Enumerating Graph Pattern Matches with ML Oracles

Paper id: 675

Abstract

This paper explores an ML-based method for SubIso, the graph pattern matching problem. Given a graph G and a pattern Q, SubIso is to find all subgraphs of G that are isomorphic to Q. We show that SubIso is EnumP-complete, where EnumP is the class of enumeration problems that is the enumeration counterpart of NP. We study revisions $\mathsf{VF3}_M$ of VF3, a well-known algorithm for SubIso, by taking an ML model $\mathcal{M}$ as an oracle. Model $\mathcal{M}$ predicts candidate matches, and $\mathsf{VF3}_M$ verifies the matches, reducing costly backtracking. We investigate whether $\mathsf{VF3}_M$ can be (a) output polynomial, *i.e.,* its time cost can be expressed as a polynomial in the sizes of the input and output, (b) consistent, *i.e.,* its accuracy approaches 100% when $\mathcal{M}$ gives error-free predictions, and (c) β-robust, *i.e.,* its accuracy is at least β even when $\mathcal{M}$ gives arbitrarily bad predictions.

We establish several results, positive and negative. On the negative side, we show that it is impossible for $\mathsf{VF3}_M$ to be both output polynomial and β-robust with a positive constant β unless P = fewP, a problem as hard as P = NP. On the positive side, we develop three versions of $\mathsf{VF3}_M$ that are (a) consistent, and (b) either 1-robust or output polynomial with a high probability. We also train a model $\mathcal{M}$ for $\mathsf{VF3}_M$. Using real-life and synthetic data, we show that $\mathsf{VF3}_M$ is up to 15.75×–21× faster than VF3, with F1-score 0.98.

1 Introduction

Graph pattern matching is the problem that, given a pattern Q and a graph G, computes the set $Q(G)$ of all matches of Q in G, *i.e.,* all subgraphs of G that are isomorphic to Q. It has been widely used in graph databases, biochemistry, computer vision, social network analysis and fraud detection, among other things. We refer to the problem as SubIso. Its decision version is to decide, given Q and G, whether there exists a match of Q in G. The decision problem is NP-complete (cf. [43]). Hence unless P = NP, it is unlikely to find an exact polynomial-time (PTIME) algorithm for SubIso.

For PTIME algorithms of an intractable problem, the study has mostly focused on its combinatorial optimization version to find an optimal (maximum or minimum) solution, based on one of the following approaches: (a) approximation, to find a solution that is at most a factor from the optimal one, (b) heuristic, which may work well in many cases but offer no proof for its accuracy and efficiency, (3) randomization, to get a fast average running time but allow to fail with a small probability, and (4) parameterization, to get PTIME algorithms when certain input parameters are fixed [42].

While the decision version of SubIso is hard, the enumeration problem SubIso is more challenging. In practice we often need to find all the answers to a computational problem. When we answer an SQL query Q', we compute all answers to Q' in a dataset, not just the existence of a solution or an optimal one. For SubIso, we have to find all matches of a pattern Q in a graph G in order to, *e.g.,* answer an SPARQL pattern query, or to compute the confidence of rules [41]. However, SubIso is inherently exponential since the number of matches may be $O(|G|^{|Q|})$ in the worst case. Moreover, the prior approaches to developing algorithms for decision and optimization problems no longer work well for enumeration problems.

Does there exist an efficient and accurate algorithm for SubIso?

An ML-based approach. We explore an approach towards SubIso by taking machine learning (ML) models as oracles. For PoC, we pick VF3 [28, 29], a well-known exact algorithm for SubIso, and revise it by employing an ML model $\mathcal{M}$ to predict whether a partial match ρ of pattern Q is *valid, i.e.,* it can be extended to a full match in G. We follow the prediction of $\mathcal{M}$, recursively enumerate matches as in VF3, but may skip its costly backtracking when $\mathcal{M}$ predicts ρ to be invalid. We refer to such revisions of VF3 as $\mathsf{VF3}_M$.

Intuitively, we want to unify algorithmic methods and ML predictions. On the one hand, SubIso algorithms have been studied for decades. On the other hand, ML models have proven effective in a variety of applications. Hence, it is natural to combine the two and take advantage of both. Moreover, even when we cannot find a good model $\mathcal{M}$ now, when ML techniques evolve, the quality of $\mathcal{M}$ predictions will improve and $\mathsf{VF3}_M$ will get more accurate.

The idea is not entirely new. Algorithms with ML predictions have been studied for decision problems [24, 25, 32, 34, 64, 70, 76, 77, 82, 85, 95]. There has also been work on learning meta-algorithms for optimization problems [61]. However, the prior methods (a) do not carry over to SubIso, an enumeration problem, (b) rely on ML alone, instead of unifying algorithmic methods and ML predictions, and (c) guarantee neither the accuracy nor the efficiency (see below).

Challenges. It is nontrivial to develop $\mathsf{VF3}_M$ algorithms that are both accurate and efficient, measured by the following criteria.

(1) Output polynomial. Ideally, we want $\mathsf{VF3}_M$ to *enumerate matches* in PTIME in the sizes of the input and output (assuming that applying $\mathcal{M}$ is in PTIME after $\mathcal{M}$ is trained, as commonly found in practice). An enumeration problem with such an exact algorithm is considered "tractable" and is in the complexity class OutputP [27]. However, we show that SubIso is EnumP-complete, where EnumP [92] is the class of enumeration problems for which the correctness of a candidate solution can be *checked* in PTIME, and is the enumeration equivalent of NP. Hence, SubIso is one of the hardest problems in EnumP. It is known that EnumP ≠ OutputP unless P = NP [27].

(2) Consistency and robustness. Obviously, the accuracy of $\mathsf{VF3}_M$ is impacted by the rate of false positives (FPs) and false negatives (FNs) of its embedded ML model $\mathcal{M}$. This said, we want $\mathsf{VF3}_M$ to be both (a) consistent, *i.e.,* it accuracy approaches 100% when the predictions of $\mathcal{M}$ are error-free, and (b) β-robust for a bound β, *i.e.,* its accuracy is at least β even when $\mathcal{M}$ gives arbitrarily bad predictions. That is, the more accurate $\mathcal{M}$ is, the more reliable $\mathsf{VF3}_M$ is. Moreover, $\mathsf{VF3}_M$ still performs well even when $\mathcal{M}$ is inaccurate.

Does there exist $\mathsf{VF3}_M$ such that it is consistent, β-robust (for a positive constant β) and output polynomial at the same time?

Contributions and organization. This paper aims to develop a better understanding of these issues. We establish several results about SubIso; some results are positive, while some are negative.

(1) EnumP-hardness (Section 2). We show that SubIso is EnumP-complete. Thus unless P = NP, one cannot expect to find an exact algorithm for SubIso that is *output polynomial, i.e.,* its cost can be expressed as a polynomial in the sizes of input and output. On the positive side, if we find an output-polynomial enumeration algorithm for SubIso, then all the enumeration problems in EnumP will have a "tractable" accurate algorithm, since all such problems can be reduced to SubIso with PTIME parsimonious reductions, *i.e.,* transformations that preserve the number of solutions.

(2) Algorithms $\mathsf{VF3}_{\mathcal{M}}$ (Section 3). We then develop three $\mathsf{VF3}_{\mathcal{M}}$ algorithms for SubIso by incorporating an ML model $\mathcal{M}$ into VF3, denoted by $\mathsf{VF3}^N_{\mathcal{M}}$, $\mathsf{VF3}^O_{\mathcal{M}}$ and $\mathsf{VF3}^D_{\mathcal{M}}$, respectively, adopting different backup plans when the predictions of $\mathcal{M}$ may be bad. $\mathsf{VF3}^N_{\mathcal{M}}$ opts to completely trust $\mathcal{M}$ and follows its predictions without backup. $\mathsf{VF3}^O_{\mathcal{M}}$ trusts $\mathcal{M}$ only if its confidence is high, and resorts to VF3 as a backup otherwise. $\mathsf{VF3}^D_{\mathcal{M}}$ conducts deep checking if $\mathcal{M}$ predicts false, to further reduce FNs and improve the robustness. The three strive for efficiency, accuracy and their balance, respectively.

(3) Performance guarantees (Section 4). We show a negative result: unless P = fewP, there exists no algorithm for SubIso that is both output-polynomial and β-robust for a positive constant β; here fewP is the class of problems that can be recognized by a nondeterministic Turing machine in PTIME and have polynomially-bounded number of solutions [9]. We know that fewP = P is as hard as P = NP [36].

On the positive side, we show the following: (a) the precision of the three $\mathsf{VF3}_{\mathcal{M}}$ algorithms is always 1, *i.e.,* any solution found by $\mathsf{VF3}_{\mathcal{M}}$ is guaranteed to be correct for SubIso; hence they have no FP; (b) all the algorithms are consistent, *i.e.,* its accuracy (F1-score) warrants to be 100% when $\mathcal{M}$ is error-free, *i.e.,* there is neither FP nor FN in this case; (c) $\mathsf{VF3}^O_{\mathcal{M}}$ and $\mathsf{VF3}^D_{\mathcal{M}}$ are "almost" 1-robust with recall = 1 when $\mathcal{M}$ predictions have a high confidence; and (d) with a high probability, $\mathsf{VF3}^N_{\mathcal{M}}$ is "almost" output polynomial and is β-robust for β determined by the size of output.

(4) ML model $\mathcal{M}$ (Section 5). We train an ML model $\mathcal{M}$ for $\mathsf{VF3}_{\mathcal{M}}$ algorithms. Departing from previous models for subgraph matching or counting, it predicts whether a partial match is valid. We adopt IDGNN [107] for vertex embedding, develop a partition-based strategy to embed patterns and graphs and encode the given partial match, make a prediction using the order-embedding space, and return a confidence of the prediction via a multilayer perceptron. Moreover, we enrich training data to ensure the robustness of $\mathcal{M}$.

(5) Effectiveness (Section 6). Using real-life and synthetic graphs, we experimentally find the following. (a) By unifying algorithmic methods and ML predictions, $\mathsf{VF3}^N_{\mathcal{M}}$, $\mathsf{VF3}^O_{\mathcal{M}}$ and $\mathsf{VF3}^D_{\mathcal{M}}$ speed up VF3 by 10.35×, 9.63× and 8.40× on average, respectively, up to 21×, 19.38× and 15.75×. (b) The $\mathsf{VF3}_{\mathcal{M}}$ algorithms are accurate. They are consistent (*i.e.,* their precision is constantly 1), and their average F1 scores are 0.44, 0.95 and 0.98, respectively, up to 0.62, 0.99 and 0.99. (c) Algorithms $\mathsf{VF3}^O_{\mathcal{M}}$ and $\mathsf{VF3}^D_{\mathcal{M}}$ are robust: their F1 scores are above 0.9, even when the embedded ML model has error rate 0.5. (d) Our ML model $\mathcal{M}$ is 15% more accurate than subgraph matching and counting models on average, up to 29%.

We discuss related work in Section 7 and future work in Section 8. The detailed proofs of the results of the paper are deferred to [2].

2 SubIso, EnumP, and OutputP

This section reviews the SubIso problem and complexity classes EnumP and OutputP. We show that SubIso is EnumP-complete.

Graphs. Assume a countably infinite set Γ of symbols for labels. We consider *labeled directed graphs* $G = (V, E, L)$, where (a) V is a finite set of vertices, (b) $E \subseteq V \times \Gamma \times V$ is a finite set of edges in which (v_1, l, v_2) is an edge labeled $l \in \Gamma$ from v_1 to v_2, and (c) L is a function such that for each vertex $v \in V$, $L(v) \in \Gamma$ is its label.

Graph pattern matching. We next present SubIso.

Patterns. A *graph pattern* is a connected graph $Q = (V_Q, E_Q, L_Q)$, where (a) V_Q (resp. E_Q) is a finite set of pattern vertices (resp. edges), and (b) L_Q assigns a label $L_Q(u)$ to each pattern vertex $u \in V_Q$.

Matching. A *full match* of pattern Q in a graph G is a bijective mapping from V_Q to V' of a subgraph $G' = (V', E', L')$ of G such that (a) for each vertex u in V_Q, $L_Q(u) = L'(h(u))$; and (b) $e = (u, l, u')$ is an edge in E_Q iff $e' = (h(u), l, h(u'))$ is an edge in E' for label l in Γ.

SubIso. The numeration problem SubIso is stated as follows.

- *Input*: A graph pattern Q and a graph G.
- *Output*: The set $Q(G)$ of all full matches of Q in G.

The set $Q(G)$ may have $O(|G|^{|Q|})$ many matches of Q in G.

Partial match. A *partial match* of Q in G is a bijective mapping ρ from a *subgraph* of Q to a subgraph of G as above. Match ρ is called *valid* if it can be extended to be a full match h of Q in G such that for each $u \in V_Q$, $\rho(u) = h(u)$ if $\rho(u)$ is defined. Full matches are a special case of partial matches that cannot be further extended.

Example 1: Given graph G and pattern Q depicted in Figure 1, there exists a unique full match of Q in G, *i.e.,* the leftmost part of G. Consider two partial matches ρ_1 and ρ_2 (colored in red and indicated by blue circles, respectively). (1) Partial match ρ_1 is valid, as it can be extended to the match in G. (2) However, ρ_2 is not valid since when mapping u_5 to w_2^1, vertex u_6 finds no match in G. □

Complexity classes. We consider two complexity classes EnumP and OutputP (see [27]). Let Σ be a finite alphabet and Σ^* be the set of finite words built on Σ. For a binary predicate $B \subseteq \Sigma^* \times \Sigma^*$, we write $B(x)$ for the set of y such that $B(x, y)$ holds. To simplify the discussion, we consider predicates B such that $B(x)$ is finite for all x.

The *enumeration problem* Π_B is the function that associates $B(x)$ to x. Intuitively, it lists all solutions y to problem instance x.

EnumP. EnumP is the class of enumeration problems Π_B such that $B \in$ P, *i.e.,* it is in PTIME to check whether $B(x, y)$ holds (whether y is a solution to x). It is the equivalent of NP in enumeration.

An EnumP-*complete problem* is a problem in EnumP to which all problems in EnumP can be reduced by parsimonious reductions. A *parsimonious reduction* R from a problem A to problem B is a PTIME transformation from instances of A to instances of B such that for any instance x of A, the number of solutions to x is equal to the number of solutions to instance $R(x)$ of problem B [81].

Proposition 1: SubIso *is* EnumP-*complete.* □

Proof sketch: We show that SubIso is EnumP-complete by parsimonious reduction from Π_{SAT}. Problem Π_{SAT} is to enumerate all

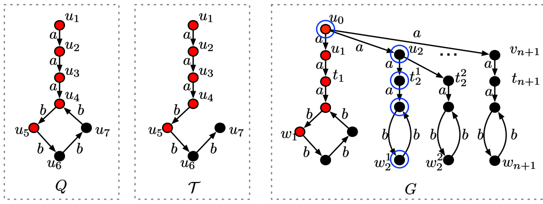


Figure 1: The difference between VF3 **and** $\mathsf{VF3}_M$

solutions of a 3NF formula, and is EnumP-complete [92]. Given a 3SAT formula $\Phi = C_1 \wedge \ldots \wedge C_n$, we give a reduction by constructing a graph G that includes seven vertices for each clause C_i ($i \in [1, n]$) and a pattern Q that is a clique of size n (the number of clauses). □

<u>*OutputP*</u>. A problem $\Pi_B \in$ EnumP is in OutputP if for all instances x of Π_B, the time for computing all solutions in $B(x)$ is bounded by a polynomial in the size $|x|$ of input and the size $|B(x)|$ of the output. Such a problem is said to be *output polynomial*.

Intuitively, Π_B is in OutputP if it can be solved in PTIME in the sizes of the input and output. The size of output is taken into account since the number of solutions may be exponential in the size of the input, *e.g.,* SubIso. The cost is measured as the total time needed to compute all solutions in $B(x)$ to input x, using a RAM machine.

Abusing the notation, we say that an enumeration algorithm for Π_B is *output polynomial* if it takes PTIME in $|x|$ and $|B(x)|$.

3 Programming with ML Oracles

This section develops revisions $\mathsf{VF3}_M$ of algorithm VF3 [28] by incorporating an ML oracle $\mathcal{M}$. We first review VF3 (Section 3.1) and present a template for $\mathsf{VF3}_M$ (Section 3.2). Based on the template, we then develop three $\mathsf{VF3}_M$ algorithms (Section 3.3).

3.1 Algorithm VF3

We first review VF3 [28, 29], a popular exact algorithm for SubIso. Given a pattern Q and a graph G, it employs a recursive procedure Enumerate to find all full matches of Q in G, and reduces the search space by using efficient heuristic rules. Here we improve the original algorithm VF3 of [28, 29] by incorporating optimization strategies that are identified in [112] and verified effective for SubIso.

Algorithm. Given Q and G, VF3 computes the set $Q(G)$ of matches of Q in G. It first sets a matching order O of Q using the Maximum Likelihood Estimation method. Here a *matching order* is a permutation of the pattern vertices V_Q in Q. Then it assigns each vertex in G or Q to a class such that vertices from the same class can be matched (line 2). Intuitively, it computes a set $C(u)$ of candidates matches from G for each pattern vertex u in Q. Then, it constructs a tree representation $\mathcal{T}$ of Q based on O, *i.e.,* a vertex u_p is a parent of u in $\mathcal{T}$ if u_p has an edge connected to u and appears before u in O. It also builds ρ_0, the initial partial match $\emptyset$. After these, it enumerates matches in $Q(G)$ following the order O via procedure Enumerate.

Procedure Enumerate. As shown in Figure 2, given graph G, pattern Q, candidates C for pattern vertices, a tree representation $\mathcal{T}$ of Q and a partial match ρ, Enumerate recursively extends the partial match ρ to full matches, if possible, following the order O.

Enumerate works as follows. It first checks whether ρ is already a full match; if so, it returns ρ as part of Output (lines 1-2). Otherwise, it checks whether ρ is invalid, *e.g.,* a pattern vertex in ρ has more neighbors than its candidate matches in ρ; if so, it returns $\emptyset$, *i.e.,* the current partial match cannot be extended to a full match (lines 3-4).

Input: A graph G, a pattern Q, a matching order O, a classification C, a tree representation $\mathcal{T}$ of Q, and the current partial match ρ.
Output: All full matches of Q in G that extend ρ.

```
1.  if IsGoal(ρ) then
2.      return {ρ};
3.  if IsDead(ρ) then
4.      return ∅;
5.  (u_c, v_c) := (⊥, ⊥);
6.  (u_n, v_n) := GetCan(G, Q, O, C, ρ, T, (u_c, v_c));
7.  while (u_n, v_n) ≠ (⊥, ⊥) do
8.      if IsFeasible(G, Q, ρ, u_n, v_n) then
9.          ρ_c := ρ + (u_n, v_n);
10.         ΔOutput := Enumerate(G, Q, O, C, T, ρ_c);
11.         Output := Output ∪ ΔOutput;
12.     (u_n, v_n) := GetCan(G, Q, O, C, ρ, T, (u_n, v_n));
13. return Output;
```

Figure 2: Procedure Enumerate

Otherwise, Enumerate iteratively extends partial match ρ by mapping the next pattern vertex u_n to its candidates $v \in C(u)$ (lines 5-12). More specifically, it first identifies a vertex v_n in $C(u_n)$ that may match u_n (lines 5-6). That is, v_n and u_n are in the same class, and they have "isomorphic" connections with vertices in ρ, *i.e.,* if u_n has an edge (u_n, l, u_1) (resp. (u_1, l, u_n)) in Q, then v has an edge $(v_n, l, \rho(u_1))$ (resp. $(\rho(u_1), l, v_n)$) in G. Function GetCan selects the next candidate for pattern vertices from the list of candidate matches; and (u_c, v_c)=$(\perp, \perp)$ marks the end of the list.

Expanding ρ with (u_n, v_n) suggested by GetCan, denoted by $\rho + (u_n, v_n)$, Enumerate checks whether $\rho + (u_n, v_n)$ makes a bijective mapping. It calls Boolean function IsFeasible, which inspects the 2-hop neighbors of u_n and v_n for further extensions, using a set of heuristic rules (line 8). If $\rho + (u_n, v_n)$ passes the check, it recursively invokes Enumerate to expand $\rho + (u_n, v_n)$ (lines 10-11). After processing v_n, it continues to checks other vertices in $C(u_n)$ (line 12), such that all candidates in $C(u_n)$ are examined (line 7).

When processing $\rho + (u_n, v_n)$, Enumerate may recurse to a depth, *i.e.,* after it expands $\rho + (u_n, v_n)$ with $(u_{n+1}, v_{n+1}), \ldots, (u_{n+m}, v_{n+m})$, it finds that the expansion is "dead" (line 3), *i.e.,* $\rho + (u_n, v_n)$ does not lead to a full match. At this point, Enumerate has to *backtrack*, and continues the search from $\rho + (u_n, v'_n)$ for a different candidate v'_n of u_n. Backtracking is costly when m is large, and the efforts for expanding $\rho + (u_n, v_n)$ are wasted.

Optimizations. The VF3 algorithm of Figure 2 extends the original algorithm of [28, 29] with the following optimization strategies.

<u>*Classification*</u>. We adopt graded simulation [80] to compute the candidate set $C(u)$. Weaker than subgraph isomorphism, graded simulation pre-filters out vertices in G that cannot match u. Given a graph G and a pattern Q, it computes the maximum binary relation $\mathcal{G} \subseteq V_Q \times V_G$ such that $(u, v) \in \mathcal{G}$ if and only if the one-hop neighbors of u can be "mapped" to those of v. More specifically, $(u, v) \in \mathcal{G}$ iff (1) u and v carry the same label; (2) if u has m distinct outgoing edges $(u, l_1, u_1), \ldots, (u, l_m, u_m)$, then v also has m distinct outgoing edges $(v, l_1, v_1), \ldots, (v, l_m, v_m)$ such that $(u_1, v_1), \ldots, (u_m, v_m) \in \mathcal{G}$; similarly for distinct incoming edges of u. Relation $\mathcal{G}$ can be computed in $O(|V|(|V| + |E|)|V_Q|(|V_Q| + |E_Q|))$ time [78].

We include in $C(u)$ all such vertices v in G that $(u, v) \in \mathcal{G}$.

<u>*Matching order*</u>. We adopt the strategy proposed in [23] to give the matching order O. It starts with the pattern vertex having the

Input: A graph G, a pattern Q, a partial match ρ, a vertex u in Q and a vertex v in G.
Output: Whether there exists a full match extended from ρ.
1. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) \geq \delta_\mathsf{T}$ **then**
2.   **return** $\mathsf{ReduceFP}(G, Q, O, \mathcal{T}, \rho)$;
3. **return** $\mathsf{ReduceFN}(G, Q, O, \mathcal{T}, \rho)$;

Figure 3: Procedure $\mathsf{IsFeasible}_\mathsf{M}$

largest degree, and then iteratively picks the pattern vertex that has the most backward neighbors. As shown in [112], this strategy works well for SubIso. Other methods to compute O can also be used, *e.g.,* topological ordering [23] and reinforcement learning [97].

Example 2: Continuing with Example 1, VF3 finds matches of Q in G as follows. (1) It first sets a matching order O of Q, *e.g.,* $u_1, \ldots, u_7$, represented by tree $\mathcal{T}$ in Figure 1. (2) Following O, it recursively enumerates matches in $Q(G)$, mapping u_1 to v_0 first, followed by finding matches of $u_3, \ldots, u_7$. It returns the unique full match. □

3.2 Algorithm Template for $\mathsf{VF3}_M$

We next outline a template $\mathsf{VF3}_M$ for revising the VF3 algorithm, by incorporating an ML model $\mathcal{M}$ into VF3 as an oracle.

ML oracle $\mathcal{M}$. As will be seen in Section 5, we will train an ML model $\mathcal{M}$ that, given a graph G, a pattern Q and a partial match ρ, returns a numeric value in $[0, 1]$ indicating the confidence of the prediction for ρ to be extended to a full match. We set a configurable threshold δ_T such that $\mathcal{M}(G, Q, \rho) \geq \delta_\mathsf{T}$ if the prediction of $\mathcal{M}$ is highly confident to be true, *i.e.,* ρ is valid. Departing from procedure IsFeasible in Enumerate that only inspects the 2-hop neighbors of pattern vertices, $\mathcal{M}$ learns topological connections of graphs and patterns, and *directly* predicts whether ρ is valid or not.

Intuitively, when the ML model $\mathcal{M}$ is highly accurate, one can use $\mathcal{M}$ instead of IsFeasible. After expanding ρ with (u, v), $\mathsf{VF3}_M$ first calls $\mathcal{M}$ to predict whether $\rho + (u, v)$ can be extended to a full match; if $\mathcal{M}$ returns true, then it extends ρ with (u, v) just like VF3; otherwise it does not extend ρ with (u, v), without backtracking.

Example 3: Consider graph G and pattern Q in Figure 1. There exists only one match of Q in G, *i.e.,* the leftmost part of G. Note that the n cycles with two vertices in G cannot match the cycle with four vertices in Q. Such mismatches cannot be filtered out by graded simulation [80] in node classification described earlier.

One can verify the following. (1) When VF3 is invoked to compute $Q(G)$, procedure Enumerate is called $4n + 11$ times, one for each vertex in G, plus the first call by VF3. (2) In contrast, if we replace IsFeasible with $\mathcal{M}$ and if $\mathcal{M}$ predictions are accurate, we only need to invoke Enumerate 8 times, which finds the unique match of Q in G. Indeed, when $\mathcal{M}$ is accurate, it correctly predicts that the partial match $\rho(u_2) = v_i$ ($i \in [2, n+1]$) cannot lead to a full match, and thus does not call Enumerate for further inspection. □

Template $\mathsf{VF3}_M$. However, when $\mathcal{M}$ is not perfectly accurate, it may incorrectly predict false on a true match and miss the match (false negative, FN). It may also predict true on false matches (false positive, FP) and incurs redundant checking. To reduce the impact of FNs and FPs, we develop $\mathsf{VF3}_M$ by including two procedures.

Template $\mathsf{VF3}_M$ is a mild variation of VF3 with the following. (1) It replaces IsFeasible (line 8 of Figure 2) with $\mathsf{IsFeasible}_\mathsf{M}$ given in Figure 3. It employs $\mathcal{M}$ to predict whether ρ is valid after expanding ρ with (u, v). (2) To reduce FPs of $\mathcal{M}$ predictions, when $\mathcal{M}$ returns true, it may call Boolean function ReduceFP to conduct further checking (line 2). (3) To reduce FNs of $\mathcal{M}$, when $\mathcal{M}$ predicts false, it may call Boolean function ReduceFN to mitigate bad predictions (line 3). ReduceFP and ReduceFN serve as *backup plans* for bad predictions of $\mathcal{M}$, and will be elaborated in Section 3.3.

Input: G, Q, ρ, $u \in Q$, and $v \in G$ as in Figure 3.
Output: true if ρ is valid.
1. $\mathsf{VF3\_check} := \mathsf{IsFeasible}(G, Q, \rho, u_n, v_n)$;
2. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) \geq \delta_\mathsf{T}$ and VF3_check **then**
3.   **return** true;
4. **if** $\mathcal{M}(G, Q, \rho + (u_n, v_n)) < \delta_F$ **then**
5.   **return** false;
6. **return** VF3_check;

Figure 4: Procedure $\mathsf{IsFeasible}_\mathsf{M}$ **in** $\mathsf{VF3}^O_M$

3.3 Three $\mathsf{VF3}_M$ Algorithms

Based on the template, we next develop three $\mathsf{VF3}_M$ algorithms with different strategies for reducing FPs and FNs of $\mathcal{M}$ predictions.

(1) Do nothing. The first $\mathsf{VF3}_M$ algorithm, denoted by $\mathsf{VF3}^N_M$, opts to completely trust the ML model $\mathcal{M}$. It follows the predictions of $\mathcal{M}$. Here ReduceFP (resp. ReduceFN) simply returns true (resp. false) when $\mathcal{M}$ predicts true (resp. false). In other words, $\mathsf{VF3}^N_M$ simply replaces IsFeasible in line 8 of Figure 2 with $\mathcal{M}(G, Q, \rho) \geq \delta_\mathsf{T}$ for a predefined threshold δ_T, indicating that $\mathcal{M}$ predicts true; the rest of the VF3 code of Figure 2 remains unchanged.

As will be seen in Section 4, $\mathsf{VF3}^N_M$ is (a) consistent even when $\mathcal{M}$ is not accurate, and (b) output-polynomial with a high probability. However, it may miss true matches when $\mathcal{M}$ makes FN predictions, *i.e.,* the recall of $\mathsf{VF3}^N_M$ is impacted by bad $\mathcal{M}$ predictions, where recall is the ratio of matches returned by $\mathsf{VF3}^N_M$ to all matches in $Q(G)$.

(2) VF3 as a backup. We develop $\mathsf{VF3}^O_M$ to improve the recall of $\mathsf{VF3}^N_M$. We set another configurable threshold δ_F such that $\mathcal{M}(G, Q, \rho) < \delta_\mathsf{F}$ if the prediction of $\mathcal{M}$ is highly confident to be false, *i.e.,* ρ cannot be extended to a full match. Note that $\delta_\mathsf{F} < \delta_\mathsf{T}$.

Algorithm $\mathsf{VF3}^O_M$ replaces IsFeasible of VF3 with Boolean function $\mathsf{IsFeasible}_M$ shown in Figure 4. It makes decisions as follows. (a) It returns true if both $\mathcal{M}$ predicts true (*i.e.,* the confidence of $\mathcal{M}$ is at least δ_T) and IsFeasible returns true, to reduce FPs (line 2-3). Here IsFeasible serves as ReduceFP. (b) If $\mathcal{M}(G, Q, \rho) < \delta_\mathsf{F}$, *i.e.,* if $\mathcal{M}$ is highly confident to predict false, it follows the prediction and returns false (lines 4-5). (c) Otherwise, *i.e.,* when $\mathcal{M}$ is confident enough to predict neither true nor false, it resorts to IsFeasible to check the two-hop neighbors of u and v as in VF3. That is, IsFeasible is invoked as ReduceFN to improve the recall (line 6).

We will see that the recall of $\mathsf{VF3}^O_M$ is 1 with a high probability. Moreover, it is 1-robust with a high probability, and guarantees to be consistent. As a price to pay, it is unlikely output-polynomial.

(3) Deep checking. When $\mathcal{M}$ makes FN predictions, $\mathsf{VF3}^O_M$ may miss valid partial matches and hamper its recall. To reduce FNs, we develop $\mathsf{VF3}^D_M$ that replaces IsFeasible with $\mathsf{IsFeasible}_M$ of Figure 5. If $\mathcal{M}$ predicts false, it conducts deep checking (line 2) as ReduceFN.

Deep checking works as follows. Given partial match ρ, let $u'_1, \ldots, u'_{k_\rho}$ be the remaining vertices of pattern Q to be matched, in the matching order O. Here $k_\rho = |V_Q| - |\rho|$ and $|\rho|$ is the number of pattern vertices in ρ. Then deep checking first samples a vertex

Input: $G, Q, \rho, u \in Q$, and $v \in G$ as in Figure 3.
Output: true if ρ is valid.

VF3_check := IsFeasible(G, Q, ρ, u_n, v_n);
if $\mathcal{M}(G, Q, \rho + (u_n, v_n)) \geq \delta_T$ and VF3_check **then**
    return true;
if $\mathcal{M}(G, Q, \rho + (u_n, v_n)) < \delta_F$ **then**
    return DeepCheck(G, Q, ρ, u_n, v_n);
return VF3_check;

Figure 5: Procedure IsFeasible$_M$ in VF3^D_M

Input: $G, Q, \rho, u \in Q$, and $v \in G$ as in Figure 3.
Output: true if ρ is valid.

1. let $u'_1, \ldots, u'_{k_\rho}$ be the remaining pattern vertices to be matched; $j := 0$;
2. **for each** $j \leq S(k_\rho)$ **then**
3.   randomly sample one vertex v_i in $C(u'_i)$ for each $i \in [1, k_\rho]$;
4. $\rho_c := \rho + (u'_1, v_1) + \cdots + (u'_{k_\rho}, v_{k_\rho})$; $j := j + 1$;
5.   **if** ρ_c is a full match **then**
6.   **return** true;
7. **return** false;

Figure 6: Procedure DeepCheck in VF3^D_M

v_1 from candidate set $C(u'_1)$ such that v_1 and u'_1 have "isomorphic" connections with vertices in ρ, and then expands ρ with (u'_1, v_1). It repeats the step to sample one vertex v_i for each u'_i ($i \in [2, k_\rho]$) following O, and generates $\rho_c = \rho + (u'_1, v_1) + \cdots + (u'_{k_\rho}, v_{k_\rho})$. It checks whether ρ_c makes a full match and returns true if so.

To reduce FNs, deep checking is conducted for $S(k_\rho)$ times, where $S(k_\rho) = |G|^{k_\rho/|Q|} \leq |G|$, following a sampling rate decay strategy [94]. Intuitively, the more remaining pattern vertices are (*i.e.,* the larger k_ρ is), the more deep checking steps are performed. This strategy is to strike a balance between the complexity and accuracy. Other strategies can be used instead to reduce FNs.

As shown in Figure 6, DeepCheck generates at most $S(k_\rho)$ extensions of ρ (line 2). Each extension samples one vertex v_i in $C(u'_i)$ for each $i \in [1, k_\rho]$ (line 3), and forms $\rho_c = \rho + (u'_1, v_1), \ldots, (u'_{k_\rho}, v_{k_\rho})$ (line 4). DeepCheck checks whether ρ_c is valid; if so, it returns true. It returns false if none of the $S(k_\rho)$ extensions finds a full match.

Example 4: Consider graph G, pattern Q and matching order O (*i.e.,* tree $\mathcal{T}$) in Figure 1. VF3^D_M first selects candidates for each pattern vertex in Q via graded simulation. It then enumerates matches in $Q(G)$ following O. Unlike VF3, upon mapping u_2 to v_2, VF3^D_M calls $\mathcal{M}$ to check whether the partial match ρ is valid. If so, it expands ρ by mapping u_3 to t^1_2 in G; otherwise, it calls DeepCheck to map $u_3, \ldots, u_7$ to vertices along the path from t^1_2 in G. If DeepCheck returns false and $\mathcal{S}(1) = 1$, it backtracks without mapping u_3 to another child t^2_2 of v_2, and inspects other candidates of u_2. □

4 Performance Guarantees

This section studies the efficiency and accuracy of VF3$_M$ algorithms. We report negative (Section 4.1) and positive (Section 4.2) results.

We first present the measures for evaluating these two aspects.

Efficiency. We adopt the following two measures for efficiency.

Output polynomial. As remarked in Section 2, we say that a VF3$_M$ algorithm is *output-polynomial* for SubIso if its time cost can be expressed as a polynomial in the size of input (graph G and pattern Q) and the size of output (the set of matches $|Q(G)|$).

Competitive ratio. Following [70], we define the *competitive ratio* λ of a VF3$_M$ algorithm *w.r.t.* a yardstick optimal algorithm such that

$$\text{cost}(Q, G) \leq \lambda \ \text{OPT}(Q, G),$$

where cost(Q, G) denotes the cost of VF3$_M$ for computing $Q(G)$, and OPT is the cost incurred by the optimal offline algorithm. The smaller the ratio λ is ($\lambda \geq 1$), the more efficient the algorithm is.

Accuracy. Consider a VF3$_M$ algorithm $\mathcal{A}$. We measure the accuracy of $\mathcal{A}$ in terms of F1 measure: $F1(\mathcal{A}) = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision}+\text{recall})}$. Here the *precision* of $\mathcal{A}$ is the ratio of true matches found by $\mathcal{A}$ to all matches returned by $\mathcal{A}$, *i.e., precision* = $\frac{|\text{Output} \cap Q(G)|}{|\text{Output}|}$, and the *recall* is the ratio of true matches returned by $\mathcal{A}$ to all true matches in $Q(G)$, *i.e., recall* = $\frac{|\text{Output} \cap Q(G)|}{|Q(G)|}$, for patterns Q and graphs G.

To evaluate the impact of the embedded ML model $\mathcal{M}$ on the accuracy of $\mathcal{A}$, we also study the following two measures.

Consistency. Denote the error rate of model $\mathcal{M}$ by η, which is the probability that $\mathcal{M}$ makes wrong predictions. A VF3$_M$ algorithm $\mathcal{A}$ with $\mathcal{M}$ is *consistent* if $F1(\mathcal{A}) = 1$ when $\eta = 0$, *i.e.,* when $\mathcal{M}$ tends to be error-free, $\mathcal{A}$ finds all and only the matches in $Q(G)$.

Robustness. A VF3$_M$ algorithm $\mathcal{A}$ with ML model $\mathcal{M}$ is β*-robust* if $F1(\mathcal{A}) \geq \beta$ for a bound β regardless of η; *i.e.,* it is still able to find true matches even when $\mathcal{M}$ makes arbitrarily bad predictions.

4.1 Negative Results

We start with an impossibility result.

Theorem 2: *No algorithm exists for* SubIso *that is both output polynomial and β-robust for any positive constant β unless* fewP = P. □

Here fewP is the class of problems that can be recognized by nondeterministic Turing machines T in polynomial time and have polynomially-bounded number of solutions [9]. It is known that P ⊆ fewP [53, 81], and fewP = P is likely improbable as P = NP [36].

Proof sketch: We prove that if there exists an output-polynomial and β-robust algorithm $\mathcal{A}$ for SubIso, where β is a positive constant, then fewP = P. Assume by contradiction that such an algorithm $\mathcal{A}$ exists with $F1(\mathcal{A}) \geq \beta$; we show that fewP ⊆ P. (1) We first construct a parsimonious reduction from fewP to a subset of SubIso, denoted by SubIso$^{\text{P}}$, which is the set of SubIso instances $C = (Q, G)$ such that $Q(G)$ consists of polynomially many matches of Q in G [33, 79]. (2) Using algorithm $\mathcal{A}$, we develop a PTIME algorithm $\mathcal{B}$ to check whether T accepts x as follows: run $\mathcal{A}$ to compute matches $Q(G)$ of Q in G; if $\mathcal{A}$ returns at least one match, then return true, *i.e., T* accepts x; otherwise, return false. We can readily verify the correctness of the reduction and that algorithm $\mathcal{B}$ is in PTIME. Thus fewP ⊆ P and hence, fewP = P by P ⊆ fewP [53, 81]. In other words, algorithm $\mathcal{A}$ does not exist unless fewP = P. □

4.2 Positive Results

Despite Theorem 2, we show that VF3$_M$ algorithms VF3^N_M, VF3^O_M and VF3^D_M have certain performance guarantees.

4.2.1 Accuracy. We start with precision and consistency.

Consistency. All the three VF3$_M$ algorithms are consistent.

Theorem 3: *Algorithms* VF3^N_M, VF3^O_M *and* VF3^D_M *always (1) have* precision *= 1 and (2) are consistent.* □

Proof sketch: (1) For precision, observe that every match returned by the three VF3$_M$ algorithms is in $Q(G)$, since it is returned only when it is full and valid (see lines 1-2 of procedure Enumerate).

(2) When the error rate η of $\mathcal{M}$ is 0, we show that the F1 measure of the three algorithms is 1. It suffices to show that the recall of $\mathsf{VF3}_M^N$ is 1, since the precision of all three $\mathsf{VF3}_M$ algorithms is 1, and the recalls of $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are not smaller than that of $\mathsf{VF3}_M^N$.

For $\mathsf{VF3}_M^N$, we show that for each full match $\rho \in Q(G)$, $\mathsf{VF3}_M^N$ returns ρ when $\eta = 0$. Let $\rho_1, \ldots, \rho_{|Q|}$ be all the partial matches of ρ constructed following the matching order O. Since $\mathsf{VF3}_M^N$ follows the predictions of $\mathcal{M}$, and $\mathcal{M}$ returns true for each partial match ρ_i when $\eta = 0$, $\mathsf{VF3}_M^N$ extends ρ_i to ρ_{i+1} for all $i \in [1, |Q| - 1]$. Since $\rho_Q = \rho$ is full, $\mathsf{VF3}_M^N$ returns ρ at line 2 of procedure Enumerate. □

Robustbess. We have a lower bound for the robustness of $\mathsf{VF3}_M^N$ using its output. Here we assume *w.l.o.g.* that the prediction of $\mathcal{M}$ is "monotonically decreasing" (see Section 5), *i.e.,* if a partial match ρ is extended from partial one ρ', when $\mathcal{M}(G, Q, \rho') =$ false, the prediction $\mathcal{M}(G, Q, \rho)$ must also be false. Intuitively, since ρ is extended from ρ', the search space for full matches of ρ' is larger, and when ρ' cannot be extended to a full match, neither can ρ.

Theorem 4: *Given graph G and pattern Q, for constant $\varepsilon \in (0, 1)$, $\mathsf{VF3}_M^N$ is $\frac{2\mathsf{Output}_T}{2\mathsf{Output}_T + f(\varepsilon)}$-robust with a probability of at least $1 - \varepsilon$.* □

Here (a) Output_T denotes the number of full matches returned by $\mathsf{VF3}_M^N$; and (b) $f(\varepsilon) = N_N\mathcal{B}$, where N_N is the number of false predictions of $\mathcal{M}$, and $\mathcal{B} = \frac{1+\eta|G|-|G|+\sqrt{(|G|-1-|G|\eta)^2-4(1-\frac{\varepsilon}{N_N})}}{2\eta}$.

Proof sketch: Since the precision of $\mathsf{VF3}_M^N$ is 1, it suffices to show that the recall of $\mathsf{VF3}_M^N$ is at least $\frac{\mathsf{Output}_T}{\mathsf{Output}_T + N_N\mathcal{B}}$. Let $\mathcal{R} = \mathsf{Output}_T +$ Miss, where Miss is the number of matches missed by $\mathsf{VF3}_M^N$. Since recall can be rephrased as $\frac{\mathsf{Output}_T}{\mathsf{Output}_T + \mathsf{Miss}}$, we prove an upper bound for Miss. We show that the number of missed matches for each false prediction of $\mathcal{M}$ is bounded by $\mathcal{B}$ with a probability of at least $1 - \frac{\varepsilon}{N_N}$. Then, by using the generalized Bonferroni inequality [30], we show that the recall of $\mathsf{VF3}_M^N$ is at least $\frac{\mathsf{Output}_T}{\mathsf{Output}_T + N_N\mathcal{B}}$, *i.e.,* $\mathsf{VF3}_M^N$ is $\frac{\mathsf{Output}_T}{\mathsf{Output}_T + N_N\mathcal{B}}$-robust, with a probability of at least $1 - \varepsilon$. □

We next show that $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are "almost" 1-robust. As commonly found in practice, $\mathcal{M}$ makes wrong predictions typically when $\mathcal{M}(G, Q, \rho)$ falls in the range $[\delta_F, \delta_T)$, *i.e.,* when $\mathcal{M}$ is neither confident to predict true nor certain to predict false. To evaluate the impact of range $[\delta_F, \delta_T)$ on $\mathsf{VF3}_M$, denote by p_u the ratio of full matches at which the confidences of $\mathcal{M}(G, Q, \rho)$ are in the range $[\delta_F, \delta_T)$ to all full matches missed by $\mathsf{VF3}_M$. Denote by Output_T (resp. Output_{FT} and Output_F) the number of returned matches at which $\mathcal{M}(G, Q, \rho)$ is in the range $[\delta_T, 1]$ (resp. $[\delta_F, \delta_T)$ and $[0, \delta_F)$).

Theorem 5: *With a probability of at least $1 - \varepsilon$, algorithms $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are $\frac{2(\mathsf{Output}_T + \mathsf{Output}_{FT})}{2\mathsf{Output}_T + 2\mathsf{Output}_{FT} + (1-p_u)N_N\mathcal{B}}$-robust and $\frac{2(\mathsf{Output}_T + \mathsf{Output}_{FT} + \mathsf{Output}_F)}{2\mathsf{Output}_T + 2\mathsf{Output}_{FT} + 2\mathsf{Output}_F + (1-p_u)N_N\mathcal{B}}$-robust, respectively.* □

Observe that when the ratio p_u approximates 1, *i.e.,* when full matches missed by $\mathsf{VF3}_M$ fall in $[\delta_F, \delta_T)$, the recalls of $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ approach 1 and both algorithms are almost 1-robust.

Proof sketch: We first show that the recall of $\mathsf{VF3}_M^O$ is at least $\frac{\mathsf{Output}_T + \mathsf{Output}_{FT}}{\mathsf{Output}_T + \mathsf{Output}_{FT} + (1-p_u)N_N\mathcal{B}}$. Observe that $\mathsf{VF3}_M^N$ misses at most $N_N\mathcal{B}$ full matches with a probability of $1 - \varepsilon$, as shown in Theorem 4. $\mathsf{VF3}_M^O$ improves the recalls by further returning the full matches when the confidences of $\mathcal{M}$ are in the range $[\delta_F, \delta_T)$. Moreover, there exist at most $(1 - p_u)N_N\mathcal{B}$ full matches at which the confidences of $\mathcal{M}$ are below δ_F. Thus the recall of $\mathsf{VF3}_M^O$ is at least $\frac{\mathsf{Output}_T + \mathsf{Output}_{FT}}{\mathsf{Output}_T + \mathsf{Output}_{FT} + (1-p_u)N_N\mathcal{B}}$. The analysis of $\mathsf{VF3}_M^D$ is similar. □

4.2.2 Efficiency. Below we first study when the $\mathsf{VF3}_M$ algorithms are output polynomial. We then investigate their competitive ratios.

Output polynomial. $\mathsf{VF3}_M^N$ is "almost" output polynomial.

Theorem 6: *Given graph G and graph pattern Q, $\mathsf{VF3}_M^N$ is output polynomial with a high probability. For a constant $\varepsilon \in (0, 1)$, $\mathsf{VF3}_M^N$ runs in $|Q||G|(\mathsf{Output}_T + 1) \times C$ time with a probability of at least $1 - \varepsilon$, where $C = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\varepsilon)}}{2(1-\eta)}$.* □

Proof sketch: We show the following: (1) when $\mathcal{M}$ is error-free, $\mathsf{VF3}_M^N$ is output polynomial; and (2) when $\mathcal{M}$ is not very accurate, $\mathsf{VF3}_M^N$ is output polynomial with a probability of at least $1 - \varepsilon$.

(1) When $\mathcal{M}$ is error-free, it suffices to show that Enumerate is called at most $O(|G|(\mathsf{Output}_T + 1))$ times. For if it holds, $\mathsf{VF3}_M^N$ runs in $O(|Q||G|(\mathsf{Output}_T + 1))$ time, *i.e.,* $\mathsf{VF3}_M^N$ is output polynomial. Indeed, consider the following two cases. (a) When $\mathcal{M}$ returns true for a partial match ρ, ρ is valid, and the total number of true predictions is bounded by $O(|Q|\mathsf{Output}_T)$. (b) When $\mathcal{M}$ returns false for ρ, $\mathsf{IsFeasible}_M$ directly returns false, which is in $O(1)$ time.

(2) We next show that when the error rate of $\mathcal{M}$ is η, $\mathsf{VF3}_M^N$ is output polynomial with a high probability. Consider the following four cases of $\mathcal{M}$ predictions: (a) true negatives (TN), (b) FN, (c) true positives (TT), and (d) FP. Cases (a)-(c) can be analyzed as in (1). For case (d), denote by $\mathcal{S}$ the set of minimal partial matches $\rho_{\min}$ such that (i) $\mathcal{M}$ returns false on $\rho_{\min}$, and (ii) $\mathcal{M}$ predicts true on all partial matches from which $\rho_{\min}$ is extended. We show that with a probability of at least $1 - \frac{\varepsilon}{|\mathcal{S}|}$, for any minimal partial match ρ'_k ($k \in [1, |\mathcal{S}|]$), there exist at most $C' = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\frac{\varepsilon}{|\mathcal{S}|})}}{2(1-\eta)}$ false-positive (FP) predictions when $\mathcal{M}$ checks partial matches extended from ρ'_k. Then we use the generalized Bonferroni inequality [30] to show that the number of FP predictions is bounded by $|G|(\mathsf{Output}_T + 1) \times C$ with a probability of at least $1 - \varepsilon$. □

Remark. (1) By Theorem 2, it is beyond reach in practice to train an error-free ML model $\mathcal{M}$ for SubIso, since otherwise, $\mathsf{VF3}_M^N$ is both 1-robust and output polynomial. This said, more accurate ML models for SubIso yield a higher recall for algorithm $\mathsf{VF3}_M^N$. We will train an ML model for SubIso with high accuracy in Section 5.

(2) In contrast, when $p_u = 1$, *i.e.,* when all those full matches missed by $\mathcal{M}$ fall in the confidence range $[\delta_F, \delta_T)$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ call IsFeasible as VF3, and they behave the same as VF3. Hence unless $\mathcal{M}$ is accurate, they cannot be output-polynomial by Theorem 2.

Competitive ratio. We start with algorithm $\mathsf{VF3}_M^N$.

Corollary 7: *Given graph G and pattern Q, for a constant $\varepsilon \in (0,1)$, the competitive ratio of* $\mathsf{VF3}_M^N$ *is* $2|G|Cf(Q,\mathcal{M})$ *with a probability of at least* $1-\varepsilon$*, where* $C = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\varepsilon)}}{2(1-\eta)}$*, and* $f(Q,\mathcal{M})$ *denotes the cost for* $\mathcal{M}$ *to make a prediction.* □

Proof: The cost of $\mathsf{VF3}_M^N$ is bounded by $|G|(\mathsf{Output}_T+1)\times C\times f(Q,M)$ with a high probability by Theorem 6. Since the precision of $\mathsf{VF3}_M^N$ is 1, there exist at least Output_T many full matches in $Q(G)$, and hence $\mathsf{Output}_T \le \mathsf{OPT}$, *i.e.,* the cost incurred by the optimal offline algorithm is at least Output_T. Then the cost of $\mathsf{VF3}_M^N$ is at most $|G|(\mathsf{Output}_T+1)\times C\times f(Q,\mathcal{M}) \le 2|G|Cf(Q,\mathcal{M})\cdot\mathsf{OPT}$. In practice, $f(Q,\mathcal{M})$ is often a polynomial after $\mathcal{M}$ is trained. □

We next study the competitive ratio of $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$. Recall p_{u} given earlier. Let $N_c = |G||C||Q|p_{\mathsf{u}}\frac{N_N}{\mathsf{Output}_T}\mathcal{B}$.

Proposition 8: *The competitive ratio of* $\mathsf{VF3}_M^O$ *(resp.* $\mathsf{VF3}_M^D$*) is* $5N_c(f(Q,\mathcal{M})+|G||Q|)$ *(resp.* $5N_c(f(Q,\mathcal{M})+2|G|^2|Q|)$*).* □

Proof sketch: We prove the competitive ratio for $\mathsf{VF3}_M^O$. The analysis of $\mathsf{VF3}_M^D$ is similar and hence omitted. We partition the partial matches into three parts based on the predictions of $\mathcal{M}$: when $\mathcal{M}$ has confidence (a) above δ_{T}; (b) between δ_{T} and δ_{F}; and (c) below δ_{F}. We prove the competitive ratio for each of these cases. □

5 Embedding and Learning

This section trains an ML model $\mathcal{M}$ for $\mathsf{VF3}_M$. Given a pattern Q, a graph G and a partial match ρ of Q in G, model $\mathcal{M}$ predicts whether ρ is valid, *i.e.,* whether it can be extended to a full match of Q in G.

Challenges. It is nontrivial to train $\mathcal{M}$. Model $\mathcal{M}$ should be (a) efficient since it is possibly invoked exponentially many times by $\mathsf{VF3}_M$ to make predictions, and (b) accurate to ensure the recall of $\mathsf{VF3}_M$. Moreover, it (c) has to encode both edge labels and directions to retain the topology of Q and G, and (d) should be monotonically decreasing, to ensure the robustness of $\mathsf{VF3}_M$ (see Section 4).

One might want to use existing models for subgraph matching or counting, *e.g.,* D2Match [69], SKETCH [113] and DMPNN [68]. However, these models do not work here since they (1) do not take partial match ρ of a pattern Q as input, and (2) even when we extend these models to predict ρ, they check only unmatched part of Q without considering what is already matched in ρ (see Section 6).

Overview. To tackle these challenges, we train $\mathcal{M}$ as shown in Figure 7. Given graph G, pattern Q and a partial match ρ (colored in red in G and Q), we (1) compute the embeddings of vertices in G and Q using IDGNN; (2) partition G and Q into multiple paths; (3) aggregate the embeddings of the paths by concatenating their vertex embeddings; and (4) make a prediction by comparing the embeddings of these paths in the order embedding space [105].

(1) We employ IDGNN [107] to extract topological information of G and Q. IDGNN extends vanilla GNNs by incorporating vertex identities into the message passing procedure, and works better than GNN in distinguishing non-isomorphic graphs. We further extend IDGNN to accommodate graphs with edge labels and directions.

(2) For efficiency, we pre-compute embeddings of vertices with IDGNN for graph G and pattern Q before the enumeration. We compute the vertex embedding of G (resp. Q) only once, and reuse them when comparing different patterns (resp. partial matches).

(3) We implement a partition-based strategy [104] to enhance the accuracy of $\mathcal{M}$. Intuitively, the paths catch topological details of G and Q, such as vertices being connected along the same path, and help $\mathcal{M}$ make accurate predictions. In contrast, existing models, *e.g.,* D2Match [69], SKETCH [113] and DMPNN [68], typically make a prediction solely based on the embeddings of the entire G and Q, and may overlook the structural insights provided by these paths.

(4) Given a partial match ρ, we assign different weights to vertices based on their inclusion in ρ, during path embedding computation. This helps $\mathcal{M}$ preserve the existing vertex mappings encoded by the partial match ρ. To make $\mathcal{M}$ monotonically decreasing, we increase the confidence score by a constant determined by ρ, and expand the training set with known invalid partial matches (see below).

5.1 Graph Embedding

We now present steps (1)–(4) one by one.

(1) Vertex embedding. Given a graph G and a vertex v, GNN computes an embedding of v via iterative aggregation of the embeddings of its neighbors. It gathers the embeddings of its neighbors, aggregates these embeddings via a message passing function MSG, and updates the embedding of v with an aggregation function AGG. These steps are repeated m times, for a predefined parameter m (*a.k.a.* the number of layers in GNN). However, such GNN may fail to distinguish non-isomorphic graphs. It is no more expressive than 1-dimensional Weisfeiler-Leman (1-WL) test [101].

To improve the accuracy of $\mathcal{M}$, we adopt IDGNN, which extends vanilla GNNs by using different MSG functions for different vertices. IDGNN is more expressive than 1-WL test and vanilla GNN [107], and can better distinguish non-isomorphic graphs. More specifically, it computes the embedding of each vertex v as follows: it (1) first collects a m-hop subgraph G_v centered at v, and (2) then iteratively computes the embedding of each vertex u in G_v by:

$$h_u^0 := x_u, \tag{1}$$

$$h_u^{l+1} := \mathsf{AGG}^l(\{\mathsf{MSG}^l_{\mathsf{I}(u_n=v)}(h_u^l), u_n \in N(u)\}, h_u^l). \tag{2}$$

Here (a) x_u is the initial feature of vertex u in G_v, *e.g.,* the embedding of its label; (b) AGG^l is the aggregation function, which updates the embedding of u by combining its current embedding h_u^l with the information of its neighbors collected through the message passing function $\mathsf{MSG}^l_{\mathsf{I}(u_n=v)}$; (c) for each neighbor u_n of u, $\mathsf{MSG}^l_{\mathsf{I}(u_n=v)}(h_{u_n})$ constructs a message sent to u using the embedding $h_{u_n}^l$ of u_n; and (d) $\mathsf{I}(u_n=v)$ is an indicator function, returning 1 if u_n and v are the same vertex, and 0 otherwise.

Here $\mathsf{MSG}^l_{\mathsf{I}(u_n=v)}$ is not uniform across all vertices in G_v, and IDGNN utilizes distinct message passing functions for v (*i.e.,* MSG^l_1) and other vertices in G_v (*i.e.,* MSG^l_0). To accommodate edge labels and directions, IDGNN assigns distinct weights to different edge labels when constructing messages for its neighbors via MSG [87]; it groups and concatenates the embeddings of incoming and outgoing edges of u when updating the embedding of u [55]. The subgraph G_v can be computed efficiently by sampling neighbors [49], and is only used to calculate the embedding of v [107].

(2) Path partition. Given a graph G, we partition G into a set of

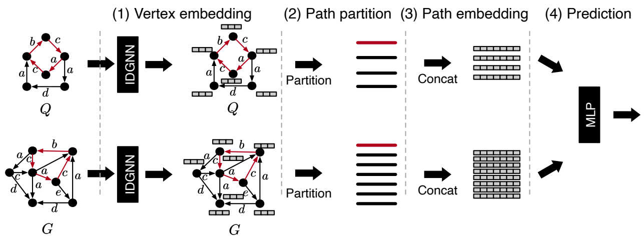


Figure 7: The framework of $\mathcal{M}$

directed paths, denoted by $\mathcal{S}_G$, such that each vertex is included in at least one path in $\mathcal{S}_G$. To reduce the cost of the partition, a degree-based strategy is employed, as shown in Figure 8. More specifically, (a) it first picks vertex v_0 with the maximum degree in G (line 1). (b) Then it randomly selects a set $\mathcal{S}_G$ of paths, each up to length l, that contain v_0 (line 2); here l is a predefined parameter to strike a balance between complexity and accuracy [104]. (c) It next iteratively enriches $\mathcal{S}_G$ with additional paths to encompass vertices that are not yet contained in any path in $\mathcal{S}_G$. This is carried out by procedure Expand (lines 3-4, not shown), which begins at an uncovered vertex v', and randomly chooses N_p paths following the direction of edges connected to v', to collect sufficient information of v' for the prediction. Here N_p is another predefined parameter to strike a balance between complexity and accuracy [49]. (d) It returns $\mathcal{S}_G$ once all vertices have been covered (line 5).

(3) Path embeddings. Given a graph G and the set $\mathcal{S}_G$ of partitioned paths of G, the embeddings of paths in $\mathcal{S}_G$ are computed as follows: for each path p in $\mathcal{S}_G$, its embedding is obtained by concatenating embeddings of the vertices along the path. The vertex embeddings in G have been computed using IDGNN in step (1), and can be reused for different paths. To retain the existing mappings between vertices encoded by the partial match ρ, distinct weights are assigned to vertices that are either part of ρ or not. More specifically, given a path p, its embedding is computed as:

$$e_p = \sum_{v \in \rho \cap p} c_1 e_v + \sum_{v \in \cap p \setminus \rho} c_2 e_v,$$

where (a) c_1 and c_2 are the weights for matched and unmatched vertices, respectively; (b) e_v is the embedding of v; (c) $\rho \cap p$ consists of vertices appearing in both partial match ρ and path p, and (d) $p \setminus \rho$ denotes the set of vertices that appear in p but not in ρ.

Let $\mathcal{S}^e_Q$ (resp. $\mathcal{S}^e_G$) be set of embeddings of paths in $\mathcal{S}_Q$ (resp. $\mathcal{S}_G$), which are extracted from pattern Q (resp. graph G) in step (2).

(4) Prediction. Given $\mathcal{S}^e_G$ and $\mathcal{S}^e_Q$, the task is to predict whether the given partial match ρ is valid. It directly compares the embeddings of paths in $\mathcal{S}_G$ and $\mathcal{S}_Q$, since these paths encapsulate the topological information of G and Q, respectively. We utilize the order embedding space [105] to compare the embeddings of paths in $\mathcal{S}_G$ and $\mathcal{S}_Q$. Given paths $p_q \in \mathcal{S}_Q$ and $p_g \in \mathcal{S}_G$, let their embeddings be $e_q = (a_1, \ldots, a_d)$ and $e_g = (b_1, \ldots, b_d)$, respectively, where d is the dimension of the embeddings. The order-embedding space guarantees that if p_q is a subpath of p_g, then $a_i \leq b_i$ for all $i \in [1, d]$. Moreover, when Q is connected, the picked paths in G must be connected too. Hence, if every path in $\mathcal{S}_Q$ is a subpath of some path in $\mathcal{S}_G$, and all picked paths in G are connected as in Q, $\mathcal{M}$ returns true, indicating that ρ can be extended to a full match.

To return the confidence for ρ to be valid (see Section 3.3), we use a multilayer perceptron (MLP) to output a number in $[0, 1]$. Since

Input: Graph G.
Output: A set $\mathcal{S}_G$ of paths containing all vertices in G.
1. pick a vertex v_0 with the maximum degree in G;
2. randomly pick a set $\mathcal{S}_G$ of paths containing v_0;
3. **while** a vertex v in G is not covered by $\mathcal{S}_G$ **do**
4. $\quad\mathcal{S}_G := \mathcal{S}_G \cup \mathsf{Expand}(G, \mathcal{S}_G, v)$;
5. **return** $\mathcal{S}_G$;

Figure 8: Path partition

MLP takes vectors as input, we encode the sets $\mathcal{S}^e_G$ and $\mathcal{S}^e_Q$ into vectors. Note that a partial match ρ is valid only when each path p_q in $\mathcal{S}_Q$ is a subpath of some path p_g in $\mathcal{S}_G$, *i.e.,* $a_i \leq b_i$ for all $i \in [1, d]$. Thus, (a) for each path p_q in $\mathcal{S}_Q$, we select a path $p_g \in \mathcal{S}_G$ such that p_q is a subpath of p_g. (b) Instead of insisting on that $a_i \leq b_i$ for all $i \in [1, d]$, we use a margin threshold for the difference between e_q and e_g, allowing $a_i > b_i$ for some $i \in [1, d]$, to improve the recall and robustness of $\mathsf{VF3}_M$ [31, 96, 106]. Moreover, (c) we employ the Chebyshev distance to quantify the distance from e_q to e_g. The Chebyshev distance, denoted by $d_c(e_q, e_g)$, is defined as $\max_{i \in [1,d]}(b_i - a_i)$, *i.e.,* the maximum difference between their coordinates [20].

Putting these together, we compute the embeddings of $\mathcal{S}^e_G$ and $\mathcal{S}^e_Q$ in two steps. (a) Construct a set $\mathcal{S}^e_{\text{path}}$ containing one pair $\langle e_q, e_g \rangle$ for each embedding e_q in $\mathcal{S}^e_Q$, where e_g is the embedding of a path in $\mathcal{S}^e_G$ that has the minimum Chebyshev distance to e_q, *i.e.,* $e_g = \arg\min_{e_1 \in \mathcal{S}^e_G} d_c(e_1, e_q)$; and (b) concatenate all pairs of embeddings in $\mathcal{S}^e_{\text{path}}$ to generate the embeddings of $\mathcal{S}^e_Q$ and $\mathcal{S}^e_G$, denoted as $e_Q = ||_{\langle e_q, e_g \rangle \in \mathcal{S}_{\text{path}}} e_q$ and $e_G = ||_{\langle e_q, e_g \rangle \in \mathcal{S}_{\text{path}}} e_g$, where $||$ is for concatenation. To illustrate, consider $\mathcal{S}^e_Q = \{(1, 2, 3), (4, 3, 1)\}$ and $\mathcal{S}^e_Q = \{(3, 1.8, 6), (1, 2, 4), (4, 2.8, 1), (3.9, 3, 1)\}$. Then $\mathcal{S}^e_{\text{path}} = \{\langle (1, 2, 3), (1, 2, 4) \rangle, \langle (4, 3, 1), (3.9, 3, 1) \rangle\}$, since $d_c((1, 2, 4), (1, 2, 3)) = 0$ and $d_c((3.9, 3, 1), (4, 3, 1)) = 0.1$ are the minimum Chebyshev distances from the embeddings $(1, 2, 3)$ and $(4, 3, 1)$ in $\mathcal{S}^e_Q$ to embeddings in $\mathcal{S}^e_G$, respectively. Thus, the embedding of $\mathcal{S}^e_Q$ (resp. $\mathcal{S}^e_G$) is $e_Q = (1, 2, 3, 4, 3, 1)$ (resp. $e_G = (1, 2, 4, 3.9, 3, 1)$).

To calculate the Chebyshev distances, we construct d sorted sets, one for each dimension of the embeddings. For each dimension i, the embeddings in sets $\mathcal{S}^e_G$ and $\mathcal{S}^e_Q$ are sorted by their values at index i. Given an embedding in $\mathcal{S}^e_Q$, we find the embedding in $\mathcal{S}^e_G$ with the minimum Chebyshev distance by inspecting these d sorted sets.

<u>*Complexity*</u>. We analyze the pre-computation and prediction times.

(1) The pre-computation, executed only once, takes $O(m(|Q| + |G|)(f_e(|G|) + f_e(|Q|)) + l(|G|d^G_{\max} + |Q|d^Q_{\max}))$ time, to (a) compute vertex embeddings for both G and Q, and (b) conduct path partitioning. Here f_e denotes the cost of updating embeddings via AGG and MSG, which is polynomial after $\mathcal{M}$ is trained, and l is the length bound for partitioned paths. Note that (i) the embedding process takes m rounds; (ii) for each round, given a vertex v, $\mathcal{M}$ takes $(|Q| + |G|)(f_e(|G|) + f_e(|Q|))$ time to update v's embedding as in Equation (2); and (iii) path partitioning takes at most $O(l(|G|d^G_{\max} + |Q|d^Q_{\max}))$ time, where $d^G_{\max}$ (resp. $d^Q_{\max}$) is the maximum degree of G (resp. Q).

(2) Given a graph G, a pattern Q and a partial match ρ, the prediction takes $O(d(P^\rho_{|Q|} + P^\rho_{|G|}) + f_m)$ time, where (a) $P^\rho_{|Q|}$ and $P^\rho_{|G|}$ are the numbers of paths in $\mathcal{S}_Q$ and $\mathcal{S}_G$ that include vertices of ρ within Q and G, respectively; (b) d is the dimension of embeddings; and f_m is the time required for prediction via MLP, which is polynomial

after $\mathcal{M}$ is trained. Observe that (i) only paths containing vertices in ρ are utilized for prediction, which improves the recall of $\mathcal{M}$, and the numbers $P^{\rho}_{|Q|}$ and $P^{\rho}_{|G|}$ of paths are small when l is small. (ii) Computing embeddings of paths takes $O(d(P^{\rho}_{|Q|} + P^{\rho}_{|G|}))$ time, due to the use of sorted sets and the locality of pattern matching (see Section 6). (iii) MLP outputs a confidence score in $O(f_m)$ time.

5.2 Training

We next outline the training process for model $\mathcal{M}$. Let D_T denote the training dataset, comprising tuples of the form (G, Q, ρ), where (a) G is a graph and Q is a pattern, and (b) ρ is a partial match of Q in G. Denote by D_T^+ (resp. D_T^-) the set of positive (resp negative) tuples (G, Q, ρ) in D_T such that ρ is valid (resp. invalid). We train model $\mathcal{M}$ by minimizing the following margin loss function [96]:

$$L(D_T) := \sum_{(G,Q,\rho)\in D_T^+} \sum_{p_q \in \mathcal{S}_Q} \min_{p_g \in \mathcal{S}_G} \|\max(0, d_c(e_g, e_q))\|_2^2$$
$$+ \sum_{(G,Q,\rho)\in D_T^-} \sum_{p_q \in \mathcal{S}_Q} \min_{p_g \in \mathcal{S}_G} \max(0, \alpha - \|\max(0, d_c(e_g, e_q))\|_2^2)$$

Here $\|\cdot\|_2$ denotes the L_2-norm, and α is a margin hyperparameter, to control the separation between positive and negative instances, thereby enhancing the generalization of the model [73, 93]. The model $\mathcal{M}$ is trained for N epochs, to improve its accuracy [58].

Remark. (1) For model $\mathcal{M}$ to be monotonically decreasing, we adopt the following two strategies. (1) We increase the confidence sore by $e^{-|\rho|}/2$, where $|\rho|$ is the number of pattern vertices in the partial match ρ. Intuitively, the larger $|\rho|$ is, the smaller $e^{-|\rho|}/2$ is, which would lead to smaller confidence. (2) We enrich the training set D_T with additional negative tuples derived from existing negative data in D_T^-. More specifically, for each negative tuple (G, Q, ρ) in D_T^-, where ρ is not valid, we add K new negative tuples (G, Q, ρ_1), ..., (G, Q, ρ_K) into D_T^-, where each partial match ρ_i $(i \in [1, K])$ is extended from ρ [74]. All partial matches $\rho_1, \ldots, \rho_K$ are invalid, *i.e.,* they cannot be extended to a full match, as ρ is invalid. Here $K = C_s(|Q| - |\rho|)$ for some predefined parameter C_s. Intuitively, we pick C_s vertices from candidate set $C(u)$ for each pattern vertex u that does not have a match in ρ, to extend the invalid match ρ.

(2) Training $\mathcal{M}$ is efficient, since it only learns the embedding model IDGNN and the prediction model. As will be seen in Section 6, $\mathcal{M}$ is fairly accurate with only 5000 epochs in 1.6h.

6 Experimental Study

Using real-life and synthetic datasets, we experimentally evaluated $\mathsf{VF3}_M$ algorithms for their (1) accuracy and (2) efficiency and scalability. We also evaluated (3) the performance of model $\mathcal{M}$.

Experimental setting. We start with the experimental setting.

Datasets. We used four real-life graphs: (1) DBLP [1], a citation network with 0.3M vertices and 1M edges; and (2) Pokec [4], an on-line social network in Slovakia with 1.6M vertices and 30M edges; (3) Patents [65], a U.S. patent citation graph (1975–1999) with 3.7M nodes and 15M edges; and (4) LiveJournal [3], a free on-line community graph with over 3M vertices and 60M edges.

To evaluate the scalability of algorithms, we generated synthetic graphs $G = (V, E, L)$ using datagen [54], controlled by the number $|V|$ of vertices (up to 4M) and the number $|E|$ of edges (up to 70M).

Method	DBLP		Patents		Pokec		LiveJournal	
	F1	Time (s)	F1	Time (s)	F1	Time (s)	F1	Time (s)
RM	1	7.72 (48.25×)	1	3.03 (2.50×)	1	33.28 (17.06×)	1	77.79 (27.59×)
CECI	1	3.88 (24.25×)	1	1.69 (1.39×)	1	14.62 (7.50×)	1	34.45 (12.21×)
DPiso	1	1.93 (12.06×)	1	1.36 (1.12×)	1	12.74 (6.53×)	1	21.60 (7.66×)
VF3	1	2.52 (15.75×)	1	4.14 (3.42×)	1	13.50 (6.92×)	1	21.21 (7.52×)
$\mathsf{VF3}_O$	1	0.17 (1.06×)	1	2.01 (1.66×)	1	2.56 (1.31×)	1	3.98 (1.41×)
GNNPE	1	0.22 (1.38×)	1	2.59 (2.14×)	1	2.24 (1.15×)	1	DNP
MLSearch	0.86	1066.55 (6665×)	*	Timeout	*	Timeout	*	Timeout
$\mathsf{VF3}_M^D$	0.99	0.16	0.98	1.21	0.98	1.95	0.97	2.82
$\mathsf{VF3}_M^N$	0.54	0.12 (↑ 25%)	0.23	1.13 (↑ 6.6%)	0.62	1.90 (↑ 2.5%)	0.38	2.20 (↑ 21.9%)
$\mathsf{VF3}_M^O$	0.99	0.13 (↑ 18.8%)	0.94	1.18 (↑ 2.4%)	0.93	1.95 (↑ 0%)	0.94	2.43 (↑ 13.8%)

Table 1: Accuracy and runtime; '*' indicates unavailable due to timeout (>6h), and DNP (Did Not Preprocess) denotes failure due to excessive preprocessing time (>6h)

Patterns. For each graph, we generated 100 graph patterns following [16, 112]. For each graph G, we first conducted random walks in G to identify vertices, and then extracted the subgraphs induced by these vertices. We treated these subgraphs as patterns.

Algorithms. We implemented in C++ (1) $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ (Section 3.3); and (2) a variant $\mathsf{VF3}_O$ of VF3 with optimizations in Section 3.1 but without using $\mathcal{M}$. We compared with six algorithms as baselines: (3) VF3 [5], the classic VF3 algorithm; (4) CECI [21], (5) RM [90], (6) DPiso [50]; these three work just like VF3, but use different matching orders, filtering strategies and index structures; (7) GNNPE [104], which applies pruning strategies guided by GNNs; and (8) MLSearch [102], which enhances VF3 by leveraging the embeddings of Q and G to predict whether a valid match exists.

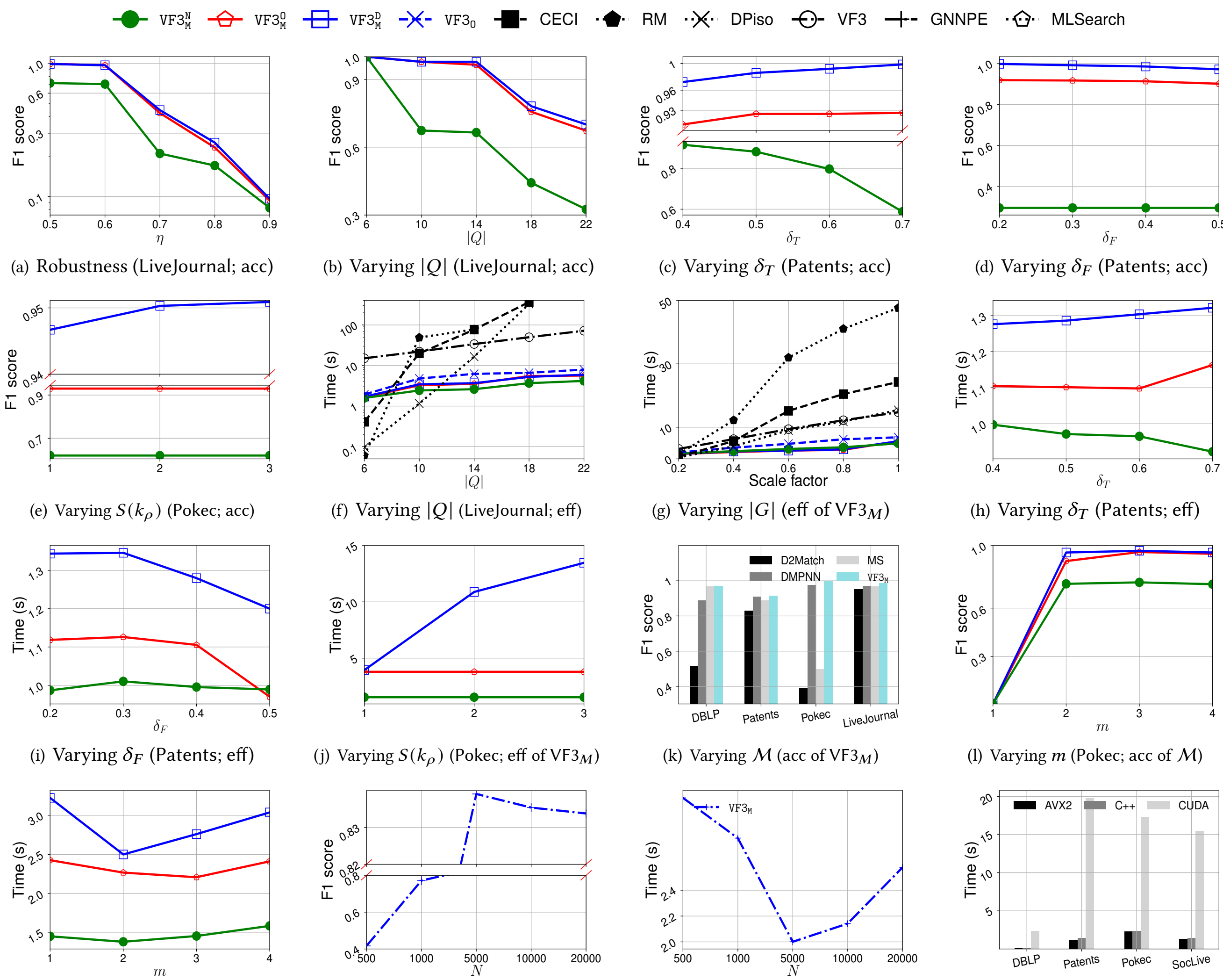
We compared our model $\mathcal{M}$ (Section 5) with the following: (9) D2Match [69], which employs deep GNNs and degeneracy for subgraph matching; (10) MS [102], which is the prediction model in algorithm MLSearch, and adopts graph attention network for subgraph matching; and (11) DMPNN [68], which develops dual message passing neural network for subgraph counting and matching. None of these models supports partial matches ρ. For example, given a pattern Q and a graph G, D2Match directly assesses the existence of a full match of Q in G. For a fair comparison, we extend these models as follows: given ρ, first generate pattern Q^r and graph G^r from Q and G by removing all vertices in ρ, respectively, and then apply these models on Q^r and G^r.

Environment. We ran experiments on a machine powered with 2 Intel Xeon Gold 6148 CPU @ 2.40GHz with 504 GB memory and 8 Tesla V100 GPU with 32 GB memory. For a fair comparison, we executed both algorithms for SubIso and model prediction on CPU, and only trained ML models on GPU. By default, we fixed $|Q| = |V_Q| + |E_Q| = 20$, $\delta_\mathsf{T} = 0.7$ and $\delta_\mathsf{F} = 0.5$ (confidence thresholds for $\mathcal{M}$ in $\mathsf{VF3}_M$ algorithms of Section 3.3). We ran each test 5 times, and report the average here. For the lack of space we report results on some datasets; the results on the others are consistent.

Experimental findings. We next report our findings.

Exp-1: Accuracy. We first report the F1 measure, consistency and robustness of $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$. Since CECI, RM, DPiso, GNNPE and VF3 return all full matches, their F1 scores are 1.

F1 score. As shown in Table 1, on average, (1) the F1 score of $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ is 0.44, 0.95 and 0.98, up to 0.62, 0.99 and 0.99, respectively. (2) $\mathsf{VF3}_M^D$ is 3% (resp. 54%) more accurate than $\mathsf{VF3}_M^O$



(a) Robustness (LiveJournal; acc)
(b) Varying $|Q|$ (LiveJournal; acc)
(c) Varying δ_T (Patents; acc)
(d) Varying δ_F (Patents; acc)
(e) Varying $S(k_\rho)$ (Pokec; acc)
(f) Varying $|Q|$ (LiveJournal; eff)
(g) Varying $|G|$ (eff of VF3$_M$)
(h) Varying δ_T (Patents; eff)
(i) Varying δ_F (Patents; eff)
(j) Varying $S(k_\rho)$ (Pokec; eff of VF3$_M$)
(k) Varying $\mathcal{M}$ (acc of VF3$_M$)
(l) Varying m (Pokec; acc of $\mathcal{M}$)
(m) Varying m (Pokec; eff of $\mathcal{M}$)
(n) Varying N (Patents; acc of $\mathcal{M}$)
(o) Varying N (Patents; eff of $\mathcal{M}$)
(p) Inference cost

Figure 9: Performance evaluation (acc for accuracy and eff for efficiency)

(resp. VF3^N_M), since it employs DeepCheck to reduce FNs.

Consistency. We tested F1(VF3$_M$) when the embedded ML model $\mathcal{M}$ is error free, *i.e.,* when $\eta = 0$ (Section 4). To this end, we stored in $Q(G)$ all full matches found by VF3, and determined whether a partial match ρ is valid by checking whether there exists a full match in $Q(G)$ extended from ρ, *i.e.,* by treating $Q(G)$ as an ideal model $\mathcal{M}$. We find that (a) all returned matches are valid, *i.e.,* the precision of these algorithms is 1, and (b) all matches in $Q(G)$ are returned by the three, *i.e.,* the recall of these algorithms is also 1. Thus VF3^N_M, VF3^O_M and VF3^D_M are consistent. These empirically confirm Theorem 3.

Robustness. We next tested the robustness. We used five models $\mathcal{M}$ with 0.5, 0.6, 0.7, 0.8 and 0.9 as error rate η. As shown in Figure 9(a): (1) with smaller η (*i.e.,* more accurate $\mathcal{M}$), all VF3$_M$ variants get more accurate and faster due to reduced redundancy (Theorem 6). (2) At $\eta = 0.5$, F1 scores of VF3^D_M, VF3^O_M and VF3^N_M are 0.99, 0.99 and 0.71, respectively; when $\eta = 0.6$, *i.e.,* when $\mathcal{M}$ performs worse than a coin toss, VF3^D_M and VF3^O_M still have F1 scores above 0.9. These are consistent with Theorems 4 and 5. (3) VF3^D_M is the most robust; even at $\eta = 0.7$, its F1 score is 0.45, outperforming VF3^O_M (0.43) and VF3^N_M (0.21), as DeepCheck improves recall.

The impact of parameters. We also evaluated the impact of $|Q|$, δ_T and δ_F on the F1 score of VF3^N_M, VF3^O_M and VF3^D_M.

(1) Varying $|Q|$. We varied the pattern size $|Q|$ from 6 to 22. As shown in Figure 9(b), (1) as $|Q|$ increases, the F1 scores of VF3^N_M, VF3^O_M and VF3^D_M generally decrease, since larger patterns make valid partial matches harder to identify and reduce ML prediction confidence. (2) VF3^D_M outperforms VF3^O_M and VF3^N_M by 5% and 76% in accuracy, respectively, as its use of DeepCheck reduces FNs.

(2) Varying δ_T. We evaluate the impact of confidence threshold δ_T. As shown in Figure 9(c), (1) when δ_T varies from 0.4 to 0.7, the F1 score of VF3^N_M decreases, since not all valid partial matches get high confidence when $\mathcal{M}$ is not error-free, and when δ_T gets larger, more valid matches may be missed. (2) VF3^O_M and VF3^D_M are less sensitive to δ_T since they invoke IsFeasible to reduce FPs, no matter whether their ML models are δ_T-confident or not.

(3) Varying δ_F. We varied the confidence threshold δ_F for false, as

shown in Figure 9(d): (1) as δ_F increases from 0.2 to 0.5, the F1 scores of $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ decrease, since $\mathcal{M}$ returns false on more partial matches. (2) $\mathsf{VF3}_M^D$ is more sensitive to δ_F than to δ_T, as larger δ_F may exclude more partial matches. (3) $\mathsf{VF3}_M^N$ remains unaffected by δ_F since it does not rely on this parameter (Section 3.3).

Varying $S(k_\rho)$. We varied the number of deep checking from $S(k_\rho)$ to $3S(k_\rho)$. As shown in Figure 9(e), (1) when $S(k_\rho)$ gets larger, $F1(\mathsf{VF3}_M^D)$ increases, as expected. (2) $F1(\mathsf{VF3}_M^D)$ becomes stable once the number reaches $2S(k_\rho)$, since ρ is invalid with high probability when no match is identified after $2S(k_\rho)$ many deep checking.

Exp-2: Efficiency and scalability. We next show that the $\mathsf{VF3}_M$ algorithms indeed speed up the enumeration process for SubIso. The results are consistent with Theorem 6, Corollary 7 and Proposition 8 for output-polynomial properties and competitive ratios.

Efficiency. As shown in Table 1, (1) $\mathsf{VF3}_M^D$ (in yellow) beats RM, CECI, DPiso, GNNPE, VF3 and $\mathsf{VF3}_O$ by 22.49×, 10.74×, 6.47×, 1.56×, 8.40× and 1.36×, respectively. MLSearch is 6,665× slower than $\mathsf{VF3}_M^D$ on DBLP and fails to terminate on other graphs after 6 hours. (2) $\mathsf{VF3}_M^D$ is only 14% and 8.8% slower than $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^N$ (marked as ↑), respectively, but is more accurate due to its deep checking. VF3 beats RM and CECI since (a) RM is join-based algorithm, which produces large intermediate results on dense graphs, and (b) CECI has a threshold for early termination, which we removed to return all matches, degrading the performance of CECI.

Varying $|Q|$. We varied $|Q|$ from 6 to 22. As shown in Figure 9(f), (1) $\mathsf{VF3}_M^D$ is faster than all the baselines; *e.g.,* when $|Q| = 14$, it is 21.15×, 20.86×, 4.49×, 9.21× and 1.69× faster than RM, CECI, DPiso, VF3 and $\mathsf{VF3}_O$, respectively; hence, ML models in $\mathsf{VF3}_M$ effectively reduce false positives and runtime. When $|Q| = 22$, $\mathsf{VF3}_M^D$ (resp. $\mathsf{VF3}_M^O$) are 1.34× (resp. 1.40×) faster than $\mathsf{VF3}_O$, the fastest baseline. (2) $\mathsf{VF3}_M^D$ is almost as efficient as $\mathsf{VF3}_M^O$, despite additional DeepCheck calls. Algorithm DPiso's runtime rises quickly when increasing $|Q|$, due to the overhead of building its failing sets to reduce the search space. GNNPE fails on LiveJournal with more than 60M edges.

Varying $|G|$. Fixing $|Q| = 22$, $\delta_T = 0.7$ and $\delta_\mathsf{F} = 0.5$, we varied the scale of synthetic graphs $G = (V, E)$ from 0.2 to 1.0. As shown in Figure 9(g): (1) as $|G|$ increases, all three $\mathsf{VF3}_M$ algorithms get slower, as expected. (2) $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ scale well; on graphs with 4M vertices and 70M edges, they take 4.89s, 5.58s and 5.59s, respectively, with F1 scores 0.36, 0.94, and 0.97. (3) The $\mathsf{VF3}_M^D$ algorithms are up to 13.61×, 6.78×, 3.90×, 4.04× and 2.05× faster than RM, CECI, DPiso, VF3 and $\mathsf{VF3}_O$ respectively. Results for GNNPE are omitted due to excessive preprocessing time (>6h). MLSearch takes more than 6h.

Varying δ_T. As shown in Figure 9(h), (1) on Patents, when δ_T increases from 0.4 to 0.7, the competitive ratio of $\mathsf{VF3}_M^N$ gets smaller. This is because $\mathsf{VF3}_M^N$ only returns valid partial matches with high confidence, and when δ_T increases, the number of such partial matches decreases. (2) In contrast, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are insensitive to δ_T, since they use IsFeasible as a backup regardless of δ_T.

Varying δ_F. As shown in Figure 9(i), (1) $\mathsf{VF3}_M^N$ is indifferent to δ_F as it does not use this parameter. (2) When δ_F increases from 0.2 to 0.5, both $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ get faster, as they inspect fewer candidate matches. (3) In this setting, $\mathsf{VF3}_M^D$ is on average 16% slower than $\mathsf{VF3}_M^O$, as with larger δ_F, $\mathsf{VF3}_M^D$ invokes DeepCheck more often.

Varying $S(k_\rho)$. We varied the number of deep checks from $S(k_\rho)$ to $3S(k_\rho)$. As shown in Figure 9(j), (1) as the number increases, $\mathsf{VF3}_M^D$ gets slower, as expected. (2) $\mathsf{VF3}_M^D$ takes 0.2s longer when the number varies from $S(k_\rho)$ to $2S(k_\rho)$, but its accuracy improves (see Fig. 9(e)), which justifies the design of $\mathsf{VF3}_M^D$ (see Section 3.3).

Exp-3: ML model. We evaluated the impact of model $\mathcal{M}$ (Section 5) on the accuracy of $\mathsf{VF3}_M$ and the cost to train $\mathcal{M}$. Unless stated otherwise, we fixed $m = 2$ and $N = 5000$.

Impact on $\mathsf{VF3}_M$ *algorithms*. We tested $\mathsf{VF3}_M$ variants that take D2Match, MS and DMPNN as ML model $\mathcal{M}$, respectively. As shown in Figure 9(k), on average $\mathsf{VF3}_M^N$ outperforms these variants by 0.61, 0.05 and 0.07 in F1-score, respectively. This verifies that model $\mathcal{M}$ of Section 5 is more accurate than D2Match, MS and DMPNN.

Varying m. We varied the number of IDGNN layers m from 1 to 4. As shown in Figure 9(l) and 9(m) on Pokec, the accuracy of $\mathsf{VF3}_M$ first improves and then remains stable; optimal performance is observed at $m = 2$, since (a) small m provides insufficient context; and (b) large m causes over-smoothing, making distant neighbors indistinguishable [101]. $\mathsf{VF3}_M$ also becomes the fastest when m=2.

Varying N. We tested the impact of the number (N) of training epochs. As shown in Figures 9(n) and 9(o), $\mathsf{VF3}_M$ achieves the highest accuracy when $N = 5000$. This is because (1) when N is too small, $\mathcal{M}$ is under-trained with near-random parameters; but (2) when N is too large, overfitting occurs, reducing generalization on unseen patterns. $\mathsf{VF3}_M$ becomes the fastest when $N = 5000$ by using a more accurate model to filter invalid candidates.

Training cost. We next report the training cost (not shown). (1) Training $\mathcal{M}$ is efficient; when m=2 and N=5000, it converges in 1.6h; and (2) training gets slower when m or N increases, as expected.

Prediction cost. We compared the impact of different implementations of $\mathcal{M}$ on $\mathsf{VF3}_M$. We implemented ML prediction using (1) AVX/AVX2 instructions, (2) standard C++ and (3) CUDA, after extracting model weights from $\mathcal{M}$. As shown in Figure 9(p), $\mathsf{VF3}_M$ implemented with AVX/AVX2 and standard C++ outperform the one with CUDA. This is because the model is lightweight and runs efficiently on CPU, while CUDA transfers data between CPU and GPU.

Summary. We find the following. *(1) Speedup*. Using ML oracles, $\mathsf{VF3}_M^D$ (resp. $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$) is 22.49×, 10.74×, 6.47×, 1.56× and 8.40× (resp. 29.97×, 14.29×, 8.45×, 1.73×, 10.35×; and 27.76×, 13.23×, 7.85×, 1.68×, 9.64×) faster than algorithms RM, CECI, DPiso, GNNPE and VF3, respectively, up to 48.25×, 24.25×, 12.06×, 2.14× and 15.75× (resp. 64.33×, 32.33×, 16.08×, 2.29×, 21.0×; and 59.38×, 29.84×, 14.84×, 2.19×, 19.38×). *(2) Accuracy*. The average F1 scores of $\mathsf{VF3}_M^N$, $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ are 0.44, 0.96 and 0.98, respectively, all with precision 1. Even when the error rate of $\mathcal{M}$ reaches 0.5, the F1 scores of $\mathsf{VF3}_M^O$ and $\mathsf{VF3}_M^D$ remain above 0.9. *(3) The accuracy of the ML model*. Our ML model $\mathcal{M}$ outperforms D2Match, MLSearch and DMPNN by 0.61, 0.05 and 0.07 in F1 score, respectively. *(4) Parameter setting*. Optimal performance is observed at $\delta_\mathsf{T} = 0.75$ and $\delta_\mathsf{F} = 0.35$, striking a balance between the accuracy and efficiency. To train model $\mathcal{M}$, N=5000 and m=2 work the best.

7 Related Work

We characterize the related work as follows.

Learning algorithms. There has been work on developing ML models for optimization problems. (1) Most models adopt deep reinforcement learning to incrementally compute solutions, by optimizing objective functions [61, 83], strengthening the ant colony optimization [103], and adopting a learning collaborative policy [63], the encoder-decoder structure [71], heuristic rules [109] or a force-directed method [75]. (2) Non-reinforcement models have also been studied to transform optimization problems to sampling problems [89, 91, 110], *e.g.,* [89] samples high quality solutions using Markov chain Monte Carlo; [91] develops graph-based denoising diffusion models to generate candidate solutions; and [110] models optimization problems as Markov decision processes, and computes the solutions by sampling with generative flow networks.

Closer to this work are [68, 69, 113]. Specifically, SKETCH [113] develops an active learning framework for subgraph counting; it trains GNN to predict counts, and samples queries to update the models. DMPNN [68] employs dual message passing neural networks for subgraph isomorphism counting. D2Match [69] leverages deep GNNs and degeneracy for subgraph matching, by transforming the subgraph matching to subtree matching.

This work differs from the prior work. (1) We unify algorithmic methods and ML models to benefit from both, as opposed to relying on ML models alone [61, 83]. (2) We study enumeration algorithms for an EnumP-complete problem, not for decision or optimization problems [75, 91]. (3) We target ML models for SubIso to scale with large graphs and patterns, rather than the one-size-fit-all model of [61, 68, 69, 113] that deal with small graphs. (4) As opposed to [68, 69, 113], our model checks whether a partial match is valid.

Algorithms with ML oracles. There has also been work on developing algorithms with ML predictions to improve the efficiency, categorized as follows. (1) Algorithms for online problems, *e.g.,* online steiner tree [100], online bin packing [13], online TSP [44], online assignment [95], contract scheduling [12], paging [14, 59], caching [70], frequency estimation [6], index structures [64], revenue-maximizing auctions [32, 77] and sorting [17]. (2) Principles for designing algorithms with predictions. [82] shows how to develop algorithms for the ski rental problem, and proves performance guarantees. [61] combines reinforcement learning and graph embedding to learn greedy heuristics for online algorithms with ML predictions. [10] proposes a regression-based strategy to learn ML models for algorithms for online search such as ski rental and bin packing, and provides a competitive ratio of upper bound for such algorithms. (3) Strategies to improve the performance of algorithms with prediction, *e.g.,* [11] redesigns ML models by incorporating optimization benchmarks in the loss functions; [15, 37, 48, 72] combine the predictions of multiple ML models; and [35, 39] minimize the cost of ML predictions when the input distribution is given.

Closer to this work are [97, 99, 102, 104]. RL-QVQ [97] generates high-quality matching order for subgraph enumeration by employing reinforcement learning. GNNPE [104] provides an enumeration algorithm for SubIso, which partitions graph patterns into multiple paths, and prunes candidates of these paths using their GNN-based embeddings. MLSearch [102] trains a GNN model to filter partial matches by ranking degrees and labels, and develops a sampling algorithm to alleviate network training collapse. OptMatch [99] uses partial embeddings to find the first matches.

This work departs from the previous work in the following. (1) We study enumeration algorithms, not (the existence of) a single solution for a decision or optimization algorithm. (2) We focus on an EnumP-complete problem, rather than tractable (*e.g.,* sorting [17]) or NP-complete problems (*e.g.,* TSP [44]). (3) We study the consistency and robustness to characterize the accuracy of $\mathsf{VF3}_M$, not just to measure the cost of algorithms as in [97, 99, 102, 104]. (4) We extend a partition-based strategy to enhance the accuracy of $\mathcal{M}$ in predicting partial matches, while [102] makes a prediction based on the embeddings of the entire G and Q, and [99] extends partial matches by aggregating node features using attention mechanism. (5) We improve backtrack-based algorithms for SubIso with ML oracles, while [104] targets join-based methods by filtering paths via GNNs.

Algorithms for SubIso. Prior work on SubIso algorithms can be categorized as follows. (1) Exact algorithms, *e.g.,* CECI [21], DPiso [50], VEQ [62], RM [90], CFL [22] and PathLAD+ [98], which work almost the same as VF3 (Section 3.1), except that they explore different strategies to (a) determine the matching order, (b) compute candidate matches for pattern vertices, and (c) develop different data structures to build indices (see [112] for a recent survey). Different from VF3, [26, 104] enumerate all matches using the join framework after filtering candidates. (2) Approximate matching, which (a) first selects candidate matches for each pattern vertex via similarity score functions [114], the Jaccard similarity and chi-square statistic [40], or statistical significance and Chebyshev's inequality [7], and (b) then computes approximate matches by removing edges and vertexes from given graphs [84], using a neural distance function to find approximate edge alignment between patterns and graphs [86], composing the matches of spanning trees [111], computing the match covers [88] or adopting genetic algorithms [18]. (3) Enumeration speedup, *e.g.,* [38, 46, 47, 52, 67] develop parallel algorithms for SubIso; [51, 108] enumerate matches of path patterns; [66] lists cliques in graphs; and [60] accelerates the enumeration by matching isolated vertices using bipartite graph matching.

Our algorithms for SubIso differs from these methods in the following. (1) We develop enumeration algorithms with ML predictors, as opposed to enumerating all possible candidate matches as exact algorithms, *e.g.,* CECI [21] and DPiso [50]. (2) No matter whether the ML predictor is error-free or not, our algorithms return exact matches, instead of approximate matches [8, 114]. (3) We provide performance guarantees (consistency and robustness) for our enumeration algorithms, which have not been studied before.

8 Conclusion

This work advocates an approach to enumeration problems by unifying algorithmic methods and ML predictions. (1) We show that SubIso is one of the hardest problems in EnumP, and for such problems, it is beyond reach in practice to find an algorithm that is both output-polynomial and β-robust for a constant β unless fewP = P. (2) This said, we develop $\mathsf{VF3}_M$ algorithms that extend VF3 with an ML model $\mathcal{M}$ to predict partial matches. We show that these algorithms are consistent, robust with a high probability when $\mathcal{M}$ makes bad predictions, and output-polynomial with a high probability. (3) We train such a model $\mathcal{M}$ via order-embedding space. (4) Our empirical study has verified that the $\mathsf{VF3}_M$ algorithms work well, and the approach is promising since ML techniques evolve.

One topic for future work is to further improve the accuracy of model $\mathcal{M}$ by fine-tuning for different types of graphs and queries. Another topic is to extend the study to other problems in EnumP.

References

[1] 2021. DBLP collaboration network. *https://snap.stanford.edu/data/com-DBLP.html*.

[2] 2025. Code, datasets and full version. https://anonymous.4open.science/r/NPML-Anonymization.

[3] 2025. LiveJournal. *https://snap.stanford.edu/data/soc-LiveJournal1.html*.

[4] 2025. Pokec. *http://snap.stanford.edu/data/soc-pokec.html*.

[5] 2025. vf3lib. *https://github.com/MiviaLab/vf3lib*.

[6] Anders Aamand, Justin Y. Chen, Huy Lê Nguyen, Sandeep Silwal, and Ali Vakilian. 2023. Improved Frequency Estimation Algorithms with and without Predictions. In *NeurIPS*.

[7] Shubhangi Agarwal, Sourav Dutta, and Arnab Bhattacharya. 2021. VerSaChI: Finding Statistically Significant Subgraph Matches using Chebyshev's Inequality. In *CIKM*. 2812–2816.

[8] Shubhangi Agarwal, Sourav Dutta, and Arnab Bhattacharya. 2024. VeNoM: Approximate Subgraph Matching with Enhanced Neighbourhood Structural Information. In *COMAD/CODS*. 18–26.

[9] Eric Allender and Roy S. Rubinstein. 1988. P-Printable Sets. *SIAM J. Comput.* 17, 6 (1988), 1193–1202.

[10] Keerti Anand, Rong Ge, Amit Kumar, and Debmalya Panigrahi. 2021. A Regression Approach to Learning-Augmented Online Algorithms. In *NeurIPS*. 30504–30517.

[11] Keerti Anand, Rong Ge, and Debmalya Panigrahi. 2020. Customizing ML Predictions for Online Algorithms. In *ICML*, Vol. 119. 303–313.

[12] Spyros Angelopoulos and Shahin Kamali. 2023. Contract Scheduling with Predictions. *J. Artif. Intell. Res.* 77 (2023), 395–426.

[13] Spyros Angelopoulos, Shahin Kamali, and Kimia Shadkami. 2023. Online Bin Packing with Predictions. *J. Artif. Intell. Res.* 78 (2023), 1111–1141.

[14] Antonios Antoniadis, Joan Boyar, Marek Eliás, Lene Monrad Favrholdt, Ruben Hoeksma, Kim S. Larsen, Adam Polak, and Bertrand Simon. 2023. Paging with Succinct Predictions. In *ICML*, Vol. 202. 952–968.

[15] Antonios Antoniadis, Christian Coester, Marek Eliás, Adam Polak, and Bertrand Simon. 2023. Mixing Predictions for Online Metric Algorithms. In *ICML*, Vol. 202. 969–983.

[16] Junya Arai, Yasuhiro Fujiwara, and Makoto Onizuka. 2023. GuP: Fast Subgraph Matching by Guard-based Pruning. In *SIGMOD*.

[17] Xingjian Bai and Christian Coester. 2023. Sorting with Predictions. In *NeurIPS*.

[18] Thomas Bärecke and Marcin Detyniecki. 2007. Combining exhaustive and approximate methods for improved sub-graph matching. In *Progress in Pattern Recognition*. 17–26.

[19] Amos Beimel, Renen Hallak, and Kobbi Nissim. 2009. Private Approximation of Clustering and Vertex Cover. *Comput. Complex.* 18, 3 (2009), 435–494.

[20] Aurélien Bellet, Amaury Habrard, and Marc Sebban. 2015. *Metric learning*. Morgan & Claypool Publishers.

[21] Bibek Bhattarai, Hang Liu, and H. Howie Huang. 2019. CECI: Compact Embedding Cluster Index for Scalable Subgraph Matching. In *SIGMOD*. 1447–1462.

[22] Fei Bi, Lijun Chang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2016. Efficient Subgraph Matching by Postponing Cartesian Products. In *SIGMOD*. 1199–1214.

[23] Vincenzo Bonnici, Rosalba Giugno, Alfredo Pulvirenti, Dennis E. Shasha, and Alfredo Ferro. 2013. A subgraph isomorphism algorithm and its application to biochemical data. *BMC Bioinform.* 14, S-7 (2013), S13.

[24] Joan Boyar, Lene M. Favrholdt, Christian Kudahl, Kim S. Larsen, and Jesper W. Mikkelsen. 2017. Online Algorithms with Advice: A Survey. *ACM Comput. Surv.* 50, 2 (2017), 19:1–19:34.

[25] Joan Boyar, Lene M. Favrholdt, and Kim S. Larsen. 2018. Relative Worst-Order Analysis: A Survey. In *Adventures Between Lower Bounds and Higher Altitudes - Essays Dedicated to Juraj Hromkovič on the Occasion of His 60th Birthday*. 216–230.

[26] Hongtai Cao, Qihao Wang, Xiaodong Li, Matin Najafi, Kevin Chen-Chuan Chang, and Reynold Cheng. 2024. Large Subgraph Matching: A Comprehensive and Efficient Approach for Heterogeneous Graphs. In *ICDE*. 2972–2985.

[27] Florent Capelli and Yann Strozecki. 2017. On The Complexity of Enumeration. *CoRR* abs/1703.01928 (2017). http://arxiv.org/abs/1703.01928

[28] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2017. Introducing VF3: A New Algorithm for Subgraph Isomorphism. In *International Workshop on Graph-Based Representations in Pattern Recognition*. 128–139.

[29] Vincenzo Carletti, Pasquale Foggia, Alessia Saggese, and Mario Vento. 2018. Challenging the Time Complexity of Exact Subgraph Isomorphism for Huge and Dense Graphs with VF3. *IEEE Trans. Pattern Anal. Mach. Intell.* 40, 4, 804–818.

[30] George Casella and Roger Berger. 2001. *Statistical Inference*. Duxbury Resource Center.

[31] Bahzad Charbuty and Adnan Abdulazeez. 2021. Classification based on decision tree algorithm for machine learning. *Journal of Applied Science and Technology Trends* 2, 01 (2021), 20–28.

[32] Richard Cole and Tim Roughgarden. 2015. The Sample Complexity of Revenue Maximization. *CoRR* abs/1502.00963 (2015). arXiv:1502.00963 http://arxiv.org/abs/1502.00963

[33] Stephen A. Cook. 1971. The Complexity of Theorem-Proving Procedures. In *STOC*. 151–158.

[34] Nikhil R. Devanur and Thomas P. Hayes. 2009. The adwords problem: Online keyword matching with budgeted bidders under random permutations. In *ACM Conference on Electronic Commerce (EC)*. 71–78.

[35] Ilias Diakonikolas, Vasilis Kontonis, Christos Tzamos, Ali Vakilian, and Nikos Zarifis. 2021. Learning Online Algorithms with Distributional Advice. In *ICML*, Vol. 139. 2687–2696.

[36] Yannis Dimopoulos, Vangelis Magirou, and Christos H. Papadimitriou. 1997. On Kernels, Defaults and Even Graphs. *Ann. Math. Artif. Intell.* 20, 1-4 (1997), 1–12.

[37] Michael Dinitz, Sungjin Im, Thomas Lavastida, Benjamin Moseley, and Sergei Vassilvitskii. 2022. Algorithms with Prediction Portfolios. In *NeurIPS*.

[38] Vibhor Dodeja, Mohammad Almasri, Rakesh Nagi, Jinjun Xiong, and Wen-Mei Hwu. 2022. PARSEC: PARallel Subgraph Enumeration in CUDA. In *IPDPS*. 168–178.

[39] Marina Drygala, Sai Ganesh Nagarajan, and Ola Svensson. 2023. Online Algorithms with Costly Predictions. In *AISTATS*, Vol. 206. 8078–8101.

[40] Sourav Dutta, Pratik Nayek, and Arnab Bhattacharya. 2017. Neighbor-Aware Search for Approximate Labeled Graph Matching using the Chi-Square Statistics. In *WWW*. 1281–1290.

[41] Wenfei Fan, Yinghui Wu, and Jingbo Xu. 2016. Functional dependencies for graphs. In *SIGMOD*. ACM, 1843–1857.

[42] Jörg Flum and Martin Grohe. 2006. *Parameterized Complexity Theory*. Springer.

[43] Michael Garey and David Johnson. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company.

[44] Themistoklis Gouleakis, Konstantinos Lakis, and Golnoosh Shahkarami. 2023. Learning-Augmented Algorithms for Online TSP on the Line. In *AAAI*. 11989–11996.

[45] F. Richard Guo and Emilija Perkovic. 2022. Efficient Least Squares for Estimating Total Effects under Linearity and Causal Sufficiency. *J. Mach. Learn. Res.* 23 (2022), 104:1–104:41.

[46] Wentian Guo, Yuchen Li, Mo Sha, Bingsheng He, Xiaokui Xiao, and Kian-Lee Tan. 2020. GPU-Accelerated Subgraph Enumeration on Partitioned Graphs. In *SIGMOD*. 1067–1082.

[47] Wentian Guo, Yuchen Li, and Kian-Lee Tan. 2022. Exploiting Reuse for GPU Subgraph Enumeration. *TKDE* 34, 9 (2022), 4231–4244.

[48] Anupam Gupta, Debmalya Panigrahi, Bernardo Subercaseaux, and Kevin Sun. 2022. Augmenting Online Algorithms with ε-Accurate Predictions. In *NeurIPS*.

[49] William L. Hamilton, Zhitao Ying, and Jure Leskovec. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*. 1024–1034.

[50] Myoungji Han, Hyunjoon Kim, Geonmo Gu, Kunsoo Park, and Wook-Shin Han. 2019. Efficient Subgraph Matching: Harmonizing Dynamic Programming, Adaptive Matching Order, and Failing Set Together. In *SIGMOD*. 1429–1446.

[51] Kongzhang Hao, Long Yuan, and Wenjie Zhang. 2021. Distributed Hop-Constrained s-t Simple Path Enumeration at Billion Scale. *PVLDB* 15, 2 (2021), 169–182.

[52] Jiezhong He, Zhouyang Liu, Yixin Chen, Hengyue Pan, Zhen Huang, and Dongsheng Li. 2022. FAST: A Scalable Subgraph Matching Framework over Large Graphs. In *HPEC*. 1–7.

[53] Lane A. Hemaspaandra and Mayur Thakur. 2004. Lower bounds and the hardness of counting properties. *Theor. Comput. Sci.* 326, 1-3 (2004), 1–28.

[54] Alexandru Iosup, Tim Hegeman, Wing Lung Ngai, Stijn Heldens, Arnau Prat-Pérez, Thomas Manhardt, Hassan Chafi, Mihai Capota, Narayanan Sundaram, Michael J. Anderson, Ilie Gabriel Tanase, Yinglong Xia, Lifeng Nai, and Peter A. Boncz. 2016. LDBC Graphalytics: A benchmark for large-scale graph analysis on parallel and distributed platforms. *PVLDB* 9, 13 (2016), 1317–1328.

[55] Guillaume Jaume, An-phi Nguyen, María Rodríguez Martínez, Jean-Philippe Thiran, and Maria Gabrani. 2019. edGNN: A Simple and Powerful GNN for Directed Labeled Graphs. *CoRR* abs/1904.08745 (2019). http://arxiv.org/abs/1904.08745

[56] Junteng Jia and Austin R. Benson. 2020. Outcome Correlation in Graph Neural Network Regression. *CoRR* abs/2002.08274 (2020).

[57] Junteng Jia and Austin R. Benson. 2020. Residual Correlation in Graph Neural Network Regression. In *KDD*. 588–598.

[58] Zhihao Jia, Sina Lin, Mingyu Gao, Matei Zaharia, and Alex Aiken. 2020. Improving the Accuracy, Scalability, and Performance of Graph Neural Networks with Roc. In *MLSys*.

[59] Zhihao Jiang, Debmalya Panigrahi, and Kevin Sun. 2022. Online Algorithms for Weighted Paging with Predictions. *ACM Trans. Algorithms* 18, 4 (2022), 39:1–39:27.

[60] Zite Jiang, Shuai Zhang, Xingzhong Hou, Mengting Yuan, and Haihang You. 2024. IVE: Accelerating Enumeration-Based Subgraph Matching via Exploring Isolated Vertices. In *ICDE*. 4208–4221.

[61] Elias B. Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. 2017. Learning Combinatorial Optimization Algorithms over Graphs. In *NeurIPS*. 6348–6358.

[62] Hyunjoon Kim, Yunyoung Choi, Kunsoo Park, Xuemin Lin, Seok-Hee Hong,

and Wook-Shin Han. 2021. Versatile Equivalences: Speeding up Subgraph Query Processing and Subgraph Matching. In *SIGMOD*. 925–937.
[63] Minsu Kim, Jinkyoo Park, and Joungho Kim. 2021. Learning Collaborative Policies to Solve NP-hard Routing Problems. In *NeurIPS*. 10418–10430.
[64] Tim Kraska, Alex Beutel, Ed H. Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The Case for Learned Index Structures. In *SIGMOD*. 489–504.
[65] Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. 2005. Graphs over time: Densification laws, shrinking diameters and possible explanations. In *SIGKDD*.
[66] Ronghua Li, Sen Gao, Lu Qin, Guoren Wang, Weihua Yang, and Jeffrey Xu Yu. 2020. Ordering Heuristics for k-clique Listing. *PVLDB* 13, 11 (2020), 2536–2548.
[67] Wenqing Lin, Xiaokui Xiao, Xing Xie, and Xiaoli Li. 2017. Network Motif Discovery: A GPU Approach. *TKDE* 29, 3 (2017), 513–528.
[68] Xin Liu and Yangqiu Song. 2022. Graph Convolutional Networks with Dual Message Passing for Subgraph Isomorphism Counting and Matching. In *AAAI*. 7594–7602.
[69] Xuanzhou Liu, Lin Zhang, Jiaqi Sun, Yujiu Yang, and Haiqin Yang. 2023. D2Match: Leveraging Deep Learning and Degeneracy for Subgraph Matching. In *ICML*, Vol. 202. 22454–22472.
[70] Thodoris Lykouris and Sergei Vassilvitskii. 2018. Competitive caching with machine learned advice. *CoRR* abs/1802.05399 (2018). arXiv:1802.05399 http://arxiv.org/abs/1802.05399
[71] Yining Ma, Jingwen Li, Zhiguang Cao, Wen Song, Le Zhang, Zhenghua Chen, and Jing Tang. 2021. Learning to Iteratively Solve Routing Problems with Dual-Aspect Collaborative Transformer. In *NeurIPS*. 11096–11107.
[72] Jessica Maghakian, Russell Lee, Mohammad Hajiesmaili, Jian Li, Ramesh K. Sitaraman, and Zhenhua Liu. 2023. Applied Online Algorithms with Heterogeneous Predictors. In *ICML*, Vol. 202. 23484–23497.
[73] Hamed Masnadi-Shirazi and Nuno Vasconcelos. 2015. A view of margin losses as regularizers of probability estimates. *J. Mach. Learn. Res.* 16 (2015), 2751–2795.
[74] Nikolai Merkel, Ruben Mayer, Tawkir Ahmed Fakir, and Hans-Arno Jacobsen. 2023. Partitioner Selection with EASE to Optimize Distributed Graph Processing. In *ICDE*. 2400–2414.
[75] Azalia Mirhoseini, Anna Goldie, Mustafa Yazgan, Joe Wenjie Jiang, Ebrahim Songhori, Shen Wang, Young-Joon Lee, Eric Johnson, Omkar Pathak, Azade Nova, et al. 2021. A graph placement methodology for fast chip design. *Nature* 594, 7862 (2021), 207–212.
[76] Michael Mitzenmacher and Sergei Vassilvitskii. 2020. Algorithms with Predictions. In *Beyond the Worst-Case Analysis of Algorithms*. 646–662.
[77] Andrés Muñoz Medina and Sergei Vassilvitskii. 2017. Revenue Optimization with Approximate Bid Predictions. *CoRR* abs/1706.04732 (2017). arXiv:1706.04732 http://arxiv.org/abs/1706.04732
[78] Linh Anh Nguyen. 2018. Computing Bisimulation-Based Comparisons. *Fundam. Informaticae* 157, 4 (2018), 385–401.
[79] Iván Olmos, Jesús A. González, and Mauricio Osorio. 2007. Reductions between the Subgraph Isomorphism Problem and Hamiltonian and SAT Problems. In *CONIELECOMP*. 20.
[80] Martin Otto. 2020. Graded modal logic and counting bisimulation. *CoRR* arXiv:1910.00039 (2020). https://doi.org/10.48550/arXiv.1910.00039
[81] Christos H Papadimitriou. 2003. *Computational complexity*. John Wiley and Sons Ltd.
[82] Manish Purohit, Zoya Svitkina, and Ravi Kumar. 2018. Improving Online Algorithms via ML Predictions. In *NeurIPS*. 9684–9693.
[83] Ruizhong Qiu, Zhiqing Sun, and Yiming Yang. 2022. DIMES: A Differentiable Meta Solver for Combinatorial Optimization Problems. In *NeurIPS*.
[84] Jiyuan Ren, Yangfu Liu, Yi Shen, Zhe Wang, and Zhen Luo. 2020. Approximate Sub-graph Matching over Knowledge Graph. In *MobiCASE*, Vol. 341. 198–208.
[85] Dhruv Rohatgi. 2020. Near-Optimal Bounds for Online Caching with Machine Learned Advice. In *SODA*. 1834–1845.
[86] Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. 2022. Interpretable Neural Subgraph Matching for Graph Retrieval. In *AAAI*. 8115–8123.
[87] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. 2018. Modeling Relational Data with Graph Convolutional Networks. In *ESWC (Lecture Notes in Computer Science, Vol. 10843)*. 593–607.
[88] Zhichao Shi, Youhuan Li, Ziming Li, Yuequn Dou, Xionghu Zhong, and Lei Zou. 2025. MatCo: Computing Match Cover of Subgraph Query over Graph Data. In *SIGMOD*.
[89] Haoran Sun, Katayoon Goshvadi, Azade Nova, Dale Schuurmans, and Hanjun Dai. 2023. Revisiting Sampling for Combinatorial Optimization. In *ICML*, Vol. 202. 32859–32874.
[90] Shixuan Sun, Xibo Sun, Yulin Che, Qiong Luo, and Bingsheng He. 2020. RapidMatch: A Holistic Approach to Subgraph Query Processing. *PVLDB* 14, 2 (2020), 176–188.
[91] Zhiqing Sun and Yiming Yang. 2023. DIFUSCO: Graph-based Diffusion Solvers for Combinatorial Optimization. In *NeurIPS*.
[92] Leslie G. Valiant. 1979. The Complexity of Enumeration and Reliability Problems. *SIAM J. Comput.* 8, 3 (1979), 410–421.
[93] Vladimir Vapnik. 1998. *Statistical learning theory*. Wiley.
[94] Luiz Felipe Vecchietti, Minah Seo, and Dongsoo Har. 2022. Sampling Rate Decay in Hindsight Experience Replay for Robot Control. *IEEE Trans. Cybern.* 52, 3 (2022), 1515–1526.
[95] Erik Vee, Sergei Vassilvitskii, and Jayavel Shanmugasundaram. 2010. Optimal online assignment with forecasts. In *ACM Conference on Electronic Commerce (EC)*. 109–118.
[96] Ivan Vendrov, Ryan Kiros, Sanja Fidler, and Raquel Urtasun. 2016. Order-Embeddings of Images and Language. In *ICLR*.
[97] Hanchen Wang, Ying Zhang, Lu Qin, Wei Wang, Wenjie Zhang, and Xuemin Lin. 2022. Reinforcement Learning Based Query Vertex Ordering Model for Subgraph Matching. In *ICDE*. 245–258.
[98] Yiyuan Wang, Chenghou Jin, Shaowei Cai, and Qingwei Lin. 2023. PathLAD+: An Improved Exact Algorithm for Subgraph Isomorphism Problem. In *IJCAI*.
[99] Bo Tang Wenzhe Hou, Xiang Zhao. 2025. OptMatch: An Effcient and Generic Neural Network-assisted Subgraph Matching Approach. In *ICDE*.
[100] Chenyang Xu and Benjamin Moseley. 2022. Learning-Augmented Algorithms for Online Steiner Tree. In *AAAI*. 8744–8752.
[101] Keyulu Xu, Weihua Hu, Jure Leskovec, and Stefanie Jegelka. 2019. How Powerful are Graph Neural Networks?. In *ICLR*. OpenReview.net.
[102] Xin Xue, Tianchen Zhu, Qingyun Sun, Haoyi Zhou, and Jianxin Li. 2025. Efficient Subgraph Matching Algorithm with Graph Neural Network. *Journal of Computer Research and Development*, 62, 3 (2025), 694–708.
[103] Haoran Ye, Jiarui Wang, Zhiguang Cao, Helan Liang, and Yong Li. 2023. DeepACO: Neural-enhanced Ant Systems for Combinatorial Optimization. In *NeurIPS*.
[104] Yutong Ye, Xiang Lian, and Mingsong Chen. 2024. Efficient Exact Subgraph Matching via GNN-based Path Dominance Embedding. *PVLDB* 17, 7 (2024), 1628–1641.
[105] Rex Ying, Tianyu Fu, Andrew Wang, Jiaxuan You, Yu Wang, and Jure Leskovec. 2024. Representation Learning for Frequent Subgraph Mining. *CoRR* abs/2402.14367 (2024). https://doi.org/10.48550/arXiv.2402.14367
[106] Rex Ying, Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, and Jure Leskovec. 2020. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*.
[107] Jiaxuan You, Jonathan Michael Gomes Selman, Rex Ying, and Jure Leskovec. 2021. Identity-aware Graph Neural Networks. In *AAAI*. 10737–10745.
[108] Yuanyuan Zeng, Yixiang Fang, Chenhao Ma, Xu Zhou, and Kenli Li. 2024. Efficient Distributed Hop-Constrained Path Enumeration on Large-Scale Graphs. *Proc. ACM Manag. Data* 2, 1 (2024), 22:1–22:25.
[109] Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. 2020. Learning to Dispatch for Job Shop Scheduling via Deep Reinforcement Learning. In *NeurIPS*.
[110] Dinghuai Zhang, Hanjun Dai, Nikolay Malkin, Aaron C Courville, Yoshua Bengio, and Ling Pan. 2023. Let the flows tell: Solving graph combinatorial problems with gflownets. *Advances in neural information processing systems* 36 (2023), 11952–11969.
[111] Shijie Zhang, Jiong Yang, and Wei Jin. 2010. SAPPER: Subgraph Indexing and Approximate Matching in Large Graphs. *PVLDB* 3, 1 (2010), 1185–1194.
[112] Zhijie Zhang, Yujie Lu, Weiguo Zheng, and Xuemin Lin. 2024. A Comprehensive Survey and Experimental Study of Subgraph Matching: Trends, Unbiasedness, and Interaction. In *SIGMOD*.
[113] Kangfei Zhao, Jeffrey Xu Yu, Hao Zhang, Qiyan Li, and Yu Rong. 2021. A Learned Sketch for Subgraph Counting. In *SIGMOD*. 2142–2155.
[114] Yu Zhao, Chunhong Zhang, Tingting Sun, Yang Ji, Zheng Hu, and Xiaofeng Qiu. 2016. Approximate Subgraph Matching Query over Large Graph. In *BigCom*, Vol. 9784. 247–256.

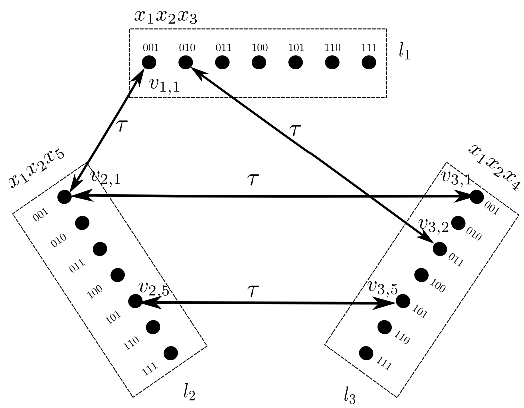


Figure 10: The construction in the reduction

Appendix

Proof of Proposition 1

The SubIso problem is in EnumP, since it is in PTIME to verify whether a given mapping is an isomorphism. To show the EnumP-completeness of SubIso, we next construct a parsimonious reduction from an existing EnumP-complete problem Π_{SAT} [19].

Given a 3SAT formula $\Phi = C_1 \wedge \ldots \wedge C_n$, we construct an instance of the SubIso problem, *i.e.,* a graph $G = (V, E)$ and a pattern Q, such that each satisfying truth assignment of Φ is in one-to-one correspondence with a match of Q in G. More specifically, the graph $G = (V, E, L)$ and the pattern Q are constructed as follows:

(1) Graph $G = (V, E, L)$ represents relations between different truth assignments of the 3SAT instance Φ. More specifically, (a) $V = \cup_{i \in [1,n]} \{v_{i,1}, \ldots, v_{i,7}\}$, *i.e.,* for each clause C_i ($i \in [1, n]$), V contains seven vertices $v_{i,1}, \ldots, v_{i,7}$ such that each vertex corresponds to a satisfying truth assignment for C_i; (b) the edges in E connect consistent assignments, *i.e.,* given two vertices (*i.e.,* two truth assignments) $v_{i,j}$ and $v_{k,l}$, there exist two directed edges between $v_{i,j}$ and $v_{k,l}$ labeled τ, *i.e.,* $(v_{i,j}, \tau, v_{k,l})$ and $(v_{k,l}, \tau, v_{i,j})$, if their common variables are assigned the same values in the two assignments; and (c) the function L assigns the labels as follows: vertices $v_{i,1}, \ldots, v_{i,7}$ carry label l_i to represent clause C_i in Φ.

For example, let $\Phi = (x_1 \vee x_2 \vee x_3) \wedge (x_1 \vee x_2 \vee x_5) \wedge (x_1 \vee x_2 \vee x_4)$. Figure 10 shows parts of the graph G. (a) The seven vertices on the top represent the seven satisfying assignments for the first clause $x_1 \vee x_2 \vee x_3$; (b) there exist two directed edges between vertices $v_{1,1}$ and $v_{2,1}$, since $v_{1,1}$ and $v_{2,1}$ represent assignments $\{x_1 = 0, x_2 = 0, x_3 = 1\}$ and $\{x_1 = 0, x_2 = 0, x_5 = 1\}$, respectively, and the two assignments are consistent, *i.e.,* the same variable is assigned the same value; (c) there does not exist an edge between $v_{2,1}$ and $v_{3,2}$, since variable x_2 is assigned different values 0 and 1; and (d) vertices $v_{1,1}, \ldots, v_{1,7}$ (*i.e.,* the seven satisfying truth assignments for the first clause) carry the same label l_1; and the vertices for the second and third clauses carry labels l_2 and l_3, respectively.

(2) Pattern Q is a clique of size n, *i.e.,* the number of clauses in Φ. Vertices $v_1, \ldots, v_n$ in Q carry distinct labels $l_1, \ldots, l_n$, respectively.

This completes the construction of G and Q. Clearly, G and Q can be constructed in PTIME, since (a) there exist $7n$ vertices and at most $49n^2$ many edges in G, and (b) Q has n vertices and $n(n-1)$ many edges. We show that this is a parsimonious reduction following [19]. That is, each satisfying truth assignment of Φ is in one-to-one correspondence with each clique in G.

($\Rightarrow$) Given a satisfying truth assignment μ of Φ, we construct a match h of Q (*i.e.,* a clique of size n) in G as follows: for each clause C_i carrying variables $x_{i,1}, x_{i,2}$ and $x_{i,3}$, vertex v_i in Q is mapped to a vertex $v_{i,j}$ in G representing the assignment $x_{i,1} = \mu(x_{i,1}), x_{i,2} = \mu(x_{i,2})$ and $x_{i,3} = \mu(x_{i,3})$. The mapping is unique, since the truth assignment of C_i is fixed in μ, and there exists a unique vertex labeled l_i representing this truth assignment. Hence the reduction is parsimonious. It remains to show that there exists an edge between $h(x_i)$ and $h(x_j)$ for any $i, j \in [1, n]$ with $i \neq j$. Indeed, since all clauses in Φ are satisfied under the truth assignment μ and each variable is assigned only one value in μ, the truth assignments represented by $h(x_i)$ and $h(x_j)$ are consistent, *i.e.,* the same variable is assigned the same value. So, there exists an edge between $h(x_i)$ and $h(x_j)$ for $i, j \in [1, n]$ with $i \neq j$. Thus h is a full match of Q in G.

($\Leftarrow$) Given a full match h of pattern Q in graph G, we deduce a satisfying truth assignment μ of Φ. More specifically, since the labels of vertices in G are distinct, each vertex v_i in Q is mapped to a satisfying truth assignment of clause C_i. We define μ as follows: assume that clause C_i carries variables $x_{i,1}, x_{i,2}$ and $x_{i,3}$, and vertex $h(v_i)$ in G encodes the truth assignment $x_{i,1} = c_1, x_{i,2} = c_1$ and $x_{i,3} = c_3$; then we set $\mu(x_{i,1}) = c_1, \mu(x_{i,2}) = c_2$ and $\mu(x_{i,3}) = c_3$. This truth assignment is unique, since the full match h is fixed and each vertex labeled l_i in G represents only one truth assignment of C_i. Hence the reduction is parsimonious. We next show that μ is a satisfying truth assignment of Φ. Indeed, (a) μ is consistent, since Q is a clique, and edges link only truth assignments that are consistent; and (b) Φ is satisfied by the assignment μ, since each vertex v_i in Q is mapped to a satisfying truth assignment of clause C_i, and pattern Q has n vertices carrying different labels. □

Proof of Theorem 2

We show that no algorithm exists for SubIso that is both output polynomial and β-robust for any positive constant β unless fewP = P. It suffices to show that there exists no output polynomial algorithm for SubIso whose recall is larger than 0. We prove that if such algorithm exists, then we can deduce fewP $\subseteq$ P. Then fewP = P.

Assume that there exists an output polynomial algorithm $\mathcal{A}$ for SubIso whose F1 score is at least β. We show that fewP $\subseteq$ P. We construct a reduction from fewP to a subset of SubIso, denoted by $\mathsf{SubIso}^{\mathsf{P}}$, which consists of SubIso instances $B = (Q, G)$ such that there exist polynomial many matches of Q in G. Let $A = (x, M)$ be an instance of fewP, where x is the input and M is a nondeterministic Turing machine. We can construct a parsimonious reduction from $A = (x, M)$ to an instance $B = (Q, G)$ in $\mathsf{SubIso}^{\mathsf{P}}$ following [33, 79]. Here a parsimonious reduction is a PTIME reduction preserving the number of solutions (see Section 2; [81]). Using $\mathcal{A}$, we develop algorithm $\mathcal{B}$ to check whether M accepts x as follows: run $\mathcal{A}$ to compute $Q(G)$; if $\mathcal{A}$ returns at least one match, then return true, *i.e.,* M accepts x; otherwise, return false, *i.e.,* M does not accept x.

We next show the correctness of the reduction. That is, if the F1 measure of $\mathcal{A}$ is larger than 0, then $\mathcal{B}$ runs in polynomial time and correctly answers whether M accepts x. In the following, assume that $\mathcal{A}$ is output polynomial and the F1 measure of $\mathcal{A}$ is above 0.

(1) We first show that algorithm $\mathcal{B}$ runs in polynomial time. Indeed, (a) since $A = (x, M)$ belongs to fewP and the reduction preserves the number of solutions, the number of matches in $Q(G)$ is bounded by $P(|x|, |M|)$ for some polynomial P, where $|x|$ and $|M|$ are the sizes of x and M, respectively [81]. Meanwhile, (b) since algorithm $\mathcal{A}$ is output polynomial, $\mathcal{A}$ runs in $P_1(P(|x|, |M|))$ time for some polynomial P_1. Therefore, algorithm $\mathcal{B}$ runs in polynomial time.

(2) We next show that if the F1 score of algorithm $\mathcal{A}$ is larger than β, then $\mathcal{B}$ can correctly answer whether the Turing machine M accepts the input x. This is to show that M accepts x if and only if $\mathcal{A}$ returns at least one match. (1) When $\mathcal{A}$ returns at least one match, by the reduction, there exist polynomially many accepting computations of M on x. That is, M accepts the input x. (2) On the other hand, if M accepts x, the number of accepting computations is bounded by a polynomial $P_2(|x|, |M|) > 0$. From the parsimonious reduction, there exist $P_2(|x|, |M|)$ many matches of Q in G. When the F1 measure of algorithm $\mathcal{A}$ is larger than 0, $\mathcal{A}$ returns at least one match, since $P(|x|, |M|) > 0$.

Therefore, there exists a polynomial time algorithm $\mathcal{B}$ that can answer whether M accepts x. Then fewP = P. □

Proof of Theorem 4

Since the precision of $\mathsf{VF3}_M^N$ is 1, it suffices to show that the recall of $\mathsf{VF3}_M^N$ is at least $\frac{\text{Output}}{\text{Output}+f(\varepsilon)}$ with a high probability. Then one can see that $F1(\mathsf{VF3}_M^N) = \frac{2\cdot\text{precision}\cdot\text{recall}}{\text{precision}+\text{recall}} = \frac{2\cdot\text{recall}}{1+\text{recall}} \geq \frac{2\text{Output}}{2\text{Output}+f(\varepsilon)}$.

Let $\mathcal{R}$ be the number of all matches of Q in G, *i.e.,* $\mathcal{R} = |Q(G)|$. Then $\mathcal{R}$ = Output + Miss, where Miss is the number of missed matches. Since recall $= \frac{\text{Output}}{\text{Output}+\text{Miss}}$, to prove the lower bound for the recall of $\mathsf{VF3}_M^N$, it suffices to prove an upper bound for Miss. We prove that the number of missed matches for each false prediction of $\mathcal{M}$ is bounded by $\mathcal{B}$ with a probability of at least $1 - \frac{\varepsilon}{N_{\mathcal{N}}}$. Here $N_{\mathcal{N}}$ is the number of false predictions of $\mathcal{M}$, which can be recorded during the running of $\mathsf{VF3}_M^N$, and $\mathcal{B} = \frac{1+\eta|G|-|G|+\sqrt{(|G|-1-|G|\eta)^2-4(1-\frac{\varepsilon}{N_{\mathcal{N}}})}}{2\eta}$. If this holds, we use the generalized Bonferroni inequality [30] to show that the recall of $\mathsf{VF3}_M^N$ is bounded by $\frac{\text{Output}}{\text{Output}+N_{\mathcal{N}}\mathcal{B}}$ with a probability of at least $1-\varepsilon$.

That is, we prove the theorem in the following two steps.

(1) We first show that the number of missed matches for each false prediction is bounded by $\mathcal{B}$ with a probability of at least $1 - \frac{\varepsilon}{N_{\mathcal{N}}}$.

Let ρ be a partial match on which $\mathcal{M}$ returns false. Let $Y_1, \ldots, Y_{N_{\mathcal{T}}}$ be all full matches that are extended from ρ. Since $\mathcal{M}$ is monotonically decreasing, $\mathcal{M}$ returns false for all partial matches extended from ρ. Because the error rate of $\mathcal{M}$ is η, we show that the probability that all of $Y_1, \ldots, Y_{N_{\mathcal{T}}}$ are missed is less than $\frac{\varepsilon}{N_{\mathcal{N}}}$. More specifically, the probability can be computed as follows:

$$
\begin{aligned}
&P(\overline{Y_1} \cap \ldots \cap \overline{Y_{N_{\mathcal{T}}}}) \\
&\leq 1 - \Big(\sum_{i=1}^{N_{\mathcal{T}}} P(\overline{Y_i}) - \sum_{1=i<j\leq N_{\mathcal{T}}}^{N_{\mathcal{T}}} P(\overline{Y_i} \cap \overline{Y_j})\Big) \qquad (1)\\
&\leq 1 - \Big(\eta N_{\mathcal{T}} - \sum_{\substack{1=i<j\leq N_{\mathcal{T}} \\ \wedge \overline{Y_i}\oplus\overline{Y_j}}}^{N_{\mathcal{T}}} P(\overline{Y_i}) - \sum_{\substack{1=i<j\leq N_{\mathcal{T}} \\ \wedge\neg(\overline{Y_i}\oplus\overline{Y_j})}}^{N_{\mathcal{T}}} P(\overline{Y_i})P(\overline{Y_j})\Big) \qquad (2)\\
&\leq 1 - \eta N_{\mathcal{T}} + \frac{N_{\mathcal{T}}}{|G|}|G|^2\eta + \Big(N_{\mathcal{T}}^2 - \frac{N_{\mathcal{T}}}{|G|}|G|^2\Big)\eta^2 \qquad (3)\\
&\leq 1 - \eta N_{\mathcal{T}} + N_{\mathcal{T}}|G|\eta + N_{\mathcal{T}}^2\eta^2 - N_{\mathcal{T}}|G|\eta^2\\
&\leq \eta^2 N_{\mathcal{T}}^2 + (|G| - 1 - |G|\eta)\eta N_{\mathcal{T}} + 1
\end{aligned}
$$

Inequality (1) holds due to the inclusion–exclusion principle [30]. The predicate $Y_i \oplus Y_j$ in Inequality (2) denotes that full matches Y_i and Y_j share the same partial matches, *i.e.,* the predictions of $\mathcal{M}$ on Y_i and Y_j are not independent. Inequality (2) holds since we assume that for any two partial matches ρ_1 and ρ_2, if they have two different vertices in G mapped to the same pattern vertex u_i in Q with $i \neq |Q|$, *i.e.,* u_i is not the last pattern node in the matching order O, then the predictions of $\mathcal{M}$ on ρ_1 and ρ_2 are independent (see below). Here (a) the probability $P(\overline{Y_i})$ of making an FN prediction is η; (b) $\mathsf{VF3}_M^N$ enumerates all candidate matches for the last pattern node u in the matching order O, without using $\mathcal{M}$. Inequality (3) holds since (a) each full match h has at most $|G|$ many full matches sharing the same partial match ρ, where the mappings of all pattern vertices except u are fixed, and (b) $\mathsf{VF3}_M^N$ directly enumerates candidate matches of u without using $\mathcal{M}$.

The independence assumption for ML models has been used in the analysis of ML models [45, 56, 57, 101]. Intuitively, given two partial matches ρ_1 and ρ_2 that are independent, the prediction of $\mathcal{M}$ on ρ_1 does not use the prediction on ρ_2, and vice versa [57]. Then we can extend ρ_1 and ρ_2 to graphs, in which the predictions of $\mathcal{M}$ on ρ_1 and ρ_2 are independent. More specifically, (a) we can construct two graphs G_1 and G_2 such that (i) both ρ_1 and ρ_2 are partial matches of Q in G_1 and G_2, respectively; and (ii) ρ_1 (resp. ρ_2) is a valid (resp. invalid) partial match. When the precision of $\mathcal{M}$ is high, the predictions of $\mathcal{M}$ on partial matches ρ_1 and ρ_2 are different. (b) Similarly, we can construct two graphs G_1' and G_2' such that (i) both ρ_1 and ρ_2 are partial matches of Q in G_1' and G_2', respectively; and (ii) the predictions of $\mathcal{M}$ on ρ_1 and ρ_2 are the same. That is, we cannot determine the prediction of $\mathcal{M}$ on ρ_1 based on the prediction of $\mathcal{M}$ on ρ_2, and vice versa.

By setting $N_{\mathcal{T}}^2\eta^2 + (|G| - 1 - |G|\eta)\eta N_{\mathcal{T}} + 1 \leq \frac{\varepsilon}{N_{\mathcal{N}}}$, we have that $N_{\mathcal{T}} \leq \frac{1+\eta|G|-|G|+\sqrt{(|G|-1-|G|\eta)^2-4(1-\frac{\varepsilon}{N_{\mathcal{N}}})}}{2\eta}$. Then with a probability $\frac{\varepsilon}{N_{\mathcal{N}}}$, the number of missed matches is bounded by $\mathcal{B} = \frac{1+\eta|G|-|G|+\sqrt{(|G|-1-|G|\eta)^2-4(1-\frac{\varepsilon}{N_{\mathcal{N}}})}}{2\eta}$ for a false prediction of $\mathcal{M}$.

(2) Using the generalized Bonferroni inequality [30], we show that the recall of $\mathsf{VF3}_M^N$ is bounded by $\frac{\text{Output}}{\text{Output}+N_{\mathcal{N}}\mathcal{B}}$ with a probability of at least $1 - \varepsilon$. Let $p_1, \ldots, p_{N_{\mathcal{N}}}$ be all false predictions of $\mathcal{M}$, and $P(p_1), \ldots, P(p_{N_{\mathcal{N}}})$ be the probabilities that $\mathsf{VF3}_M^N$ misses at most $\mathcal{B}$ full matches for each false prediction p_i ($i \in [1, N_{\mathcal{N}}]$). Then the generalized Bonferroni inequality states that the probability that $\mathsf{VF3}_M^N$ misses $N_{\mathcal{N}}\mathcal{B}$ many full matches is no less than $\sum_{i=1}^{N_{\mathcal{N}}} P(p_i) - (N_{\mathcal{N}} - 1)$,

i.e., $P(\bigcap_{i\in[1,N_{\mathcal{N}}]} p_i) \geq \sum_{i=1}^{N_{\mathcal{N}}} P(p_i) - (N_{\mathcal{N}}-1)$, where $\bigcap_{i\in[1,N_{\mathcal{N}}]} p_i$ denotes that all false predictions miss at most $N_{\mathcal{N}}\mathcal{B}$ many full matches.

As shown above, $P(p_i) \geq 1 - \frac{\varepsilon}{N_{\mathcal{N}}}$ with $i \in [1, N_{\mathcal{N}}]$. Based on the generalized Bonferroni inequality, we have that $P(\bigcap_{i\in[1,N_{\mathcal{N}}]} p_i) \geq \sum_{i=1}^{N_{\mathcal{N}}} P(p_i) - (N_{\mathcal{N}} - 1) \geq N_{\mathcal{N}}(1 - \frac{\varepsilon}{N_{\mathcal{N}}}) - (N_{\mathcal{N}} - 1) = 1 - \varepsilon$. □

Proof of Theorem 5

We prove the lower bounds for $\mathsf{VF3}_{\mathcal{M}}^{O}$ and $\mathsf{VF3}_{\mathcal{M}}^{D}$ as follows.

(1) We first show that the recall of $\mathsf{VF3}_{\mathcal{M}}^{O}$ is at least $\frac{\mathrm{Output}_T+\mathrm{Output}_{FT}}{\mathrm{Output}_T+\mathrm{Output}_{FT}+(1-p_u)N_{\mathcal{N}}\mathcal{B}}$ with a probability of $1 - \varepsilon$.

Observe the following. (A) for full matches, the predictions of model $\mathcal{M}$ can be categorized as follows: the confidence of $\mathcal{M}$ is (a) above δ_{T}; (b) between δ_{T} and δ_{F}; and (c) below δ_{F}. (B) The numbers of full matches in cases (a) and (b) are Output_T and Output_{FT}, respectively. (C) There exist at most $(1 - p_{\mathsf{u}})N_{\mathcal{N}}\mathcal{B}$ full matches of class (c) with a probability of $1 - \varepsilon$, and all these full matches are missed by $\mathsf{VF3}_{\mathcal{M}}^{O}$. Indeed, (i) there exist at most $N_{\mathcal{N}}\mathcal{B}$ full matches at which the confidence of $\mathcal{M}$ is in the range $[0, \delta_T)$, as shown in Theorem 4; and (ii) among these matches, there exist at most $(1 - p_{\mathsf{u}})N_{\mathcal{N}}\mathcal{B}$ full matches at which the confidences of $\mathcal{M}$ is below δ_{F}, by the definition of p_{u}. Thus the recall of $\mathsf{VF3}_{\mathcal{M}}^{O}$ is at least $\frac{\mathrm{Output}_T+\mathrm{Output}_{FT}}{\mathrm{Output}_T+\mathrm{Output}_{FT}+(1-p_u)N_{\mathcal{N}}\mathcal{B}}$ with a probability of $1 - \varepsilon$.

(2) The analysis of $\mathsf{VF3}_{\mathcal{M}}^{D}$ is similar, except that (a) there exist Output_F returned matches at which the confidence of $\mathcal{M}$ is in the range $[0, \delta_F)$. Therefore, the recall of $\mathsf{VF3}_{\mathcal{M}}^{D}$ is at least $\frac{2(\mathrm{Output}_T+\mathrm{Output}_{FT}+\mathrm{Output}_F)}{2\mathrm{Output}_T+2\mathrm{Output}_{FT}+2\mathrm{Output}_F+(1-p_u)N_{\mathcal{N}}\mathcal{B}}$ with a probability of $1-\varepsilon$.

This completes the proof of Theorem 5. □

Proof of Theorem 6

We prove the theorem in two steps: (1) when $\mathcal{M}$ is error-free (*i.e.,* $\eta = 0$), $\mathsf{VF3}_{\mathcal{M}}^{N}$ is output polynomial; and (2) when the error rate of $\mathcal{M}$ is η, $\mathsf{VF3}_{\mathcal{M}}^{N}$ is output polynomial with a probability of at least $1 - \varepsilon$. To prove (2), we use an inequality deduced when proving (1).

(1) We first show that when $\eta = 0$, $\mathsf{VF3}_{\mathcal{M}}^{N}$ is output polynomial. It suffices to show that $\mathcal{M}$ makes at most $O(|G|(\mathrm{Output}_T + 1))$ many false predictions, which will be used to prove (2). If it holds, $\mathsf{VF3}_{\mathcal{M}}^{N}$ runs in $O(|Q||G|(\mathrm{Output}_T + 1))$ time, *i.e.,* $\mathsf{VF3}_{\mathcal{M}}^{N}$ is output polynomial. Observe that (a) when $\mathcal{M}$ returns true for a partial match ρ, there exist full matches extended from the partial matches, and the total number of true predictions is bounded by $O(|Q|\mathrm{Output}_T)$; and (b) when $\mathcal{M}$ returns false for a partial match ρ, $\mathsf{VF3}_{\mathcal{M}}^{N}$ discards ρ and continues to check other partial match.

We next show that $\mathcal{M}$ makes at most $O(|G|(\mathrm{Output}_T + 1))$ many false predictions. To this end, we first define a partial order $\prec$ over partial matches. More specifically, given two partial matches ρ_1 and ρ_2, denote by $\rho_2 \prec \rho_1$ if ρ_1 is extended from ρ_2. Denote by $\mathcal{S}$ the set of minimal partial matches $\rho_{\min}$ satisfying the following two conditions: (1) $\mathcal{M}$ returns false on $\rho_{\min}$, and (2) $\mathcal{M}$ returns true on all partial ρ_p matches such that $\rho_p \prec \rho_{\min}$, *i.e.,* $\rho_{\min}$ is extended from ρ_p. Then $|\mathcal{S}| \leq |G| \times (\mathrm{Output} + 1)$. Indeed, for each minimal partial match ρ'_i, consider the following two cases: (1) when there exists a true prediction ρ' such that $\rho' \prec \rho'_i$ and ρ'_i is directly extended from ρ', the number of such minimal partial matches is bounded by the number of true predictions, *i.e.,* Output. (2) When ρ'_i is not extended from a partial match at which $\mathcal{M}$ makes a true prediction, since the error rate of $\mathcal{M}$ is 0, ρ'_i can only be extended from the initial partial match $\emptyset$. Note that $\mathsf{VF3}_{\mathcal{M}}^{N}$ does not expand any partial match at which $\mathcal{M}$ makes a false prediction. So only the first pattern vertex u in the matching order O is mapped in ρ'_i, and the number of such partial matches is bounded by $|G|$, since u_1 has at most $|G|$ many candidate matches.

(2) We next show that when the error rate of $\mathcal{M}$ is $\eta > 0$, $\mathsf{VF3}_{\mathcal{M}}^{N}$ is output polynomial with a probability of at least $1 - \varepsilon$. Consider the following four cases of the predictions of $\mathcal{M}$: (a) true-negative predictions, (b) false-negative predictions, (c) true-positive predictions, and (d) false-positive predictions. For cases (a) and (c), we can verify that $\mathcal{M}$ is output polynomial as in (1). For case (b), we can verify that the number of such predictions is bounded by $|G|(\mathrm{Output}+1)$, similar to the inequality in (1) (see more about this below). For case (d), we show that the number of such predictions is bounded by $|G|(\mathrm{Output}_T + 1) \times C$ with a probability of at least $1 - \varepsilon$, where $C = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\varepsilon)}}{2(1-\eta)}$.

(I) At first, we define the minimal partial matches as in (1). Let $\mathcal{S} = \{\rho_1, \ldots, \rho_{\mathcal{K}}\}$ be all partial matches on which M makes false positive predictions. Assume that $\rho'_1, \ldots, \rho'_{|\mathcal{S}|}$ are all the minimal partial matches in $\mathcal{S}$ based on the defined order $\prec$. Moreover, we can show that $|\mathcal{S}| \leq O(|G|(\mathrm{Output} + 1))$ due to the proof in (1).

(II) We first show that with a probability of at least $1 - \frac{\varepsilon}{|\mathcal{S}|}$, for any minimal partial match ρ'_k ($k \in [1, |\mathcal{S}|]$), there are at most $C' = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\frac{\varepsilon}{|\mathcal{S}|})}}{2(1-\eta)}$ false-positive (FP) predictions when $\mathcal{M}$ checks partial matches extended from ρ'_k. Let $Y_1, \ldots, Y_{N_k}$ be all FP predictions when $\mathcal{M}$ inspects partial matches extended from ρ'_k. As the error rate of $\mathcal{M}$ is η, we show that the probability that $Y_1, \ldots, Y_{N_k}$ are all predicted true is less than $\frac{\varepsilon}{|\mathcal{S}|}$. More specifically, the probability can be computed as:

$$
\begin{aligned}
& P(Y_1 \cap \ldots \cap Y_{N_k}) \\
\leq & 1 - \Big(\sum_{i=1}^{N_k} P(\overline{Y_i}) - \sum_{1=i<j\leq N_k} P(\overline{Y_i} \cap \overline{Y_j})\Big) && (1) \\
\leq & 1 - \Big((1-\eta)N_k - \sum_{\substack{1=i<j\leq N_k \\ \wedge Y_i \oplus Y_j}} P(\overline{Y_i}) - \sum_{\substack{1=i<j\leq N_k \\ \wedge \neg Y_i \oplus Y_j}} P(\overline{Y_i})P(\overline{Y_j})\Big) && (2) \\
\leq & 1 - (1-\eta)N_k + \frac{N_k}{|G|}|G|^2(1-\eta) + \Big({N_k}^2 - \frac{N_k}{|G|}|G|^2\Big)(1-\eta)^2 && (3) \\
\leq & 1 - (1-\eta)N_k + N_k|G|(1-\eta) + {N_k}^2(1-\eta)^2 - N_k|G|(1-\eta)^2 \\
\leq & (1-\eta)^2{N_k}^2 + (|G| - 1 - |G|(1-\eta))(1-\eta)N_k + 1
\end{aligned}
$$

Inequalities (1), (2) and (3) hold due to the same reasons given for their counterparts in the proof of Theorem 4. Different from the proof of Theorem 4, the probability $P(\overline{Y_i})$ of making a correct prediction $\overline{Y_i}$ with $i \in [1, N_k]$ is $1-\eta$. Recall that $\overline{Y_i}$ ($i \in [1, N_k]$) denotes that $\mathcal{M}$ makes a false prediction on a partial match extended from the minimal partial match ρ'_i, which is a correct prediction.

By setting $(1-\eta)^2 {N_k}^2+(|G|-1-|G|(1-\eta))(1-\eta)N_k+1 \leq \frac{\varepsilon}{|\mathcal{S}|}$, we have that $N_k \leq C' = \frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\frac{\varepsilon}{|\mathcal{S}|})}}{2(1-\eta)}$. Then with a probability of at least $\frac{\varepsilon}{|\mathcal{S}|}$, the number of false-positive predictions is bounded by C' for each minimal partial match ρ'_k.

(III) Using the generalized Bonferroni inequality [30], we show that $\mathsf{VF3}_M^N$ is output polynomial with a probability of at least $1-\varepsilon$. Let $p'_1, \ldots, p'_{|\mathcal{S}|}$ be the minimal partial matches in $\mathcal{S}$, and $P(p'_1), \ldots, P(p'_{|\mathcal{S}|})$ be the probabilities that $\mathsf{VF3}_M^N$ makes at most C' FP predictions for each minimal partial match p_i ($i \in [1, |\mathcal{S}|]$). Then we show that the probability that $\mathcal{M}$ makes at most $|\mathcal{S}|C'$ FP predictions is no less than $\sum_{i=1}^{|\mathcal{S}|} P(p'_i)-(|\mathcal{S}|-1)$, *i.e.,* $P(\bigcap_{i\in[1,|\mathcal{S}|]} p'_i) \geq \sum_{i=1}^{|\mathcal{S}|} P(p'_i)-(|\mathcal{S}|-1)$, where $\bigcap_{i\in[1,|\mathcal{S}|]} p'_i$ denotes that all $|\mathcal{S}|$ minimal partial matches leads to at most $|\mathcal{S}|C'$ FP predictions.

As shown above, $P(p'_i) \geq 1-\frac{\varepsilon}{|\mathcal{S}|}$ with $i \in [1, |\mathcal{S}|]$. Based on the generalized Bonferroni inequality, we have that $P(\bigcap_{i\in[1,|\mathcal{S}|]} p'_i) \geq \sum_{i=1}^{|\mathcal{S}|} P(p'_i)-(|\mathcal{S}|-1) \geq |\mathcal{S}|(1-\frac{\varepsilon}{|\mathcal{S}|})-(|\mathcal{S}|-1) = 1-\varepsilon$.

Therefore, the number of FP predictions is bounded by $|\mathcal{S}|C' \leq |G|(\mathsf{Output}_T+1)\times\frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\frac{\varepsilon}{|\mathcal{S}|})}}{2(1-\eta)} \leq |G|(\mathsf{Output}_T+1)\times\frac{1+(1-\eta)|G|-|G|+\sqrt{(|G|-1-|G|(1-\eta))^2-4(1-\varepsilon)}}{2(1-\eta)} = |G|(\mathsf{Output}_T+1)\times C$ with a probability of at least $1-\varepsilon$. □

Proof of Corollary 7

It suffices to show that the complexity of $\mathsf{VF3}_M^N$ is bounded by $2|G|Cf(Q,\mathcal{M})\cdot\mathsf{OPT}(Q,G)$ with a probability of at least $1-\varepsilon$, where OPT is the cost incurred by the optimal offline algorithm, and $f(Q,\mathcal{M})$ denotes the cost for $\mathcal{M}$ to make a prediction. Observe that (1) the cost of $\mathsf{VF3}_M^N$ is bounded by $|G|(\mathsf{Output}_T+1)\times C\times f(Q,M)$ with a probability of at least $1-\varepsilon$ by Theorem 6; and (2) $\mathsf{Output}_T \leq \mathsf{OPT}$, *i.e.,* the cost incurred by the optimal offline algorithm is at least Output_T, since the precision of $\mathsf{VF3}_M^N$ is 1, and there exist at least Output_T many full matches in $Q(G)$. Then, the cost of $\mathsf{VF3}_M^N$ is at most $|G|(\mathsf{Output}_T+1)\times C\times f(Q,\mathcal{M}) \leq 2|G|Cf(Q,\mathcal{M})\cdot\mathsf{OPT}$ with a probability of at least $1-\varepsilon$. So, the competitive ratio of $\mathsf{VF3}_M^N$ is at least $2|G|Cf(Q,\mathcal{M})$ with a probability of at least $1-\varepsilon$. □

Proof of Proposition 8

(1) We first prove the competitive ratio for $\mathsf{VF3}_M^O$. We partition the partial matches into three parts based on the predictions of $\mathcal{M}$: (a) when $\mathcal{M}$ has confidence above δ_T; (b) when $\mathcal{M}$ has confidence between δ_T and δ_F; and (c) when $\mathcal{M}$ has confidence below δ_F.

We analyze these parts as follows:

(A) For part (a), the computation is similar to that of $\mathsf{VF3}_M^N$, except that $\mathsf{VF3}_M^O$ calls $\mathsf{IsFeasible}$. Because the complexity of $\mathsf{IsFeasible}$ is bounded by $|G||Q|$ [29], and the complexity of $\mathsf{VF3}_M^N$ is bounded by $|G|(\mathsf{Output}_T+1)\times C\times f(Q,\mathcal{M})$ (see the proof of Corollary 7), the complexity of $\mathsf{VF3}_M^O$ is bounded by $|G|(\mathsf{Output}_T+1)\times C\times(f(Q,\mathcal{M})+|G||Q|) \leq 2|G|C(f(Q,\mathcal{M})+|G||Q|)\cdot\mathsf{OPT}$.

(B) For part (c), it simply returns false, and its cost is bounded by the number of false predictions just like in the case of $\mathsf{VF3}_M^N$. However, some of these predictions may be further checked via procedure $\mathsf{IsFeasible}$. Therefore, the complexity in this case is also bounded by $2|G|C(f(Q,\mathcal{M})+|G||Q|)\cdot\mathsf{OPT}$.

(C) For part (b), the number of full matches at which $\mathcal{M}$ has confidences in the range $[\delta_\mathsf{F}, \delta_\mathsf{T})$ is at least $p_\mathsf{u}N_\mathcal{N}\mathcal{B}$ by the definition of p_u. The complexity of part (b) is bounded by $p_\mathsf{u}N_\mathcal{N}\mathcal{B}|Q||G| \leq p_\mathsf{u}\frac{N_\mathcal{N}}{\mathsf{OPT}}\mathsf{OPT}\mathcal{B}|Q||G| \leq p_\mathsf{u}\frac{N_\mathcal{N}}{\mathsf{Output}}\mathcal{B}|Q||G|\cdot\mathsf{OPT}$.

Putting these together, the competitive ratio of $\mathsf{VF3}_M^O$ is bounded by $4|G|C(f(Q,\mathcal{M})+|G||Q|)+p_\mathsf{u}\frac{N_\mathcal{N}}{\mathsf{Output}}\mathcal{B}|Q||G| \leq 5|G|C|Q|p_\mathsf{u}\frac{N_\mathcal{N}}{\mathsf{Output}}\mathcal{B}(f(Q,\mathcal{M})+|G||Q|) = 5N_c(f(Q,\mathcal{M})+|G||Q|)$.

(2) For the competitive ratio of $\mathsf{VF3}_M^D$, observe that compared with $\mathsf{VF3}_M^O$, $\mathsf{VF3}_M^D$ further conducts deep checking to reduce FNs. The complexity of deep checking is bounded by $|G|^2|Q|$, since it is conducted for at most $|G|$ times, and each deep checking takes at most $|G||Q|$ time. Similar to the analysis of $\mathsf{VF3}_M^O$, one can verify that the competitive ratio of $\mathsf{VF3}_M^D$ is bounded by $4|G|C(f(Q,\mathcal{M})+|G||Q|+|G|^2|Q|)+p_\mathsf{u}\frac{N_\mathcal{N}}{\mathsf{Output}}\mathcal{B}|Q||G| \leq 5|G|C|Q|p_\mathsf{u}\frac{N_\mathcal{N}}{\mathsf{Output}}\mathcal{B}(f(Q,\mathcal{M})+2|G|^2|Q|)$=$5N_c(f(Q,\mathcal{M})+2|G|^2|Q|)$. □