattern mining [23]. This technique assigns a list of weights to each graph node. Each weight corresponds to a label and reflects the proximity of that label to the graph node.

Definition 3.1. Let u be a node in the graph G. The neighborhood signature of u within a distance D is the set of pairs:

$$NS_u{}^D = \{(l_1, w_1^D), (l_2, w_2^D) \ldots (l_n, w_n^D)\}$$

where each l_i is a node label and each weight $w_i^D \in R$.

Label weights are propagated from nearby nodes such that for each label l_i, its weight w_i^D is calculated as:

$$w_i^D = \sum_{d=0}^{D} 2^{-d} \times C_u(l_i, d)$$

where D is the maximum propagation depth, and $C_u(l_i, d)$ is the number of nodes with label l_i having a shortest distance d from node u. Notice that the signature $NS_u{}^D$ is small since its size depends on the number of distinct labels that appear in the graph. It is also effective as the used weights reflect how neighbors are structured around the node. In this work, we use the same maximum propagation depth for query graphs and the input graph. Thus, we will omit the use of the superscript D where it is not needed.

Example: Consider the data graph G in Figure 1, G contains three different labels; $\{A, B, C\}$. Assuming that the maximum propagation depth is set to 2, the signature of node u_1 is calculated as follows: (*i*) first, the label of u_1, which is 'A', is given a weight = 1. Consequently, the list of weights from distance 0 is: $\{(A, 1), (B, 0), (C, 0)\}$. (*ii*) For the first level of neighbors, there exist four nodes, two nodes are labeled 'B' and the other two are labeled 'C'. Therefore, the weight of 'B' in this case is 2×0.5 and the weight of 'C' = 2×0.5. Consequently, the list of propagated weights from this level is $\{(A, 0), (B, 1), (C, 1)\}$. (*iii*) There is only one node that is two hops away from u_1; i.e. u_6, which has label 'A', and thus the weight of 'A' = $1 \times 0.25 = 0.25$. Consequently, the propagated weights from this level = $\{(A, 0.25), (B, 0), (C, 0)\}$. (*iv*) Finally, $NS_{u_1}^2$ is the sum of weights from all levels, hence, $NS_{u_1}^2 = \{(A, 1.25), (B, 1), (C, 1)\}$.

Signature Computation. Traditional approaches [23] follow an exploration-based strategy to compute node signatures. It executes a breadth-first search algorithm iteratively till the maximum propagation depth is met. This approach is very simple

to implement but has an exponential computational complexity ($O(|N|.|L|.d^D)$), where |**N**| is the number of nodes in the graph, |**L**| is the number of distinct node labels, **d** is the average node degree and **D** is the maximum traversal depth.

Optimization. To improve the performance of computing neighbourhood signatures, we propose a significantly faster approach based on matrix multiplication to compute all neighborhood signatures of the input graph. In comparison to the exploration-based technique [23], the computational complexity of our proposal is $O(|N|.|L|.d.D)$. The first step is to create a matrix $NS^0 : |N| \times |L|$. Each row $NS^0(n)$ represents the neighborhood signature of a node n and is initialized as: $NS^0(n) = [b_1, b_2, ...b_L]$ where $b_l = 1$ if $l \in L(n)$ and 0 otherwise. Then for the subsequent D iterations, the corresponding row for each node n is updated as follows:

$$NS^i(n) = NS^{i-1}(n) + \frac{1}{2} Adj(n) \times NS^{i-1}$$

where $Adj(n)$ is the adjacency matrix row that corresponds to node n. Notice that the label weights obtained by the iterative matrix multiplications may differ from the ones generated by the exploration based approach. This is because the iterative matrix approach considers neighbors labels multiple times through different paths, compared to one time through the shortest path in the previous approach. However, these weights are still based on the proximity of the labels that exist around the node under consideration.

Example: Consider the PSI query in Figure 2. Assuming a maximum propagation depth of 2, we first initialize NS^0 and Adj to be:

$$NS^0 = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \end{array}$$

$$Adj = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{ccccc} v_0 & v_1 & v_2 & v_3 & v_4 \end{array} \\ \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 \\ 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 1 \\ 0 & 0 & 0 & 1 & 0 \end{pmatrix} \end{array}$$

Then, we apply two iterations of the matrix multiplication to compute the results for NS^1 and NS^2:

$$NS^1 = \begin{array}{c} \\ v_0 \\ v_1 \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 1 & 1/2 & 0 & 0 \\ 1/2 & 3/2 & 1/2 & 0 \\ 0 & 3/2 & 1/2 & 0 \\ 0 & 1 & 1 & 1/2 \\ 0 & 0 & 1/2 & 1 \end{pmatrix} \end{array}$$

$$NS^2 = \begin{array}{c} \\ v_0 \\ \mathbf{v_1} \\ v_2 \\ v_3 \\ v_4 \end{array} \begin{array}{c} \begin{array}{cccc} A & B & C & D \end{array} \\ \begin{pmatrix} 5/4 & 5/4 & 1/4 & 0 \\ \mathbf{1} & \mathbf{3} & \mathbf{5/4} & \mathbf{1/4} \\ 1/4 & 11/4 & 5/4 & 1/4 \\ 1/4 & 13/4 & 2 & 1 \\ 0 & 1/2 & 1 & 5/4 \end{pmatrix} \end{array}$$

Notice that row v_1 in NS^2 is the neighborhood signature of the query graph pivot node in Figure 2(a).

PSI query evaluation benefits from neighborhood signatures in two ways. First, graph traversals are guided towards the relevant graph nodes in order to efficiently reach a match for a given query. Second, significant pruning is achieved during the evaluation of graph nodes that do not satisfy the query.

3.2 Neighborhood Signature Satisfaction

The neighborhood signatures of two nodes can be utilized to quickly judge if they have the same neighbourhood and hence could be a match. A signature NS_i is said to satisfy another signature NS_j if for every $(l_j, w_j) \in NS_j$, there exists $(l_i, w_i) \in NS_i$ where $l_i = l_j$ and $w_i \geq w_j$. In Figure 1(b), u_1 has a neighborhood signature $NS^2_{u_1} = \{(A, 1.25), (B, 1), (C, 1)\}$. For query S in Figure 1(a), $NS_{v_1} = \{(A, 1), (B, 0.5), (C, 0.5)\}$. NS_{u_1} satisfies NS_{v_1} since the weights of all labels in NS_{u_1} are larger than their corresponding ones in NS_{v_1}.

Proposition 3.2. *Given a graph G, a graph node u and a pivoted query $\eth$=(S, v_p), where S is a subgraph and v_p is the pivot node, if NS_u does not satisfy NS_v, then u is an invalid node given the pivoted query $\eth$.*

Proof: We will use a proof by contradiction approach. Let NS_u do not satisfy NS_v. Let's assume a graph node u be valid for the pivot node v of subgraph S. Since NS_u do not satisfy NS_v. This implies that there exists at least one pair $(l_i, w_u) \in NS_u$ and a pair $(l_j, w_v) \in NS_v$, where $l_i = l_j$ and $w_u < w_v$. For w_u to be less than w_v, the contribution to the weight w_u from neighbor nodes with label l_i should be less than of neighbors of node v with the same label. The weight contribution relies on the number of neighbors labeled l_i and their distances. Accordingly, either the number of neighbor nodes of u with label l_i is less than that for neighbors of node v with the same label, or their distances are larger than their corresponding ones in the query. The last statement contradicts with the assumption that v is a valid node of u since this requires that each neighbor of v to have a distinct corresponding match with a neighbor of u. □

3.3 The Optimist

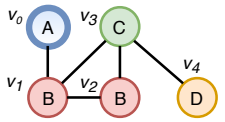
The optimistic algorithm uses a greedy depth-first guided search to quickly find a match for the given query. Its name is derived from the fact that it assumes that input candidate nodes are valid and optimizes the evaluation accordingly. The algorithm starts by generating a satisfiability score for each neighbor of the candidate node and then it sorts them based on the calculated scores descendingly. The satisfiability score measures the likelihood of a graph node being a match for its corresponding query node, it is calculated as:

$$SS(u, v) = avg_{(l, w_l) \in NS_v}(NS_u(l)/w_l)$$

For example, the satisfiability score of u_1 in G and v_1 in S from Figure 1, where $NS_{u_1} = \{(A, 1.25), (B, 1), (C, 1)\}$ and $NS_{v_1} = \{(A, 1), (B, 0.5), (C, 0.5)\}$, is calculated as follows:

$$\frac{(1.25/1) + (1/0.5) + (1/0.5)}{3} = 1.75$$

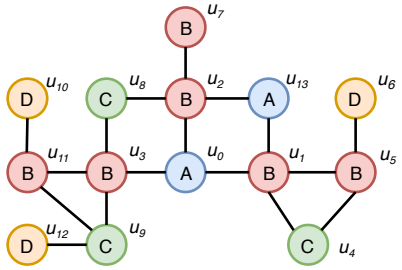
We show more examples of satisfiability scores in Figure 2. Moreover, Figure 2 shows how to use the computed neighborhood signatures of v_1, u_1, u_2 and u_3 (based on Section 3.1) to determine the satisfiability scores of node pairs (u_1, v_1), (u_2, v_1) and (u_3,



Neighborhood Signature

	A	B	C	D
NS^2_{v1}	1	3	5/4	1/4

(a) PSI query with v_0 as pivot



Neighborhood Signatures

	A	B	C	D
NS^2_{u1}	2	4	5/4	1/4
NS^2_{u2}	2	4	1	0
NS^2_{u3}	1	4	9/4	1/2

$SS(u_1,v_1) = \frac{(2/1) + (4/3) + ((5/4)/(5/4)) + (1/4)/(1/4)}{4} = 1.33$

$SS(u_2,v_1) = \frac{(2/1) + (4/3) + (1/(5/4)) + (0/(1/4))}{4} = 1.05$

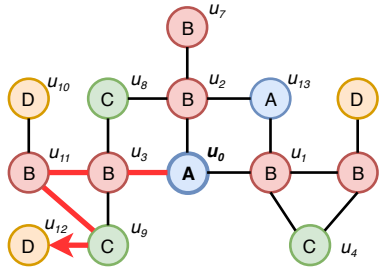
$SS(u_3,v_1) = \frac{(1/1) + (4/3) + ((9/4)/(5/4)) + ((1/2)/(1/4))}{4} = 1.54$

(b) Input graph G

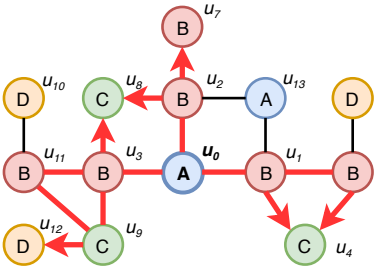
Figure 2: (a) PSI query S, showing the neighbors signature for v_1 (b) Input graph G with satisfiability scores with regard to query node v_1

v_1). Note that a higher neighbourhood signature of label l for v_i means that v_i is connected with many nodes having label l which increases the chance for finding a match for u. As a result, the larger the satisfiability score of a graph node, the higher the chances that it satisfies a corresponding node in the PSI query. Once the satisfiability scores are calculated for all neighbors, the optimistic algorithm sorts them according to their scores and traverses those with higher scores first. The idea is to prioritize nodes with high likelihood to reach a result over other nodes with less chances, and thus the algorithm can finish faster. This step is repeated recursively by calculating the satisfiability scores of the new neighboring nodes and change paths until a match is found or all possible graph traversals are exhausted. The optimistic algorithm is useful only when used with valid nodes; i.e. nodes that match the pivot node. Otherwise, it suffers from performance degradation caused by the overhead of calculating the satisfiability scores for candidate nodes and sorting them, which is an unnecessary overhead for the case of invalid nodes.

Figure 3(a) shows an example of how the optimistic algorithm employs the satisfiability scores obtained in Figure 2(b) to reach a solution quickly. Consider evaluating the neighbors of u_0 for matching with query node v_1, the optimistic algorithm is going to traverse the graph starting from u_3, as shown in Figure 3(a), since it has the highest score ($SS(u_3) = 1.54$). As a result, the optimistic algorithm is able to find that u_0 is valid using the least number of traversals by avoiding the traversal of u_1 and u_2. A random



(a) Traversals with sorting



(b) Traversals without sorting

Figure 3: The behaviour of the optimistic algorithm starting from u_0 with (a) sorting and (b) without sorting

Algorithm 1: PSI Evaluation

Input: G the input graph, S query subgraph, M Mapping, T Method type; Optimistic or Pessimistic
Output: R true if a result is found, false otherwise

```
1  if M is full mapping then Return true
2  v_next ←GetNextQueryNode(S)
3  C ←GetNextCandidateNodes(G, S, v_next)
4  if Super Optimistic then C ←GetLimitedCandidates(C, Max)
5  if T is Optimistic then C ←SortBySatisfiabilityScore(C)
6  foreach c ∈ C do
7      if T is Pessimistic And ¬Satisfy(c, v_next) then Continue
8      M_new ← M + (v_next, c)
9      if ValidMapping(M) then
10         valid ←PSI(G, S, M_new, T)
11         if valid is true then Return true
12 Return false
```

traversal would have exhausted up to 3 different traversals from u_0, as shown in Figure 3(b), since u_1 and u_2 do not lead to an occurrence.

Algorithm 1 shows the steps required for evaluating PSI queries. To enable the optimistic algorithm, the input flag T is set to optimistic. Note that this algorithm is recursive where it stops whenever a result is found (Line 1). The crux of the optimistic method is the sorting of graph nodes based on their satisfiability scores (Line 5). The sorting function introduces extra overhead, which might be unacceptable when evaluating nodes with high degrees. To avoid such problem and benefit from situations where a match can be quickly found, we propose a super optimistic step,

where only a small portion of the neighbor nodes are checked. The difference between the "super" optimistic and the original optimistic method is in Line 4. For the super optimistic version, the number of candidates is limited by a maximum value (we use 10 in our experiments). Therefore, the overhead of sorting is significantly minimized and less intermediate graph nodes are visited. For each graph node, the optimistic approach proceeds with the super optimistic version. If a result is found, it returns that result without further processing. Otherwise, the normal optimistic method is used.

3.4 The Pessimist

In this method, we exploit aggressive pruning to quickly reach a decision. The pessimistic algorithm evaluates the label, degree and the neighborhood signature of the candidate nodes against the corresponding query node. Then, it prunes non-matching candidate nodes without carrying out further graph traversals. In comparison to a typical subgraph isomorphism evaluation, this early pruning reduces the number of intermediate results since it does not go deeper in the search plan for pruned nodes. This pruning results in a significant improvement, especially for graphs with high degree nodes. Although pruning has a notable computation price due to its excessive calculations per evaluated node, it is worth exploiting it for invalid nodes to reduce the cost of unproductive traversals. On the contrary, using this pruning for valid nodes introduces unnecessary overhead to reach the conclusion that a node is valid.

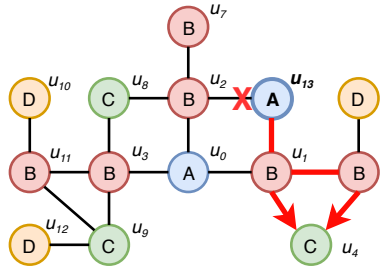
The utilization of the neighborhood signature in pruning is important since it captures wider properties of the area surrounding each graph node. Proposition 3.2 allows pruning of a node when its neighborhood signature cannot satisfy the neighborhood signature of the query node. Based on this proposition, a significant number of graph nodes are pruned but with a small number of false positive nodes (i.e., nodes that satisfy the corresponding neighborhood signature but are non-candidates because they do not match the query graph). Algorithm 1 follows the pessimistic approach by setting the flag T to Pessimistic. It is similar to the optimistic method except Line 7 which applies pruning on nodes using Proposition 3.2 and the neighborhood signature.

We show in Figure 4 the difference between the pessimistic and optimistic algorithms when evaluating an invalid node, i.e., u_{13}, against v_0 in Figure 2(a). In this example, the candidate mappings of query node v_1 are u_1 and u_2. Notice that both nodes satisfy label and direct neighbors requirements; both have label "B" and are connected to nodes labeled "B" and "C". When looking at the neighborhood signatures of both nodes, we discover that u_1 satisfies the neighborhood signature of v_1, i.e., its features have weights higher than or equal to those of the features of v_1. On the other hand, the neighborhood signature of u_2 does not satisfy that of v_1 because weights corresponding to labels "C" and "D" are both less than those for v_1. As a result, the pessimistic algorithm prunes u_2 early in the evaluation and traverse u_1 for further evaluation as shown in Figure 4(a). The optimistic algorithm, if employed, would explore both paths, u_1 and u_2, as shown in Figure 4(b).

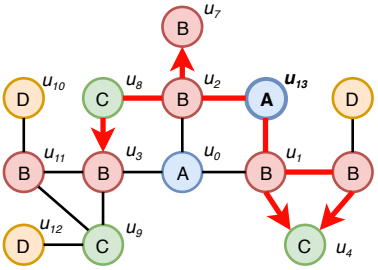
4 THE REALIST: SMARTPSI

4.1 Two-threaded PSI Baseline

In order to benefit from the optimistic and pessimistic methods, we need to know the type of each graph node; i.e., valid



(a) Pessimistic



(b) Optimistic

Figure 4: The behaviour of (a) the pessimistic vs. (b) the optimistic algorithm starting from u_{13}

or invalid, and pick the appropriate method accordingly. However, such knowledge is not available beforehand. Therefore, we propose a simple baseline, based on threading, that leverages the two algorithms for PSI evaluation. Figure 5 shows how this baseline works. For each graph node, two threads are executed concurrently. One thread runs the optimistic method while the other runs the pessimistic method. The thread that finishes its evaluation first stops the other thread and returns the result. This approach guarantees that each node is evaluated in almost the least wall-clock time of both methods (optimistic and pessimistic). However, it suffers from two issues: (*i*) under-utilization, as two threads are doing the job of a single task, and (*ii*) initiating and stopping millions of threads to evaluate the candidate graph nodes leads to a huge unnecessary cumulative overhead.

4.2 SmartPSI

This section discusses SmartPSI; our proposed PSI solution that employs machine learning techniques to avoid the limitations of the above-mentioned two-threaded baseline. We show in Figure 6 the architecture of SmartPSI which integrates the utilization of both PSI methods with two independent classification models. The first model (Model α) is a node type classifier that identifies whether a graph node is valid or invalid. Compared to the two-threaded approach, this model allows SmartPSI to benefit from the optimized PSI methods, i.e., the optimistic and the pessimistic, without sacrificing extra resources. The second classification model (Model β) is trained to predict an efficient query execution plan for each candidate node.

SmartPSI starts by loading the entire input graph in-memory, then it computes the neighborhood signatures for each graph node. When receiving a query, it extracts the candidate nodes from the input graph. A small percentage of the candidate nodes (around 10% up to a maximum value) is randomly selected to train

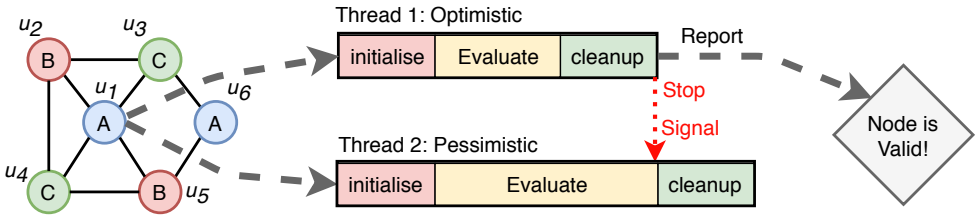


Figure 5: The two-threaded PSI solution

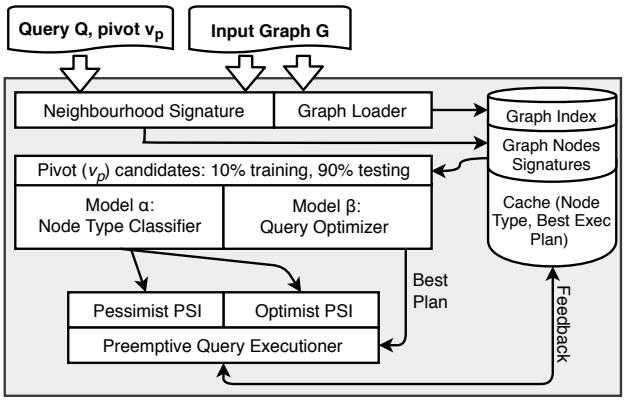


Figure 6: SmartPSI System Architecture

both models (α and β) using the Random Forest classification algorithm [10]. Our empirical evaluation shows that using this simple classifier is effective since it provides both a lightweight training time as well as a decent prediction accuracy (see Section 5.4). However, utilizing other machine learning classifiers; e.g. Neural Networks or Support Vector Machines, is orthogonal to our work. The remaining candidate nodes are passed to the trained models to predict their types and evaluation plans based on their neighborhood signatures. SmartPSI uses these predictions to evaluate each node against the query and the result of this evaluation is cached for future use. In the rest of this section, we describe in more details each one of these steps.

4.2.1 Predicting the Node Type (Model α). Correctly predicting the candidate node type (as *valid* or *invalid*) helps SmartPSI to decide the node's most efficient graph matching method (optimistic or pessimistic). Valid nodes are best evaluated using the optimistic algorithm while the pessimistic algorithm is much better at evaluating invalid nodes. To select the best method for each graph node, we utilize a binary classification model to predict the node's type (class) based on its neighborhood signature, and accordingly decide which method to use for evaluation.

Training: The neighborhood signature of each graph node is used to build the feature vectors for our classifier. Each label in the neighborhood signature represents a feature and the value associated with this label in the signature is considered as the feature's weight. Each training node is evaluated using the pessimistic method and is labeled based on its confirmed type. The pessimistic method is used during model training since, on average, it is more stable and performs better than the optimistic method.

Prediction: For the remaining graph nodes, the trained model is used to predict the node type, either valid or invalid. Nodes predicted as valid are passed to the optimistic method while invalid nodes are given to the pessimistic method.

4.2.2 Predicting an Optimal Plan (Model β). Selecting the order in which query nodes are evaluated is important for efficient evaluation. A bad order results in an excessive number of intermediate results and consequently poor performance. Existing techniques [17, 18, 30] employ heuristic approaches to prioritize the evaluation of query nodes that have higher selectivity. However, these approaches may not consistently provide good performance since their proposed plans are not adapted to the local features of each graph node. We describe next how SmartPSI trains a classifier to predict an efficient locality-aware query plan for each graph node based on its neighborhood signature.

Training: Model β is a *multi-class* classifier; its training phase uses the same set of graph nodes used for model α. For each query, a set of plans are generated and evaluated for each training node u_t. The plan that results in the least time is selected as the optimal plan for u_t. This plan is considered the class of u_t and is fed along with u_t feature vector to the multi-class classifier.

The training time can be very large especially for big queries with a considerably large number of possible plans. For example, a subgraph with 6 nodes can have up to 6! (or 720) plans. It is not practical and very expensive to evaluate all these plans. To mitigate this problem, we only train using a small sample of these plans to minimize the training time. Furthermore, we avoid evaluating very expensive plans by enforcing a configurable time limit when evaluating each one of the sample plans. We first set the time limit to a relatively small value. Evaluations of subsequent plans are not allowed to exceed that time limit. If no plan is able to finish within the allotted time, the time limit is gradually increased. This process repeats until at least one plan can finish within the time limit.

Prediction: For each remaining candidate node, SmartPSI uses the created model β to predict a good query execution plan based on its neighborhood signature. Then, this plan is fed to the corresponding PSI method for evaluation.

4.2.3 Prediction Caching. As shown in Figure 6, correct predictions of both models are cached to improve the run-time performance of SmartPSI. The cache module stores the node signature of already evaluated nodes. Upon checking next candidate nodes, SmartPSI checks if a similar node has been evaluated before. If exists, the PSI evaluation model and execution plan decisions are looked up from the cache without consulting the

classifiers. Using caching improves the efficiency of SmartPSI since the time needed to consult the prediction model and possible wrong predictions are avoided. This is because nodes having the same neighborhood signature are deemed similar since they have similar graph structures around them. Thus, they are expected to have the same optimal method and execution plan.

4.3 Preemptive Query Execution

As in any classifier, the trained model may result in incorrect predictions. In our case, there are two types of prediction errors; incorrect plan prediction and incorrect node type prediction. SmartPSI, however, employs a *detection and recovery* technique to minimize the impact of wrong predictions. As shown in Figure 6, the preemptive query processor monitors the evaluation of a node u_i and an incorrect prediction is detected if the time of this evaluation exceeds a maximum value. We calculate this value as follows:

$$MaxTime(u_i) = 2 \times AvgT(PSIMethod(u_i), Plan(u_i))$$

$PSIMethod(u_i)$ returns the used PSI method for u_i while $Plan(u_i)$ is the currently used plan. Note that $AvgT(X, Y)$ returns the average time needed for the selected PSI method X and plan Y during the training phase. To be able to recover from bad predictions, the query processor goes through three states; (*i*) it evaluates candidate nodes using predicted plan and PSI method with a time limit. If the execution timed-out, (*ii*) the processor uses the opposite PSI method and restarts the evaluation with a time limit. This step overcomes wrong predictions in the first model (Model α). If the execution timed-out again, (*iii*) the processor restarts the node evaluation, without time limits, by using the original predicted PSI method with a standard execution plan generated by selectivity-based heuristics. This step uses a best guess for the execution plan because Model β was not able to provide a credible suggestion. Since the accuracy of our classification models is high (as we show later in Section 5.4), the majority of candidate nodes should be able to finish before hitting the first timeout.

5 EVALUATION

In this section, we experimentally evaluate the performance of SmartPSI and compare it against existing competitors using several real large-scale datasets. Specifically, we show the following: (*i*) SmartPSI significantly outperforms the state-of-the-art subgraph isomorphism solutions for solving PSI queries (Section 5.2). (*ii*) Our proposed optimizations provide significant improvements including the optimized matrix-based signature generation and the proposed machine-learning approach compared to optimistic-only, pessimistic-only and the two-threaded baseline search techniques (Section 5.3). (*iii*) Our machine-learning model achieves both high accuracy and minimal run-time overhead (Section 5.4). Finally, (*iv*) we show a significant improvement in the efficiency of ScaleMine [4]; the state-of-the-art distributed frequent subgraph mining system; when subgraph isomorphism is replaced by pivoted subgraph isomorphism (Section 5.5).

5.1 Experimental setup

Datasets: We use six real graphs to evaluate the performance of SmartPSI and compare it to existing techniques. These graphs are widely used in the subgraph isomorphism and frequent subgraph mining literature [4, 9, 17]. Table 3 shows the details of each dataset. *Yeast* [8] and *Human* [26] are protein-protein interaction networks where nodes represent proteins and edges represent the interactions among them. *Cora* [1] is a citation graph where each node represents a publication with a label representing a machine learning area. In the *YouTube* [3] graph, each node represents a YouTube video and is labeled with its category, while edges connect similar videos. *Twitter* [2] models the Twitter social network where each node represents a user and edges represent follower-followee relationships. The original Twitter graph is unlabeled, we follow the same approach used in [4] to assign node labels. Finally, *Weibo* [40] is a social network crawled from Sina Weibo micro-blogging website. Each node represents a user and is labeled with the city that user lives in, while each edge represents a follower-followee relationship.

Table 3: Datasets and their characteristics

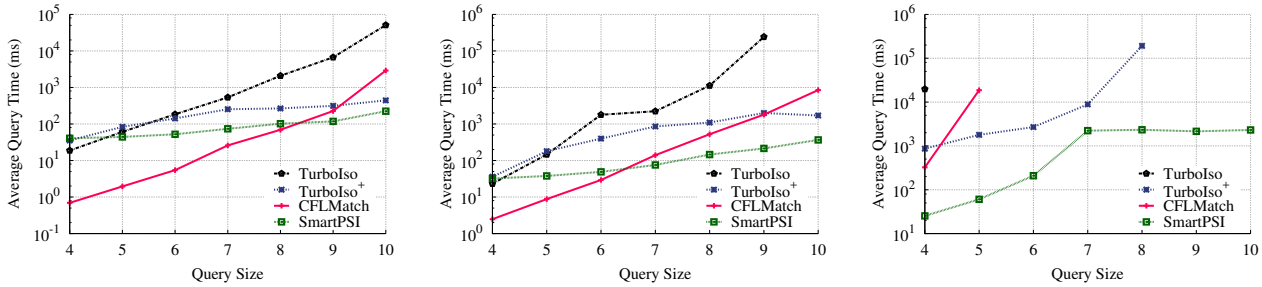
Dataset	Nodes	Edges	#node labels
Yeast [8]	3,112	12,519	71
Cora [1]	2,708	5,429	7
Human [26]	4,674	86,282	44
Youtube [3]	5,101,938	42,546,295	25
Twitter [2]	11,316,811	85,331,846	25
Weibo [40]	1,655,678	369,438,063	55

Query Graphs: From each input graph, we extract a set of random connected subgraphs as query graphs. Then, a random node is assigned as a pivot node for each query. Similar to the related work in [9, 17], a random walk with restart algorithm is used to extract 1000 query graphs for each size. Query sizes range from four to ten nodes. The resulted queries span a wide range of query complexities including paths, trees, stars and other complex shapes. Thus, they cover query loads for a wide variety of PSI applications. Unless stated otherwise, in all experiment, we use 1000 queries per query size.

Hardware Setup: SmartPSI and its competitors are deployed on a single Linux machine with 148GB RAM and two 2.1GHz AMD Opteron 6172 CPUs, each with 12 cores. We also use a distributed hardware setting for deploying ScaleMine [4] and its PSI version. In particular, we conduct ScaleMine experiments (see Section 5.5) on a Cray XC40 supercomputer which has 6,174 dual sockets compute nodes based on 16 cores Intel processors running at 2.3GHz with 128GB of RAM. We used up to 32 compute nodes with a total of 1024 cores. In all experiments, the maximum allowed time is 24 hours, any task which exceeds this limit is aborted.

5.2 Comparison with Existing Systems

SmartPSI is evaluated against three state-of-the-art subgraph isomorphism solutions: (*i*) CFL-Match [9]; the fastest reported subgraph isomorphism solution, (*ii*) TurboIso [17]; a recent subgraph ismorphism system that utilizes the query structure to combine the evaluation of similar parts of the query in a single task. and (*iii*) TurboIso$^+$; a modified version of TurboIso. Since TurboIso is a subgraph isomorphism algorithm, it finds all matches of a given query regardless of the pivot node. To optimize the performance for PSI queries, we configured TurboIso$^+$ to start evaluating the given queries using candidate matches of the pivot nodes and stop the evaluation once a pivot node match is found. Since the source code of CFL-Match is not publicly available, we could not follow a similar approach to provide a modified version of CFL-Match that is optimized for PSI queries. We also tried to



(a) Yeast Dataset (b) Cora Dataset (c) Human Dataset

Figure 7: Query Performance of SmartPSI vs. state-of-the-art Subgraph Isomorphism Systems

compare against TurboIso-Boost [30]; however, similar to what mentioned in [9], we encountered several run-time problems and we could not resolve them even after communicating with the authors. Thus, we had to omit its results. Notice that SmartPSI and all its competitors are single-machine in-memory systems. Moreover, for each input graph, we use a maximum of 1000 nodes to train the classification models in SmartPSI.

This experiment shows how available solutions (i.e., subgraph isomorphism techniques) perform when solving pivoted subgraph isomorphism queries in comparison to SmartPSI. In order to evaluate PSI queries, these algorithms use subgraph isomorphism to find all matches, then generates the list of distinct graph nodes that correspond to the pivot query node. Figure 7 shows the results on Yeast, Cora and Human datasets. The x-axis shows the query size (in number of nodes) and the y-axis shows the processing time. For Yeast (Figure 7(a)), subgraph isomorphism is a comparably easy task since this dataset is relatively small. As a result, existing techniques outperform SmartPSI on queries of size four. As the query size increases, subgraph isomorphism techniques become slower than SmartPSI. The optimizations proposed by CFLMatch allows it to outperform other systems up to queries of size 8, however, SmartPSI starts gaining momentum and becomes the fastest for larger queries. Notice that TurboIso$^+$ is significantly faster than TurboIso due to its optimization step, however, it is still slower than SmartPSI on larger queries.

For Cora dataset (Figure 7(b)), SmartPSI significantly outperforms other systems with up to two orders of magnitude. CFLMatch's optimizations worked well for small queries; it is the fastest for queries of size 4, 5 and 6. As the query size grows, similar to the Yeast dataset, CFLMatch generates huge amounts of intermediate query matches and becomes worse than SmartPSI. For queries with size 10, TurboIso fails to finish query evaluation within 24 hours. Finally for Human dataset (Figure 7(c)), both TurboIso and CFLMatch fail to evaluate most of the queries as this dataset is significantly larger and denser. TurboIso$^+$ could only complete its evaluation for queries of size up to 8. Compared to all systems, SmartPSI is able to evaluate all queries on the Human dataset with performance improvements of up to two orders of magnitude. This experiment highlights the unsuitability of the state-of-the-art subgraph isomorphism solutions to solve PSI queries even for small graphs. It also shows SmartPSI's significant improvements over the existing solutions.

5.3 SmartPSI Optimizations

Neighborhood Signature Overhead: Figure 8 compares between the matrix-based and exploration-based methods in terms

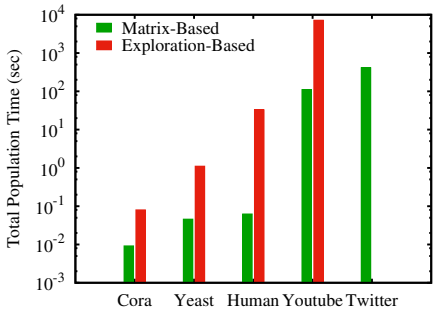
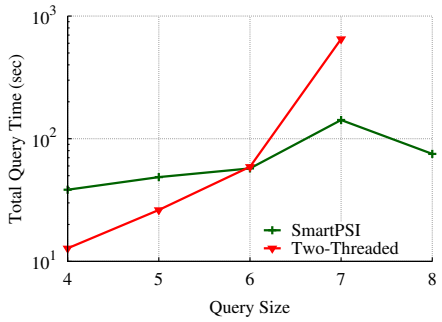


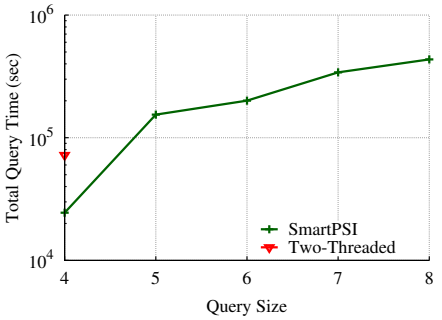
Figure 8: Exploration vs. Matrix Based Approach for Populating Neighborhood Signatures of various datasets.

of the total time for populating the neighborhood signatures (see Section 3.1). As shown in Figure 8, the processing time of both methods increases as the input graph gets larger. Exploration-based method, however, suffers from being computationally expensive especially for large datasets. For Twitter, exploration-based could not finish generating the nodes' neighborhood signatures within 24 hours. On the other hand, the matrix-based method reduces the overhead of exploration-based method significantly. It requires around 400 seconds, which is more than two orders of magnitude faster than the exploration-based method. This significant improvement is a direct result of the difference in complexities between the two methods.

SmartPSI vs. the two-threaded baseline: Figure 9 shows the run-time overhead of evaluating 100 queries on SmartPSI compared to the two-threaded solution using YouTube and Twitter datasets. The X-axis shows the query size in terms of number of nodes, while the Y-axis shows the total time to evaluate the corresponding queries. Only 100 queries are used since evaluating 1000 queries takes too much time for the two threaded approach. Since the two-threaded baseline utilizes two parallel threads, for a fair comparison, only in this experiment we run a modified version of SmartPSI that uses two concurrent threads to evaluate two different graph nodes in parallel. Using YouTube datatset, Figure 9(a) shows that the two-threaded baseline outperforms SmartPSI for small queries because it does not possess the overhead of training and prediction. Nevertheless, the two-threaded solution becomes slower than SmartPSI as queries grow in size till it exceeds the time limit on queries larger than 7 nodes. This is also true for Twitter dataset in Figure 9(b), where the two-threaded baseline can only evaluate queries of size 4. SmartPSI



(a) Youtube Dataset



(b) Twitter Dataset

Figure 9: SmartPSI (2 threads) vs. two-threaded Baseline

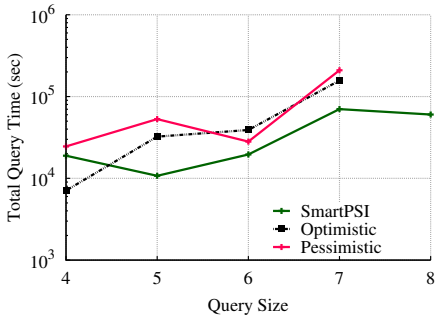


Figure 10: SmartPSI vs. Optimistic and Pessimistic on Twitter Dataset

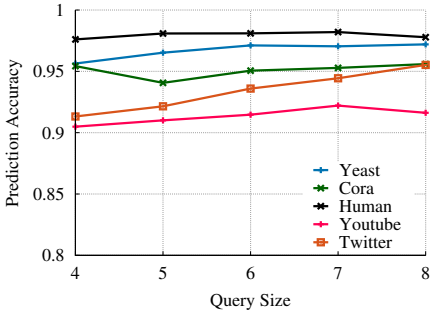


Figure 11: Prediction Accuracy

has an upper hand since the two-threaded baseline underutilizes the available resources by using two threads to finish a task that can be conducted by one thread. Moreover, the heuristic-based query execution plans, which are used in the two-threaded solution, are far less competent than the optimized plans of SmartPSI.

SmartPSI vs. optimistic and pessimistic: Figure 10 compares SmartPSI against *Pessimistic*-only and *Optimistic*-only using 10 queries for each query size on the Twitter dataset. In particular, the Pessimistic solution uses the pessimistic PSI method regardless of the type of the graph node. Likewise, Optimistic solution always use the optimistic PSI method. Moreover, the *Pessimistic* and *Optimistic* solutions use a heuristic-based query evaluation plan. SmartPSI significantly outperforms the optimistic and pessimistic methods. Furthermore, the two competitors fail to evaluate queries of size eight. The node type prediction in SmartPSI allows it to avoid exploring the full search space (compared to *Pessimistic*) and to reduce the overhead of calculating scores and sorting graph nodes accordingly (compared to *Optimistic*). Moreover, SmartPSI is able to predict the best evaluation plan for the candidate nodes, which is not achievable for the other two solutions.

5.4 Prediction Accuracy and Training Overhead

Machine Learning Models: We tested several machine learning models to build SmartPSI's classifier including Random Forest (RF), Support Vector Machines (SVM) and Neural Networks (NN). In our experiments, we found that the RF classifier provides the best accuracy. For example, it always achieves more than 95% accuracy on Human dataset compared to SVM (90%) and NN (92%). At the same time, RF is two times faster in building the model and getting predictions. We also observed similar performance results for the other datasets. Therefore, we use RF as it provides consistently good performance in terms of accuracy and training time.

Model accuracy: We conducted a set of experiments to measure the prediction accuracy of the node type prediction model using Yeast, Human, Cora, YouTube and Twitter datasets. Accuracy is calculated by comparing the result of the model's prediction to the ground truth result obtained by node evaluation using subgraph isomorphism, i.e., the number of correctly predicted node types divided by the total number of examined nodes. Figure 11 shows that the prediction accuracy of the proposed classification model is always higher than 90% for the used datasets. Furthermore, it also shows that prediction accuracy is effective and stable where the variation in predictions quality among the different datasets is small regardless of the query size.

Training overhead: Table 4 shows the overhead of models training/prediction compared to the total query evaluation time. This overhead includes model building, node type prediction and suggesting a good execution plan. In this experiment, we ran 100 queries per query size for each dataset. For small datasets and queries, the training overhead is relatively large compared to the total time. For example in Human dataset, the prediction overhead exceeds the PSI evaluation time for small queries. The reason is that query evaluation on small graphs is usually comparably fast. For larger query sizes in Human dataset, PSI evaluation becomes more expensive which reduces the relative overhead of the classification model. For larger datasets; e.g., YouTube and Twitter, the training time is very insignificant compared to PSI evaluation time. This shows that high accuracy of our classification model comes at a low computation cost, especially on large graphs.

Table 4: The overhead of model training and prediction to the total time of SmartPSI

Dataset	4	5	6	7	8
Human	75.41%	57.94%	50.85%	34.88%	33.05%
YouTube	1.13%	1.19%	1.20%	1.79%	2.83%
Twitter	1.98%	5.33%	20.15%	2.35%	5.19%

5.5 PSI Application: Frequent Subgraph Mining

This experiment highlights the advantage of using SmartPSI to boost the efficiency of an example PSI-dependent application; Frequent Subgraph Mining (FSM). For this experiment, we use ScaleMine[1] [4]; the state-of-the-art distributed FSM system. Note that ScaleMine uses subgraph isomorphism for finding occurrences for each candidate subgraph. We implemented ScaleMine+SmartPSI, a variation of ScaleMine that employs SmartPSI's techniques instead of the traditional subgraph isomorphism for computing the frequency of each candidate subgraph. Figures 12(a) and 12(b) show the performance of ScaleMine compared to the optimized version, ScaleMine+SmartPSI, using Twitter and Weibo datasets on frequency thresholds of 155K and 460K, respectively. The X-axis shows the total number of compute nodes while the Y-axis shows the overall time required to finish the mining task. For the Weibo dataset, the maximum size of allowed frequent subgraphs is set to six edges. In this experiment, ScaleMine+SmartPSI outperforms ScaleMine significantly for both datasets. For Twitter, ScaleMine+SmartPSI is up to 5X faster than ScaleMine. As for Weibo, ScaleMine+SmartPSI is up to 6X faster. By replacing the traditional subgraph isomorphism, ScaleMine+SmartPSI generates significantly less intermediate results and quickly reaches the final answer.

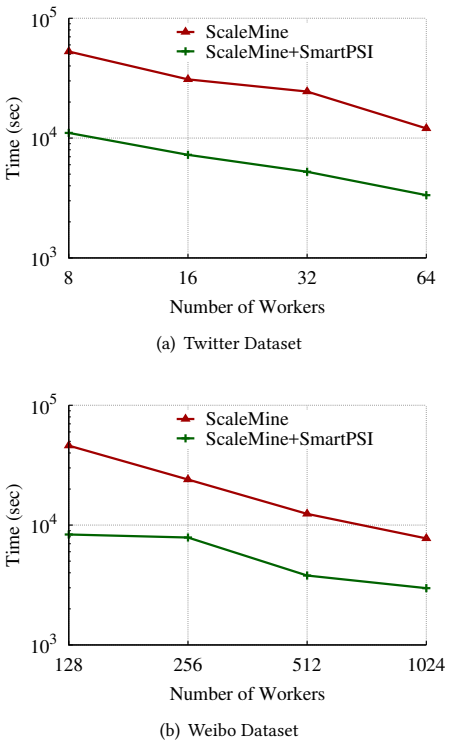


Figure 12: ScaleMine vs. ScaleMine+SmartPSI

6 RELATED WORK

6.1 Subgraph Isomorphism

Subgraph isomorphism is the bottleneck of many graph operations since it is an NP-complete problem. Therefore, many research efforts attempted to reduce its overhead in practice. The first practical algorithm that addresses this problem follows a backtracking approach [35]. Since then, several performance enhancements were proposed, ranging from CSP-based techniques [29], search order optimization [18], indexing [38] and parallelization [33]. As reported in [26], GraphQL [18] is considered one of the best subgraph isomorphism techniques, though its performance is not stable over the different datasets. GraphQL prunes the search space by using local and global graph information. Moreover, it utilizes a search order optimization technique based on a global cost model.

TurboIso [17] is a recent approach that outperforms previous techniques by grouping similar parts of the query and process them at once. Moreover, TurboIso adapts its search order plan according to each input graph node. BoostIso [30] is a plugin that enhances the efficiency of other approaches. It is able to avoid duplicate computations by exploiting the relationship among the input graph nodes. CFLMatch [9] is the current state-of-the-art technique for subgraph isomorphism. It decomposes the query graph into a core and multiple trees. Such decomposition allows better search order optimization. Moreover, CFLMatch avoids redundant evaluations of tree leaf nodes by representing the mappings of these nodes in a compressed way. Although these techniques lead to relatively significant improvements in the domain of subgraph isomorphism, the algorithm itself is impractical especially for large graphs. The main reason behind this limitation of subgraph isomorphism is the excessive number of results it generates.

The closest approach to our work is Ψ-framework [22]. It utilizes both isomorphic query re-writings and existing alternative algorithms in parallel to improve the performance of subgraph isomorphism. Ψ-framework is similar to our two-threaded prototype (Section 4.1) where it requires more processing power to achieve a reasonable performance. Unlike SmartPSI, Ψ-framework is unable to decide on which approach is better to use during runtime.

6.2 Neighborhood Signature

A graph node can be represented by a signature that reflects the neighborhood structures around it. This neighborhood signature can be represented as a feature vector and used in many important applications such as abnormal nodes detection [6] and predicting missing links [27].

Neighborhood signatures are also used for indexing and pruning. GADDI [41] is a subgraph matching solution that builds an index based on graph structures that exist between any pair of vertices. Based on this index,a two way pruning technique is used to efficiently prune candidate matches. GADDI suffers from the excessive cost required for creating its index. NOVA [43] and DSI [24] maintain the paths to and from surrounding nodes. These paths are used to filter out unqualified graph nodes. Both Nova and DSI rely on maintaining a huge number of paths. As such, both technique comprise excessive overhead, especially

[1] https://github.com/ehab-abdelhamid/ScaleMine

when processing large-scale dense graphs. (NNT) [36] is a more effective index that relies on finding the trees around each graph node. NNT requires the use of tree matching algorithms which makes pruning and matching more complex.

The above approaches are expensive, especially for large-scale dense graphs. Other simpler indexes were proposed such as SMS [42] which maintains the list of labels and node degrees that appear in the neighborhood. Although this indexing technique is simple, it significantly outperforms NOVA and GADDI. Tale [34] is a system that finds approximate matches of query subgraphs in an input graph. It relies on an index, called NH-Index, which maintains information about the direct neighbors of each graph node. NH-Index maintains information such as node label, node degree, neighbor nodes and how they are connected. Both SMS and Tale focus on the direct neighbors. Thus, their pruning and descriptive power is limited. More recently, k-hop label [7] extends its indexing by considering all neighbor nodes within K-hops. Such representation extends beyond direct neighbors, but it lacks effective representation of the surrounding structure. Finally, label propagation [23] captures the neighborhood structure around each graph node using a list of labels along with their corresponding weights. Weights represent how labels are placed and connected around the node under consideration. Thus, more informative structural information is captured.

7 CONCLUSION

In this paper, we propose SmartPSI; an efficient machine-learning based system for evaluating Pivoted Subgraph Isomorphism (PSI) queries. It is based on two effective PSI algorithms; each is optimized for a certain graph node type. SmartPSI is also coupled with a machine-learning model to predict the graph node type and call the appropriate PSI algorithm accordingly. It also uses a machine-learning based query optimizer to select execution plans that reduce the total query run-time by minimizing intermediate query results. Our experiments show that SmartPSI is efficient and able to scale PSI evaluation where it is up to two orders of magnitude faster compared to existing subgraph isomorphism techniques. Furthermore, when applied to Frequent Subgraph Mining (FSM), PSI improves the performance of the state-of-the-art distributed FSM system by up to a factor of 6X.

ACKNOWLEDGMENTS

This research made use of the Supercomputing Laboratory resources at King Abdullah University of Science & Technology (KAUST), Saudi Arabia.

REFERENCES

[1] 2017. Cora Dataset. http://linqs.umiacs.umd.edu/projects/projects/lbc/. (2017).
[2] 2017. Twitter Dataset. http://socialcomputing.asu.edu/datasets/Twitter. (2017).
[3] 2017. YouTube Dataset. http://netsg.cs.sfu.ca/youtubedata/. (2017).
[4] Ehab Abdelhamid, Ibrahim Abdelaziz, Panos Kalnis, Zuhair Khayyat, and Fuad Jamour. 2016. Scalemine: Scalable Parallel Frequent Subgraph Mining in a Single Large Graph. In *Proceedings of SC*.
[5] E. Abdelhamid, M. Canim, M. Sadoghi, B. Bhattacharjee, Y. Chang, and P. Kalnis. 2017. Incremental Frequent Subgraph Mining on Large Evolving Graphs. *IEEE Transactions on Knowledge and Data Engineering* 29, 12 (December 2017), 2710–2723.
[6] Leman Akoglu, Mary McGlohon, and Christos Faloutsos. 2010. Oddball: Spotting anomalies in weighted graphs. In *Proceedings of PAKDD*.
[7] Pranay Anchuri, Mohammed J Zaki, Omer Barkol, Shahar Golan, and Moshe Shamy. 2013. Approximate Graph Mining with Label Costs. In *Proceedings of SIGKDD*.
[8] Saurabh Asthana, Oliver D King, Francis D Gibbons, and Frederick P Roth. 2004. Predicting Protein Complex Membership using Probabilistic Network Reliability. *Genome Research* 14, 6 (2004).
[9] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient Subgraph Matching by Postponing Cartesian Products. In *Proceedings of SIGMOD*.
[10] Leo Breiman. 2001. Random Forests. *Machine Learning* 45, 1 (2001), 5–32.
[11] Björn Bringmann and Siegfried Nijssen. 2008. What is frequent in a single graph?. In *Pacific-Asia Conference on Knowledge Discovery and Data Mining*. Springer, 858–863.
[12] Young-Rae Cho and Aidong Zhang. 2010. Predicting Protein Function by Frequent Functional Association Pattern Mining in Protein Interaction Networks. *IEEE Transactions on Information Technology in Biomedicine* 14, 1 (2010).
[13] Mohammed Elseidy, Ehab Abdelhamid, Spiros Skiadopoulos, and Panos Kalnis. 2014. GraMi: Frequent subgraph and pattern mining in a single large graph. *PVLDB* 7 (2014).
[14] Michael R. Garey and David S. Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co.
[15] Jialong Han and Ji-Rong Wen. 2013. Mining frequent neighborhood patterns in a large labeled graph. In *Proceedings of CIKM*.
[16] Jialong Han, Kai Zheng, Aixin Sun, Shuo Shang, and Ji-Rong Wen. 2016. Discovering Neighborhood Pattern Queries by sample answers in knowledge base. In *Proceedings of ICDE*.
[17] Wook-Shin Han, Jinsoo Lee, and Jeong-Hoon Lee. 2013. Turboiso: Towards Ultrafast and Robust Subgraph Isomorphism Search in Large Graph Databases. In *Proceedings of SIGMOD*.
[18] Huahai He and Ambuj K. Singh. 2008. Graphs-at-a-time: Auery Language and Access Methods for Graph Databases. In *Proceedings of SIGMOD*.
[19] Keith Henderson, Brian Gallagher, Tina Eliassi-Rad, Hanghang Tong, Sugato Basu, Leman Akoglu, Danai Koutra, Christos Faloutsos, and Lei Li. 2012. Rolx: Structural Role Extraction & Mining in Large Graphs. In *Proceedings of SIGKDD*. ACM.
[20] CA James, D Weininger, and J Delany. 2003. Daylight Theory Manual Daylight Version 4.82. Daylight Chemical Information Systems. (2003).
[21] Glen Jeh and Jennifer Widom. 2002. SimRank: A Measure of Structural-context Similarity. In *Proceedings of SIGKDD*. ACM.
[22] Foteini Katsarou, Nikos Ntarmos, and Peter Triantafillou. 2017. Subgraph querying with parallel use of query rewritings and alternative algorithms. (2017).
[23] Arijit Khan, Xifeng Yan, and Kun-Lung Wu. 2010. Towards Proximity Pattern Mining in Large Graphs. In *Proceedings of SIGMOD*.
[24] Yubo Kou, Yukun Li, and Xiaofeng Meng. 2010. DSI: A Method for Indexing Large Graphs using Distance Set. In *Proceedings of WAIM*.
[25] Michael Lappe and Liisa Holm. 2004. Unraveling Protein Interaction Networks with Near-optimal Efficiency. *Nature Biotechnology* 22, 1 (2004).
[26] Jinsoo Lee, Wook-Shin Han, Romans Kasperovics, and Jeong-Hoon Lee. 2012. An In-depth Comparison of Subgraph Isomorphism Algorithms in Graph Databases. *PVLDB* 6, 2 (2012).
[27] Ryan N Lichtenwalter, Jake T Lussier, and Nitesh V Chawla. 2010. New Perspectives and Methods in Link Prediction. In *Proceedings of SIGKDD*. ACM.
[28] Andrew McCallum, Xuerui Wang, and Andrés Corrada-Emmanuel. 2007. Topic and Role Discovery in Social Networks with Experiments on Enron and Academic Email. *Journal of Artificial Intelligence Research* 30 (2007).
[29] J J McGregor. 1979. Relational Consistency Algorithms and their Application in Finding Subgraph and Graph Isomorphisms. *Information Sciences* 19 (1979).
[30] Xuguang Ren and Junhu Wang. 2015. Exploiting vertex relationships in speeding up subgraph isomorphism over large graphs. *PVLDB* 8 (2015).
[31] Sherif Sakr, Faisal Moeen Orakzai, Ibrahim Abdelaziz, and Zuhair Khayyat. 2016. *Large-Scale graph processing using Apache giraph*. Springer.
[32] Tom AB Snijders, Philippa E Pattison, Garry L Robins, and Mark S Handcock. 2006. New Specifications for Exponential Random Graph Models. *Sociological Methodology* 36, 1 (2006).
[33] Zhao Sun, Hongzhi Wang, Haixun Wang, Bin Shao, and Jianzhong Li. 2012. Efficient Subgraph Matching on Billion Node Graphs. *PVLDB* 5, 9 (May 2012).
[34] Yuanyuan Tian and Jignesh M Patel. 2008. Tale: A Tool for Approximate Large Graph Matching. In *Proceedings of ICDE*.
[35] J. R. Ullmann. 1976. An Algorithm for Subgraph Isomorphism. *Journal of ACM* 23 (1976). Issue 1.
[36] Changliang Wang and Lei Chen. 2009. Continuous Subgraph Pattern Search over Graph Streams. In *Proceedings of ICDE*.
[37] Xifeng Yan and Jiawei Han. 2002. gspan: Graph-based Substructure Pattern Mining. In *Proceedings of ICDM*.
[38] Xifeng Yan, Philip S. Yu, and Jiawei Han. 2004. Graph indexing: A Frequent Structure-based Approach. In *Proceedings of SIGMOD*.
[39] Yu Yang, Jian Pei, and Abdullah Al-Barakati. 2017. Measuring In-network Node Similarity Based on Neighborhoods: A Unified Parametric Approach. *Knowledge and Information Systems* (2017).
[40] Jing Zhang, Biao Liu, Jie Tang, Ting Chen, and Juanzi Li. 2013. Social Influence Locality for Modeling Retweeting Behaviors. In *Proceedings of IJCAI*.
[41] Shijie Zhang, Shirong Li, and Jiong Yang. 2009. GADDI: Distance Index Based Subgraph Matching in Biological Networks. In *Proceedings of EDBT*.
[42] Weiguo Zheng, Lei Zou, and Dongyan Zhao. 2011. Answering Subgraph Queries over Large Graphs. In *Proceedings of WAIM*. Springer.
[43] Ke Zhu, Ying Zhang, Xuemin Lin, Gaoping Zhu, and Wei Wang. 2010. Nova: A Novel and Efficient Framework for Finding Subgraph Isomorphism Mappings in Large Graphs. In *Proceedings of DASFAA*.