Answering Regular Path Queries through Exemplars

Komal Chauhan
Department of CSE, IIT Delhi
New Delhi, India
komalc1612@gmail.com

Kartik Jain
Department of CSE, IIT Delhi
New Delhi, India
jainkartik203@gmail.com

Sayan Ranu
Department of CSE, IIT Delhi
New Delhi, India
sayanranu@cse.iitd.ac.in

Srikanta Bedathur
Department of CSE, IIT Delhi
New Delhi, India
srikanta@cse.iitd.ac.in

Amitabha Bagchi
Department of CSE, IIT Delhi
New Delhi, India
bagchi@cse.iitd.ac.in

ABSTRACT

Regular simple path query (RPQ) is one of the fundamental operators in graph analytics. In an RPQ, the input is a graph, a source node and a regular expression. The goal is to identify all nodes that are connected to the source through a simple path whose label sequence satisfies the given regular expression. The regular expression acts as a formal specification of the search space that is of interest to the user. Although regular expressions have high expressive power, they act as barrier to non-technical users. Furthermore, to fully realize the power of regular expressions, the user must be familiar with the domain of the graph dataset. In this study, we address this bottleneck by bridging RPQs with the *query-by-example* paradigm. More specifically, we ask the user for an exemplar pair that characterizes the paths of interest, and the regular expression is automatically inferred from this exemplar. This novel problem introduces several new challenges. How do we infer the regex? Given that answering RPQs is NP-hard, how do we scale to large graphs? We address these challenges through a unique combination of *Biermann and Feldman's algorithm* with *NFA-guided random walks with restarts*. Extensive experiments on multiple real, million-scale datasets establish that RQuBE is at least 3 orders of magnitude faster than baseline strategies with an average accuracy in excess of 90%.

PVLDB Reference Format:
Komal Chauhan, Kartik Jain, Sayan Ranu, Srikanta Bedathur, and Amitabha Bagchi. Answering Regular Path Queries through Exemplars. PVLDB, 15(2): 299 - 311, 2022.
doi:10.14778/3489496.3489510

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/idea-iitd/RQuBE.

1 INTRODUCTION AND RELATED WORK

Regular simple path query (RPQ) on labelled graphs is one of the core operators in graph analytics. In an RPQ, the input is a source node, a labelled graph, and a regular expression. The goal is to identify all nodes that are reachable from the source via a path whose label sequence conforms to the regular expression.

In this work, we only consider *simple* paths (i.e., acyclic except the relaxation that the source and the destinations nodes may be the same). Simplicity is desired since allowing cycles makes the set of all possible paths infinite, and thus answering both the enumeration and decision versions of RPQs become infeasible. Answering RPQs under path simplicity is NP-hard [23]. Among graph query languages, Cypher from Neo4j only considers simple paths [21]. G-Core restricts the search space to only shortest paths [3] to overcome the computational bottleneck. While shortest paths are simple by definition, it covers a small and restricted subset of all simple paths, and therefore, may return incorrect results. For example, there may be m paths that have an edge larger than the shortest path but all satisfy the regex (but the shortest does not). SPARQL 1.1 uses the notion of *property paths*, which extends RPQ semantics, but allows cycles [33]. Thus, cycles need to be separately handled and a suitable termination condition needs to be enforced (Ex. bounding the number of repeated cycles). Without cycle-detection, property-path evaluation is #P-complete [5]. Similar to SPARQL 1.1, PGQL by Oracle also allows cycles and hence suffers from the same limitations [42]. Recent ISO/IEC SC32/WG3:BNE-023 working document towards standardizing declarative graph query languages, GQL (https://www.gqlstandards.org), offers three different path eligibility modifiers – acyclic, simple, trail (only node repetitions are allowed)– stating that in practice all these modifiers are necessary in different settings. The methodology we propose in this paper is easily extensible to both acyclic and trail. Owing to NP-hardness of (simple) RPQs, several index structures and heuristics have been proposed in the literature [13, 16, 29, 31, 36, 38, 41, 45, 47]. While impressive progress has been made on scalability, a hidden assumption across all algorithms for answering RPQs is that the user is technically knowledgeable enough to formally express a regular expression. This assumption prohibits democratization of RPQs to the general public. Even for technical users, a precise formulation of the regular expression requires prior knowledge of the data set. This requirement acts as a barrier since real networks often contain thousands of labels and millions of nodes and edges. In addition, the meaning of these labels may be opaque. As an example, in the Wikidata knowledge graph, the edges are labeled with tags such as "P127" and "P128" [44]. The user needs to separately investigate the semantic meanings of these labels ("P127" encodes "owned by", while "P128" encodes "regulates" from molecular biology). The goal

Proceedings of the VLDB Endowment, Vol. 15, No. 2 ISSN 2150-8097.
doi:10.14778/3489496.3489510

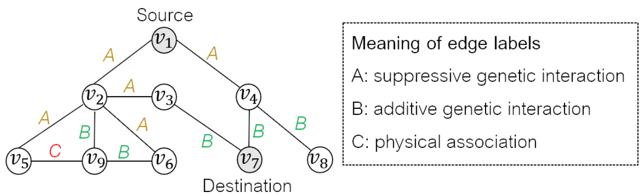


Figure 1: Illustration of regular path queries by example. In this figure, we depict a PPI in the form of a labeled graph [12]

.

of our work is to remove the barrier imposed by regular expressions without compromising on their expressive power. Towards this end, we leverage the *query-by-example* [46] paradigm.

QBE for RPQs: Instead of providing a source node and a regular expression (regex), in the QBE paradigm, the user provides an *exemplar* source-destination pair. The exemplar pair acts as a proxy for the regex constraint and communicates to the query evaluation engine the constraints in a more user-friendly manner. However, to make sense of this query, the query evaluation engine needs to execute the following tasks: **(1)** *Infer* the regular expression that characterizes the paths between the source and the destination, and **(2)** identify nodes that are connected in a similar manner to the source as characterized by the regex inferred from the exemplar.

Let us consider a concrete use-case. Fig. 1 shows a *protein-protein interaction network (PPI)*. Proteins rarely act alone in regulating biological functions. Rather, multiple proteins act jointly to form the *interactome* of an organism [11, 25, 26, 32, 35]. An aberrant protein may induce a *molecular cascade* through the protein pathways and disturb the functioning of other proteins. This, in turn, may affect higher level biological functions that depend on the affected proteins. The PPI encodes these pathways. Within this context, let us assume, a biologist knows that protein v_1 is acting abnormally and is also affecting the functioning of v_7. The biologist now wishes to identify other proteins that are connected in a similar manner to v_1 and, therefore, likely to get similarly affected. To this end the biologist submits (v_1, v_7) as an exemplar pair. There are two simple paths from v_1 to v_7, which are summarized by the regex (A^+B). The query engine should now identify other nodes that are reachable from v_1 via (simple) paths satisfying (A^+B). With this constraint, two other candidates emerge; v_8 and v_9. While all paths to v_8 are accepted by (A^+B), two out of the three paths from v_1 to v_9 satisfy (A^+B). The biologist may then further investigate v_8 and v_9.

In this paper, we enable answering of RPQs via exemplars through a novel technique called RQuBE (**R**egular path **Qu**eries **b**ased on **e**xemplars). Our key contributions are as follows.

- We design a novel mechanism to infer the regex that best fits the paths between the exemplar pair. This goal is achieved by learning a *tight* non-deterministic finite state automaton (*NFA*) that accepts the label sequences in the paths between the exemplar.
- Answering RPQs is NP-hard [23]. To overcome this computational bottleneck, we develop a sampling-based approach to estimate the final answer set. Our sampling algorithm is powered by the novel concept of *NFA-centrality* (§ 3.2). We theoretically characterize the proposed sampling strategy and justify its accuracy (§ 3.3). Owing to a sampling-based strategy, RQuBE is *index-free* and hence easily adapts to updates in the underlying network.
- Extensive experiments across several million-scale, real datasets establish that RQuBE is accurate and at least 3 orders of magnitude faster than baseline strategies (§ 4).

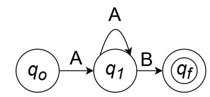


Figure 2: The NFA that accepts A^+B.

Our work is closest in spirit to Bonifati et al. [9], although technically very different. While in RQuBE we focus on deriving a compact regular expression that best describes the set of *all* simple paths between the source and example target node, [9] is designed to work with a set of +ve and -ve example nodes such that there exists *at least one* regex-satisfying path between each positive pair, and no regex satisfying path between each negative pair.

2 PROBLEM FORMULATION

Definition 1 (Graph). *A labeled graph (directed or undirected) is a triple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$, where $\mathcal{V}$ is the set of nodes, $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ is the set of edges, and $\mathcal{L}$ is a finite non-empty set of labels over nodes and/or edges in the graph.*

We use the notations $\mathcal{L}(v_i)$ and $\mathcal{L}(e_i)$ to denote the labels in node v_i and edge e_i respectively.

Computational Model: We assume that graph $\mathcal{G}$ is stored *in memory* in the form of an *adjacency list*. Each node is assigned an ID in the range $[1, |\mathcal{V}|]$. Hence, accessing a specific node in the graph consumes $O(1)$ time. Furthermore, the adjacency list stores the node IDs of each outgoing neighbor and the associated edge label in the form of a *hashmap*. Hence, accessing an outgoing edge or its label also consume $O(1)$ time.

Definition 2 (Path). *A path P in a graph $\mathcal{G}$ is a sequence of distinct vertices $\langle v_1, v_2, \cdots, v_n \rangle$ such that $\forall i, 1 \le i \le n{-}1, (v_i, v_{i+1}) \in \mathcal{E}$. Note that paths with no vertex repeated are also referred to as simple paths. In this paper, we only consider simple paths.*

We use the notation $P.v_i$ and $P.e_i$ to denote the i^{th} node and edge in path P respectively. $P \subseteq P'$ denotes that path P is a sub-path of another path P'. The *label sequence* of a path P is defined as $\mathcal{L}(P) = \langle \mathcal{L}(P.v_1), \mathcal{L}(P.e_1), \mathcal{L}(P.v_2), \cdots, \mathcal{L}(P.v_{k-1}), \mathcal{L}(P.e_{k-1}), \mathcal{L}(P.v_k) \rangle$.

Definition 3 (Regular Expression (Regex)). *Let $\mathcal{L}$ be a set of labels not containing the symbols $\{(,),\emptyset\}$ and let ϵ be a special symbol not in $\mathcal{L}$ denoting the empty string. A regular expression C over $\mathcal{L}$ is defined as follows:*

- *ϵ, $\emptyset$ (the empty set), and each label $l \in \mathcal{L}$ are regular expressions.*
- *If A and B are regular expressions, then $(A|B)$, (AB) and (A^*) are regular expressions.*
- *Nothing else is a regular expression.*

Definition 4 (Non-deterministic Finite State Automaton (NFA)). *An NFA $\mathcal{M}$ is a five-tuple (Q, Σ, f, Q_o, F) where:*

- *Q is a finite non-empty set of states*
- *Σ is a finite non-empty set of input symbols*
- *f is a mapping $Q \times \Sigma \rightarrow 2^Q$ (the transition function)*

- $Q_o \subseteq Q$ *is the set of initial states*
- $F \subseteq Q$ *is the set of final states*

Any *regular expression* C can be converted into an equivalent NFA $\mathcal{M}$ [39]. For example, the NFA of A^+B is shown in Fig. 2.

A label sequence S is *accepted* by $\mathcal{M}$ if we reach a final state in $\mathcal{M}$ through transitions on the labels of S. We use this idea to formally define *path acceptance.*

Definition 5 (Path Acceptance). *A path P is accepted by NFA $\mathcal{M} = (Q, \mathcal{L}, f, Q_o, F)$ if $\mathcal{L}(P)$ is accepted by $\mathcal{M}$. $\mathcal{M}$ accepts label sequence $\mathcal{L}(P) = \langle a_1, \cdots, a_k \rangle$ if there exists a sequence of states $\langle q_0, \cdots, q_k \rangle$ such that:*
(1) $q_0 \in Q_o$
(2) $q_{i+1} \in f(q_i, a_{i+1})$ $\forall i, 0 \le i < k$
(3) $q_k \in F$

Definition 6 (Node pair acceptance). *Let $\mathbb{P}$ be the set of all simple paths from u to v. We say the pair u, v is accepted by $\mathcal{M}$ if every path from u to v is accepted by $\mathcal{M}$, i.e., $A(\mathcal{M}, u, v) = 1$ iff $\forall P \in \mathbb{P}$, $A(\mathcal{M}, P) = 1$.*

As discussed in § 1, there may be multiple NFAs that accept the same set of paths. At this juncture, we assume there is an *oracle* that constructs the *optimal* NFA $\mathcal{M}^*$ with respect to a pair (u, v). The definition and algorithm to construct the optimal NFA will be discussed in § 3.1.

Given any pair of nodes (u, v) and an NFA $\mathcal{M}$, we also define the notions of *support* and *confidence.*

Definition 7 (Support). *Let $\mathbb{P}$ be the set of all simple paths from u to v. The support of (u, v) with respect to an NFA $\mathcal{M}$ is the number of connecting paths accepted by $\mathcal{M}$. Mathematically,*

$$Sup(\mathcal{M}, u, v) = |\{P \mid P \in \mathbb{P}, A(\mathcal{M}, P) = 1\}| \tag{1}$$

Definition 8 (Confidence). *The confidence of (u, v) with respect to $\mathcal{M}$ is the proportion of connecting paths accepted by $\mathcal{M}$.*

$$Conf(\mathcal{M}, u, v) = \frac{Sup(\mathcal{M}, u, v)}{|\ \mathbb{P}\ |} \tag{2}$$

Problem 1 (RPQ based on exemplars (RQuBE)). *Given a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$, an exemplar source-destination node pair (s, t), a support threshold θ, and k, identify the top-k node pairs of the form (s, v), $v \in \mathcal{V}$, with the highest confidence values $conf(\mathcal{M}^*, s, v)$, such that $Sup(\mathcal{M}^*, s, v) \ge \theta$. Here, $\mathcal{M}^*$ denotes the optimal accepting NFA for exemplar pair (s, t).*

3 RQUBE

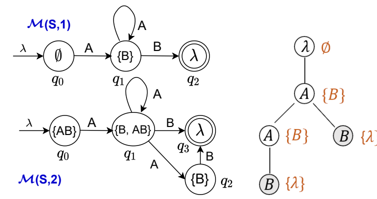
As discussed earlier, RQuBE processes the input in a two-phases: (1) Infer the optimal NFA corresponding to the exemplar and (2) Rank nodes based on their confidence with respect to the inferred NFA. For scalability, we split the second phase into two subphases: 2a) Reduce the search space by identifying promising candidate nodes and 2b) Rank the candidates.

3.1 Phase 1: Inferring the optimal NFA $\mathcal{M}^*$

Let $\mathbb{P}$ be the set of all paths from node s to node t in the input graph $\mathcal{G}$, where (s, t) is the exemplar pair. To construct an NFA that accepts all paths in $\mathbb{P}$, we use the *Biermann and Feldman's (BF)* algorithm [7]. In order to make the paper self-contained we present the BF algorithm here.

Table 1: Illustrates the concept of ℓ-tails with respect to the example depicted in Fig. 1. Here, $S = \{AB, AAB\}$.

Prefix	1-Tail	2-Tail
λ	$\emptyset$	$\{AB\}$
A	$\{B\}$	$\{B, AB\}$
AA	$\{B\}$	$\{B\}$
AB	$\{\lambda\}$	$\{\lambda\}$
AAB	$\{\lambda\}$	$\{\lambda\}$



(a) (b) Trie

Figure 3: (a)$\mathcal{M}(S, 1)$ and $\mathcal{M}(S, 2)$ as formed by BF. (b) Trie constructed for $S = \{AB, AAB\}$ at $\ell = 1$.

The BF algorithm takes a set of finite strings (sequences) $S \subseteq \Sigma^*$ as input, where Σ is the alphabet set. Additionally, it requires an integer parameter $\ell \ge 0$. BF returns an NFA, $\mathcal{M}(S, \ell)$ that accepts all strings in S. When applied to our problem, $S = \{\mathcal{L}(P) \mid P \in \mathbb{P}\}$, and $\Sigma = \mathcal{L}$. From S, BF computes the the set of ℓ-suffixes.

Definition 9 (ℓ-suffix). *The ℓ-suffix, S_ℓ, of S is the set of the suffixes of strings in S that are at most ℓ in length. More formally, $S_\ell = \{w \in \mathcal{L}^* \mid \exists x \in S : w \text{ is a suffix of } x, |w| \le \ell\}$.*

Note that S_ℓ includes the empty string λ.

Example 1. *Revisiting the example in Fig. 1, $S = \{AB, AAB\}$. Hence, $S_1 = \{\lambda, B\}$.*

Let S_p denote the set of all prefixes of strings in S. BF next computes the ℓ-tails of each string in S_p.

Definition 10 (ℓ-tail). *For any string $x \in \mathcal{L}^*$, its ℓ-tail w.r.t. S, denoted $tail(x, S, \ell)$, is the set of strings $w \in S_\ell$ such that $x \oplus w \in S$. $\oplus$ denotes the concatenation of two strings.*

Note that, for each $x \in S_p$, $tail(x, S, \ell) \subseteq S_\ell$. Further, $tail(x, S, \ell)$ could also be the empty set.

Example 2. *The 1-tail and 2-tail of each prefix corresponding to $S = \{AB, AAB\}$ of Fig. 1 is shown in Table 1.*

We are now ready to define $\mathcal{M}(S, \ell)$ on alphabet $\mathcal{L}$.

(1) *The set of states:* $Q = \{tail(x, S, \ell) \mid x \in S_p\}$ is the set of ℓ-tails of the prefixes of the strings in S. Note that for a $q \in Q$, there may be more than one x whose ℓ-tail is q.
(2) *Transitions:* For a symbol $\alpha \in \Sigma$, we place a transition between two states q and q' if there is a string w such that q is the ℓ-tail of w and q' is the ℓ-tail of $w \oplus \alpha$.
(3) *Initial state:* The initial state is $q_o = tail(\lambda, S, \ell)$, i.e., the ℓ-tail of the empty string.
(4) *Final states:* $F = \{q \in Q \mid \lambda \in q\}$, i.e., a state q is final if it contains the empty string λ.

Example 3. *We will continue from Ex. 2. Fig. 3a shows the structures of $\mathcal{M}(S, 1)$ and $\mathcal{M}(S, 2)$. As visible in Table 1, there are three unique 1-tails and hence, we have three states in $\mathcal{M}(S, 1)$. Note that $\mathcal{M}(S, 1)$ is identical to Fig. 2 and accepts the regex A^+B.*

A rigorous proof of correctness is available in [7].

Theorem 1 ([7]). *The language accepted by $\mathcal{M}(S, \ell+1)$ is a subset of the language accepted by $\mathcal{M}(S, \ell)$.*

Theorem 2 ([7]). *If we set $\ell \geq \max_{\forall s \in S} |s|$, i.e., the length of the longest string in S, then $\mathcal{M}(S, \ell)$ accepts exactly S. On the other hand, $\ell = 0$ degenerates to the regex $\mathcal{L}^*$, i.e., it accepts all strings.*

The following corollary follows from Thm. 2.

Corollary 1. *Let $S_{\mathcal{M}(S,\ell)} = \{w \in \Sigma^* \mid A(\mathcal{M}(S, \ell), w) = 1\}$ be the set of all strings accepted by $\mathcal{M}(S, \ell)$. If $\ell \leq \max_{\forall s \in S} |s|$, then $S_{\mathcal{M}(S,\ell)} \supseteq S$.*

More simply, ℓ has an effect that is similar to a *regularizer* in supervised model learning. When ℓ is high, it may learn an NFA that *overfits* the paths between the exemplar pair. On the other hand, a low value of ℓ may be too generic that accepts almost all paths in $\mathcal{L}^*$. Therefore, to learn the *optimal* value of ℓ, and in turn build the optimal NFA, we borrow the *cross-validation* paradigm.

We first partition the set of all paths $\mathbb{P}$ between the exemplar into f *folds*. $f-1$ of these folds are used as the train set. The held-out fold is used for *validation*. We use the notations S_T and S_V to denote the strings in the train and validation set respectively. We first construct $\mathcal{M}(S_T, \ell)$ for each $\ell \in [1, \ell_{max}]$, where $\ell_{max} = \max_{\forall s \in S_T} |s|$. The optimal ℓ is defined as:

$$\ell^* = \arg \max_{\ell \in [1, \ell_{max}]} \left\{ \frac{|\{s \in S_V \mid A(\mathcal{M}(S_T, \ell), s) = 1\}|}{|S_V|} \geq \omega \right\} \quad (3)$$

Here, $\omega \in [0, 1]$ is a hyper-parameter. More simply, we identify the largest ℓ such that proportion of strings accepted from the validation set is at least ω. The rationale behind this choice stems from the property that as ℓ increases, the likelihood of a validation string being accepted reduces. We, however, need to avoid being too generic by lowering ℓ. Hence, we choose the most discriminative ℓ that achieves an acceptable level of quality on the validation set.

As typical in cross-validation, the above process is repeated taking each fold as the validation set. We next compute the mean ℓ^* across all folds. The optimal NFA $\mathcal{M}^*$ is finally set to $\mathcal{M}^* = \mathcal{M}\left(\mathbb{P}, floor\left(\ell^*_{mean}\right)\right)$.

3.1.1 BF Implementation. We maintain a *hashmap* T whose keys are the strings of S_p. The entries in the hashmap are their ℓ-tails. We note that the size of S_ℓ is at most $\sum_{i=0}^{\ell} |\mathcal{L}|^\ell \leq |\mathcal{L}|^{\ell+1} = \theta(|\mathcal{L}|^\ell)$.

(1) *Pre-processing*: We store all the strings of S in a *trie* $\mathcal{S}$. Each node in this trie represents a prefix of some string(s) of S, so the size of this trie is at most $|S_p|$. In case a complete string of S is stored in an internal node of $\mathcal{S}$, we associate an "End-of-word" (EoW) tag with it.
(2) *Constructing the ℓ-tails*: We traverse $\mathcal{S}$ bottom-up computing ℓ-tails as follows:
    (a) For each string x corresponding to a leaf of $\mathcal{S}$ we set its ℓ-tail to $\{\lambda\}$ and update hashmap $T(x)$ accordingly.
    (b) If x is a string corresponding to an internal node, we form its ℓ-tail from the ℓ-tails of its children. Specifically, let $x \oplus \alpha$ be a child of x, where $\alpha \in \mathcal{L}$. Now, if w is in the ℓ-tail of $x \oplus \alpha$ and $|w| < \ell$, then we put $\alpha \oplus w$ into the ℓ-tail of x. Further, if the node storing x has an EoW tag, we add λ to its ℓ-tail. Once the ℓ-tail is computed we update $T(x)$.

  We prove the correctness of this step in Proposition 1.
(3) *Adding the transitions*: For each $y \in S_p$ if $\exists y \oplus \alpha \in S_p$, we place a transition labeled α from $T(y)$ to $T(y \oplus \alpha)$. If this transition already exists, we do nothing.
(4) *Initial and final states*: We set the state $T(\lambda)$ as the initial state. For each $x \in S$ (i.e. each leaf of $\mathcal{S}$ or node with an EoW tag) we set $T(x)$ as a final state.

Example 4. *Fig. 3b shows the trie created for set $S = \{AB, AAB\}$ at $\ell = 1$. The shaded nodes have their EoW tag set to 1 and the ℓ-tail of every node is written outside in brackets. Note that at the root node of $\mathcal{S}$, the prefix is λ.*

Proposition 1. *Step (2) above correctly computes the ℓ-tails of all the prefixes of S_p from $\mathcal{S}$.*

Proof. By the property of tries any string x represented by a leaf of $\mathcal{S}$ is clearly a string of S. Hence, λ must be part of its ℓ-tail. Let us assume there is some other non-empty string $w = \alpha_1 \cdots \alpha_\ell$ that is also part of the ℓ-tail of x. Then $x \oplus w \in S$ and hence the node containing x in $\mathcal{S}$ must have at least one child corresponding to symbol α_1, which contradicts the assumption that it is a leaf.

Now let us consider an internal node corresponding to string $x \in S_p$. There exists at least one symbol $\alpha \in \mathcal{L}$ such that $\mathcal{S}$ contains a node corresponding to $x \oplus \alpha$. Now, if w is in the ℓ-tail of $x \oplus \alpha$ then $x \oplus \alpha \oplus w \in S$. $|w| < \ell$ implies that $|\alpha \oplus w| \leq \ell$ and so $\alpha \oplus w$ is part of the ℓ-tail of x. This argument holds for all symbols α such that $x \oplus \alpha \in S_p$. Further, if $x \in S$ then this node contains an EoW tag and we place λ in x's ℓ-tail. □

3.1.2 Optimizations. We next discuss some optimization strategies.
Computing optimal ℓ^*: From Thm. 1, we know that with lowering of ℓ, the acceptance likelihood of validation strings monotonically increases. Thus, instead of pre-computing the NFA for each value of $\ell \in [1, \ell_{max}]$, we perform a *binary search* starting from ℓ_{max}. Consequently, we compute $O(f \log(\ell_{max}))$ NFAs to identify the optimal one in f-fold cross-validation.
Path enumeration: The number of paths between two nodes can be exponential and hence not tractable. Furthermore, as established in Cor. 1, the inferred NFA generalizes to more paths (strings) than the enumerated set. Hence, enumerating all paths may not even be necessary. To obtain a good balance between computational tractability and path coverage, we enumerate all paths of length till the diameter of the graph. Consequently, given any exemplar pair that is reachable, we enumerate at least one path between them.

3.1.3 Complexity Analysis. To derive the complexity, we divide it into two sub-tasks: (1) *Enumerating paths S* from source to destination, and (2) *NFA construction* over S.
Path Enumeration: As mentioned above, we enumerate paths of length up to the diameter of the graph. Accordingly, $|S|$ is bounded by $d_{\mathcal{G}}^{diam}$, where $diam$ denotes the diameter and $d_{\mathcal{G}}$ is the maximum degree of a node in $\mathcal{G}$. Many real-world networks display small-world properties [37], under which $diam = \theta(\log(|\mathcal{V}|))$ [27]. Thus, $|S| \leq d_{\mathcal{G}}^{\theta(\log(|V|))}$. Owing to constant time edge access in our computational model (Recall from § 2), the complexity reduces to $O(diam|S|) = O\left(\log(|V|) d_{\mathcal{G}}^{\log(|V|)}\right)$.
NFA Construction: Constructing the trie $\mathcal{S}$ takes time $|S_p|$. The time taken to process each node of the trie is $\theta\left(|\mathcal{L}|^\ell\right)$ since the

Algorithm 1 Baseline

Require Graph $\mathcal{G}(\mathcal{E}, \mathcal{V}, \mathcal{L})$, exemplar pair (s, t), k, θ
1: $\mathcal{M}^* \leftarrow$ optimal NFA for (s, t) constructed using BF algorithm
2: $\mathbb{PQ} \leftarrow$ Priority Queue
3: **for** each node $v \in \mathcal{V} \setminus \{t\}$ **do**
4: $\quad \mathbb{P} \leftarrow$ all paths from s to v
5: $\quad$ **if** $Sup(\mathcal{M}^*, s, v) \geq \theta$ **then**
6: $\quad\quad c \leftarrow Conf(\mathcal{M}^*, s, v)$
7: $\quad\quad$ **if** $c > PQ.top()$ or $PQ.size() < k$ **then**
8: $\quad\quad\quad PQ.insert(\langle v, c \rangle)$
9: $\quad\quad$ **if** $PQ.size() > k$ **then**
10: $\quad\quad\quad PQ.pop()$
11: **return** $\mathbb{PQ}$

ℓ-tail of each child can have size at most ℓ and there are at most $|\mathcal{L}|$ children. Hence the total time for ℓ-tail construction is $\theta\left(|S_p||\mathcal{L}|^{\ell}\right)$ since the number of nodes in $\mathcal{S}$ is at most $|S_p|$. Computing the transitions can be done along with the process of building the ℓ-tails and contributes another $\theta\left(|S_p||\mathcal{L}|\right)$ steps. Similarly, the initial and final states can be marked along with the ℓ-tail construction and contribute only one extra check per node of $\mathcal{S}$. Overall the time complexity of the algorithm is bounded by $\theta\left(|S_p||\mathcal{L}|^{\ell_{\max}}\right)$, which is $O\left(\ell_{\max}|S||\mathcal{L}|^{\ell_{\max}}\right)$. Since, $\ell_{\max} \leq diam \leq \theta(\log(|V|))$ and $|S| \leq d_{\mathcal{G}}^{\theta(\log(|V|))}$, the complexity is $O\left(\log(|V|) d_{\mathcal{G}}^{\log(|V|)} |\mathcal{L}|^{\log(|V|)}\right)$.

Overall time complexity: Combining both factors, and under the assumption of constant label set size and bounded degree, the complexity is $O(\log(|V|)C^{\log(|V|)})$, where C is a constant. Thus, the complexity is *polynomial* with $|V|$ in small-world graphs.

Space Complexity: The space complexity is bounded by the size of the trie, which is $O(|S_p|) = O(|S|) = O\left(d_{\mathcal{G}}^{\log(|V|)}\right)$.

3.1.4 Discussion: Other algorithms for NFA construction. There are a number of different approaches to constructing an NFA to fit a set of strings [4, 7, 10, 15, 17, 18, 24, 34, 43]. Several algorithms require negative examples [4, 15, 17, 28]. Hence, they are not relevant since we only have positive examples, i.e., strings that we want recognised. Among algorithms that only work with positive examples, the general approach is to begin by exhaustively building an NFA, possibly with a large number of states, that recognizes all the positive examples. In the next phase, this NFA is reduced by merging states that are *equivalent* in some way [7, 18, 24, 43]. Typically, both phases are merged in the actual algorithm. The BF algorithm we use [7] is a prominent example of this approach. A second class of algorithms merge states with a view to minimize an information theoretic criterion, the *minimum message length (MML)*, which trades off the size of the NFA with the encoding length of the positive examples given an NFA [10, 30, 34].

In theory, any of the NFA-inference algorithms that work only with positive samples can be used. We choose BF due to two reasons: **(1)** The class of algorithms based on MML is computationally expensive since we need to compute the change in MML before making each merging decision. **(2)** MML-based algorithms do not allow a supervised mode of NFA-construction, wherein the balance between generalizability and specificity can be controlled (As in § 3.1.2). This may affect the quality of the inference task.

3.2 Phase 2a: Reducing the Search Space

Once the query NFA $\mathcal{M}^*$ is inferred from the exemplar, our next task is to identify the top-k nodes based on $Conf(\mathcal{M}^*, s, v)$ as long as $Sup(\mathcal{M}^*, s, v) \geq \theta$.

3.2.1 Baseline and bottlenecks. Alg. 1 outlines the pseudocode. The most expensive step in Alg. 1 lies in the path enumeration computation (line 4). The number of paths in a graph can be exponential with respect to the size of the node set and this property lies at the core of why answering RPQs is NP-hard [23]. We overcome this scalability bottleneck through sampling.

(1) Identify a set of candidate nodes $C \subseteq \mathcal{V}$, such that with a high likelihood the answer set $\mathcal{A} \subseteq C$ and $|C| \ll |\mathcal{V}|$.
(2) For each node $v \in C$, sample a set of paths $\mathbb{P}'$ from s to v such that the confidence computed from $\mathbb{P}'$ is close to the true confidence, i.e., $Conf_{\mathbb{P}'}(\mathcal{M}, s, v) \approx Conf(\mathcal{M}, s, v)$. Note that $|\mathbb{P}'|$ must be at least as large as the support threshold θ.

3.2.2 Motivating our sampling strategy. To streamline our search process, we first note the following:

- **Scale-free graphs**: Many real-world graphs display the *scale-free* property [2]. In a scale-free graph, the degree distribution follows a power law. Consequently, these graphs contain a small number of "hub" nodes that have high degree and act as bridges between large number of nodes. Consequently, if a hub cannot be part of an NFA-accepting path, then nodes that are only reachable through this hub cannot be part of the answer set.
- **Homophily**: Real graphs often display homophily [22]. In homophily, neighboring nodes/edges have a higher likelihood of sharing labels than nodes further away. Thus, nodes that are part of the answer set may be clustered in similar regions of the graph. Consequently, limiting the search process to *only* these clusters may provide a good accuracy-efficiency trade-off.

We take inspiration from the above observations and formalize the idea through *NFA-guided random walks with restarts*.

3.2.3 NFA-guided Random Walks with Restarts (NRWR). Given graph $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{L})$ and NFA $\mathcal{M}^* = (Q, \Sigma, f, \{q_0\}, F)$ with $\Sigma \subseteq \mathcal{L}$, we define the product space $\mathcal{V} \times Q$. In this space, the random walk unfolds in discrete time steps. At $t = 0$, the walk starts from the initial state $(v, q) = (s, q_0)$. A transition from (v, q) to (v', q') is allowed if:

(1) $(v, v') \in \mathcal{E}$
(2) One of the below conditions are met depending on whether the graph is node labeled, edge labeled, or both.
    (i) Edge labelled case: $\mathcal{L}((v, v')) = \alpha$ and $q' \in f(q, \alpha)$
    (ii) Node labelled case: $\mathcal{L}(v') = \alpha$ and $q' \in f(q, \alpha)$.

In addition, a transition is also allowed from (v, q) to (s, q_0) if

(1) $q \in F$, i.e., q is an accepting state.
(2) $\nexists(v', q'),\ q' \notin F$, such that (v, q) to (v', q') is an allowed transition as per the rules defined above. In other words, (v, q) is a dead-end. Dead-ends occur when either v has no outgoing edge, or there are no outgoing labels from v that is allows a valid state transition in $\mathcal{M}^*$.

NRWR is a Markov Chain with transition matrix $\mathbf{W}$, where

$$\mathbf{W}\left[(v, q) \rightarrow (v', q')\right] = \begin{cases} \frac{1}{deg((v,q))} & \text{if the transition is allowed} \\ 0, & otherwise \end{cases}$$

Here, $deg((v,q))$ represents the total number of allowed transitions from (v,q). In addition to the transitions allowed by $\mathbf{W}$, the walker may decide to return to the initial state (s,q_0) with probability β from its current state. The t-step probability vector of NRWR is $\boldsymbol{\pi}_t$, i.e., $\boldsymbol{\pi}_t(v,q)$ denotes the probability of the walker being at state (v,q) at time t. Mathematically, $\boldsymbol{\pi}_t = (1-\beta)\mathbf{W}\boldsymbol{\pi}_{t-1} + \beta\boldsymbol{\pi}_0$. In our case, we will always begin NRWR from one particular state, i.e., we will always have $\boldsymbol{\pi}_0[(s,q_0)] = 1$.

Proposition 2. *NRWR begun from (s,q_0) has a unique stationary distribution $\boldsymbol{\pi}$ and $\lim_{t\to\infty}\boldsymbol{\pi}_t = \boldsymbol{\pi}$. Further, if $S_r \subseteq \mathcal{V}\times Q$ is the set of states reachable from (s,q_0) then $\boldsymbol{\pi}$ assigns non-zero probability to all $(u,q)\in S_r$ and 0 probability to all other states.*

Proof. Since we only begin NRWR from (s,q_0) we consider the restriction of the Markov Chain to the state space S_r.

- **Irreducibility:** Given any two states $(v,q),(v',q') \in S_r$, we know (s,q_0) is reachable from (v,q) through restart. Furthermore, we also know that (v',q') is reachable from (s,q_0) since $(v',q') \in S_r$. Hence, (v',q') is reachable from (v,q).
- **Aperiodicity:** The restart acts as a self-loop on the initial state (s,q_0), i.e., this state has period 1. Since all states of an irreducible Markov Chain have the same period [20, Lemma 1.6] the Markov chain is aperiodic.

By the definition of S_r it is clear that the limiting probability for any state *not* in S_r is 0. The fact that each state of S_r has non-zero probability is a well-known fact about finite Markov Chains and can be established by combining Lemma 1.13 and Prop. 1.19 of [20]. □

The query-specific *hub score* of a node is quantified through *NFA-guided centrality.*

Definition 11 (NFA-Guided Centrality). *The NFA-guided centrality relative to s and NFA $\mathcal{M}^*$ is a $|\mathcal{V}|\times 1$ vector $\boldsymbol{\phi}$ where $\forall v\in\mathcal{V}$, $\boldsymbol{\phi}(v) = \sum_{q\in Q}\boldsymbol{\pi}(v,q)$.*

Corollary 2. *$\boldsymbol{\phi}$ is well-defined and unique.*

Proof. $\boldsymbol{\phi}$ is an aggregation over $\boldsymbol{\pi}$. Since $\boldsymbol{\pi}$ exists, and is unique, so is $\boldsymbol{\phi}$. □

3.2.4 Identifying the candidate set C. Intuitively, $\boldsymbol{\phi}(v)$ quantifies the proportion of NFA-guided paths that go through node v. Thus, if v has a low $\boldsymbol{\phi}(v)$, it is unlikely that nodes that are *only* NFA-reachable through v will be part of the answer set. Te eliminate such unlikely nodes we first sort nodes based on $\boldsymbol{\phi}$. We next select the highest centrality nodes H (except the source node) by iteratively adding nodes from the sorted list till the *marginal increase* in the cumulative distribution saturates. Mathematically,

$$\left(\sum_{\forall v\in H\cup\{v'\}}\boldsymbol{\phi}(v) - \sum_{\forall v\in H}\boldsymbol{\phi}(v)\right) < \eta \tag{4}$$

Here, $\eta\approx 0$ is a small threshold and v' is the $|H|+1$ ranked node in $\boldsymbol{\phi}$. Thus, H contains the hub nodes and their neighborhoods represent the regions containing answer set candidates. We extract these candidates C as follows.

$$C = \{v\in\mathcal{V} \mid \exists v'\in H, sp(v',v)\le d\} \tag{5}$$

Here, $sp(v',v)$ denotes the shortest path distance from v' to v, and d is a distance threshold.

3.2.5 Implementation and Computation Complexity: We break-up the analysis into the following components.

NRWR: We simulate NRWR on the state space and terminate once the set of hub nodes H stabilizes. In our implementation, we compute each state and outgoing transitions on the fly since only a minority of the state space is actually visited during the NRWR. To elaborate, at any state, we first compute the outgoing states. This consumes $O(d_{\mathcal{G}})$ time where $d_{\mathcal{G}}$ is the maximum outgoing degree in $\mathcal{G}$. Selecting the next transition consumes $O(1)$ time since it is chosen uniformly at random if outgoing degree of the current state is above 1; otherwise, the walker returns to (s,q_0). This process continues for T iterations requiring $O\left(Td_{\mathcal{G}}\right)$ time. In our implementation we do not fix T, stopping when H has converged. Fogaras et. al. [14] show that the probability of rankings getting interchanged decays exponentially with T, so after $T=\theta(\log 1/\varepsilon)$ we expect H to stabilize with probability at least $1-\varepsilon$.

Extracting hub nodes H**:** We stop NRWR when H converges, i.e., it remains unchanged for two iterations. To keep nodes sorted based on $\boldsymbol{\phi}$, we use an array and after each transition the count and the position of the visited node is updated. Using binary search, this consumes $O(\log(\min\{|\mathcal{V}|,T\}))$ time since at most T vertices can be visited in T steps.

Computing candidate set C**:** We do breadth-first search for d hops from H. This consumes $O\left(|H|d_{\mathcal{G}}^{d}\right)$ time.

Time and space complexity: We only store $\boldsymbol{\phi}$ during NRWR and hence space complexity is $O(|\mathcal{V}|)$. By aggregating the time complexity from each of the above components, the final complexity is $O\left(T\left(d_{\mathcal{G}}\log(\min\{|\mathcal{V}|,T\})\right)+|H|d^{d_{\mathcal{G}}}\right)$ where $T=\theta(\log 1/\varepsilon)$ for a success probability of at least $1-\varepsilon$. The probability of convergence increases exponentially with T. Hence, T is a small number. d is typically 2 or 3. Consequently, under the assumption that the degree is bounded, $O\left(T\left(d_{\mathcal{G}}\log(\min\{|\mathcal{V}|,T\})\right)+|H|d^{d_{\mathcal{G}}}\right)\approx O(|H|$.

3.3 Phase 2b: Ranking the Candidates

Once C is computed, our task is to rank these nodes based on their confidence under the constraint that a node is reachable from the source by at least θ NFA-accepting paths. Since enumeration of all possible paths to each node in C is worst-case exponential, we sample paths. The higher the sample size, the more accurate would be our confidence estimates. On the other hand, a higher sample size leads to increased computation cost. To fine-tune quality-efficiency balance, we propose an *iterative sampling* methodology (Alg. 2).

In addition, to the input parameters, Alg. 2, takes two additional hyper-parameters: *sample size z* and *pruning percentage p*. Alg. 2 proceeds in an iterative manner. In each iteration, z additional paths are sampled for current candidate set (line 7). With these additional samples, a more accurate estimate of the confidence values are computed (lines 8-9). Based on the updated confidence values, the top-k nodes are recomputed. If the top-k set remains unchanged then the algorithm terminates (line 4). Otherwise, we eliminate the bottom-p% of the candidate set, and proceed towards the next iteration with the smaller candidate set (lines 10-11).

3.3.1 Theoretical Characterization. The proposed sampling strategy is based on the hypotheses that nodes that have a high fraction

Algorithm 2 Ranking Nodes through Iterative Sampling

Require Graph $\mathcal{G}(\mathcal{V}, \mathcal{E}, \mathcal{L})$, $\mathcal{M}^*$, k,support threshold θ, sample size z, retain percentage p, candidate set C
Returns Set of top-k ranked nodes

```
1: t ← 0
2: C_t ← C
3: ∀v ∈ C, P_t^v ← ∅
4: while t ≤ 1 or top-k(C_{t−1}) ≠ top-k(C_t) do
5:     t ← t + 1
6:     for each node v in C_{t−1} do
7:         P ← sample z paths from s to v.
8:         P_t^v ← P_{t−1}^v ∪ P
9:         Conf(v) ← Compute confidence based on P_t^v
10:    B ← bottom top-p% nodes of C_{t−1}
11:    C_t ← C_{t−1} \ B
12:    if |C_t| ≤ k then
13:        return top-k(C_{t−1})
14: return top-k(C_t)^1
```

of paths accepted by the NFA maintain that high fraction even in a small sample. This claim is next made rigorous.

Proposition 3. *Given a target node such that the number of paths from the source to the target is n and the fraction of these paths accepted by the NFA is f, then, for any* $\delta \in [0, 1]$, *a sample of z independently chosen paths will have more than* $(1 - \delta) f z$ *paths accepted by the NFA with probability at least* $1 - 1/n^c$ *for some constant* $c > 0$ *whenever* $z = \theta((\log n)/f\delta)$.

Proof. For $1 \leq i \leq z$, let X_i be an indicator random variable that takes value 1 if the i^{th} path sampled is a path accepted by the NFA. Then $X = \sum_{i=1}^{z} X_i$ is the number of accepted paths and X/z is the fraction of accepted paths. Note that $\mathrm{E}[X/z] = f$. Assuming the X_i are independent, applying a Chernoff bound for the lower tail we have $\mathrm{P}[X < (1 - \delta) f z] \leq e^{-\delta^2 f z/2}$. Choosing $z = \theta((\log n)/f\delta)$ gives us the result. □

From Prop. 3, it is clear that as the iterations proceed, the probability of a high ranked node getting pruned falls exponentially. We also show that for moderate choices of the parameter z, lower ranked nodes cannot overtake higher ranked nodes.

Proposition 4. *Suppose we have two node v and u such that the fractions of the paths to these nodes that are accepted by the NFA are* f_v *and* f_u *respectively. If* $f_v > f_u$ *then, for any* $\varepsilon > 0$, *with probability* $1 - \varepsilon$, *the empirical fraction of the accepted paths of v is greater than empirical fraction of the accepted paths of u after z independently drawn samples from each set whenever z is* $\theta\left((\log 1/\varepsilon)/(f_v - f_u)^2\right)$.

Proof. For $1 \leq i \leq z$ let $X_i = -1$ if the i^{th} path for node u is accepted but the i^{th} path for node v is not, let $X_i = 1$ if the situation is reversed and let $X_i = 0$ if the i^{th} paths for both nodes are both rejected or both accepted. If $X = \sum_{i=1}^{z} X_i$, the event $X < 0$ corresponds to the event that the rankings are reversed. $\mathrm{Var}[X_i] = f_u(1 - f_v) + f_v(1 - f_u) = f_u + f_v - 2 f_u f_v$. By Bernstein's inequality:

$$\mathrm{P}[X < 0] \leq \exp\left\{\frac{-((f_v - f_u)z)^2}{(f_u + f_v - 2 f_u f_v + 1/3)z}\right\}.$$

Upper bounding the denominator of the exponent by 3 we get

$$\mathrm{P}[X < 0] \leq e^{-(f_v - f_u)^2 z/3}.$$

□

Time Complexity	$O\left(\log(\lvert\mathcal{V}\rvert)\, C^{\log(\mathcal{V})}\right)$	$O(\lvert H\rvert)$	$O(\lvert\mathcal{C}\rvert(z\log(\mathcal{V}) + \log(\lvert\mathcal{C}\rvert)))$
Space Complexity	$O\left(d_{\mathcal{G}}^{\log(\mathcal{V})}\right)$	$O(\lvert\mathcal{V}\rvert)$	$O(\lvert\mathcal{C}\rvert)$

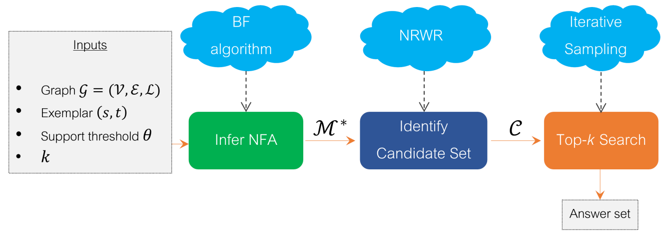


Figure 4: Flowchart of RQuBE.

3.3.2 *Computational Complexity:*

In the worst case, there will be at most $\frac{|C|}{|C| \times p/100} = \frac{100}{p}$ iterations. Sampling z paths of length *diam* consumes $O(z \cdot diam) \approx O(z \log(|V|))$ time (in small-world graphs (Recall § 3.1.3)). Updating the confidence of a node in each iteration consumes $O(z)$ time. Removing the bottom-p% and computing top-k set requires $O(|C| \log\left(\frac{p|C|}{100}\right)$ and $O(|C| \log(k))$ time respectively using min-heap and max-heap respectively. Thus, the total time consumption is $O\left(\frac{1}{p}|C|\,(z\log(|V|) + \log(p|C|k))\right) \approx O(|C|\,(z\log(|V|) + \log(|C|)))$ since both $1/p$ and k are small.

Regarding space complexity, we only store two heaps to track top-k and bottom-p% nodes in the candidate set. Thus, the space consumption is $O(kp|C|) \approx O(|C|)$. Note that although for ease of presentation we denote every sampled path being stored (line 8 in Alg. 2), in our implementation we do not require that since $Conf_t(v) = \frac{sup_{t-1}(v) + sup_t(v)}{\text{no. of paths sampled till now}}$.

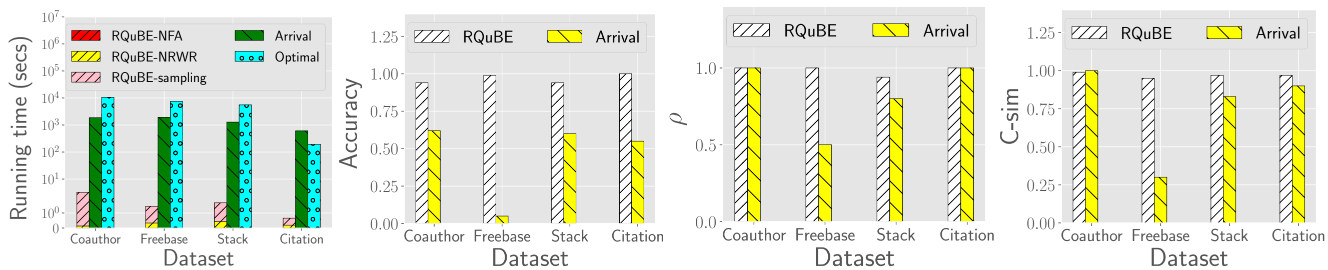
3.4 Putting it All Together

Fig. 4 outlines the flowchart of RQuBE. Given the input information, first, we infer the optimal NFA $\mathcal{M}^*$(§ 3.1). Next, we use $\mathcal{M}^*$ to perform NFA-guided random walks with restart (NRWR) on $\mathcal{G}$ to identify the candidate set C. C is constructed by identifying the high centrality nodes in NRWR and then extracting their d-hop neighborhoods. These candidates are then ranked through iterative sampling of paths and the k nodes with highest confidence are returned as the answer set.

3.4.1 *Complexity Analysis:*

In Fig. 4, we list the time and space complexities of each of the components (See § 3.1.3,§ 3.2.5, and § 3.3.2 for the derivations). Recall, C is a constant, $|H|$ is the number of hub nodes, C is the number of nodes within d-hops from H, z is the sample size in Top-k search, and $d_{\mathcal{G}}$ is the maximum degree in $\mathcal{G}$. Since $H \subseteq C$, the total time complexity is therefore $O(\log(|\mathcal{V}|)C^{\log(|\mathcal{V}|)} + |C|(z\log(|V|) + \log(|C|)))$. It is difficult to bound $|C|$ in general. However, for real-world networks typically a small number of hubs of the graph capture most of the mass of any centrality metric. Thus, we expect $|H|$ to be moderate and hence, C to be moderate as well. In § 4.2, we empirically profile the time consumed by each of the three phases. Our experiments reveal that iterative sampling for Top-k search is the most expensive phase and NFA-inference is the fastest.

From Fig. 4, the total space complexity is $O(d_{\mathcal{G}}^{\log(|\mathcal{V}|)} + |\mathcal{V}|)$.



(a) Running time (b) Accuracy (c) Rank correlation (d) Confidence similarity

Figure 5: (a) Running time comparison of RQuBE with Optimal and Arrival. (b-d) Accuracy, rank correlation, and confidence similarity of RQuBE and Arrival when compared to the ground-truth.

4 EXPERIMENTS

In this section, we benchmark RQuBE and establish that:

- **Quality:** The proposed sampling methodologies are effective and retains an average accuracy above 0.90.
- **Efficiency:** NFA-guided centrality is an effective measure to guide the search process towards promising regions and enables scalability to million-sized graphs containing thousands of labels.
- **Semantics:** The proposed NFA inference mechanism returns semantically meaningful results. Furthermore, the inferred NFA enables visualization of relationship diffusion in the graph.

4.1 Experimental Setup

All experiments are performed on a machine with Intel(R) Xeon(R) Platinum CPU 2.1GHz with 256 GB RAM running Ubuntu 18.04. All implementations are done in C++. Each experiment reports the mean of the metric being measured over 100 queries. For each query, the exemplar source and the destination are chosen uniformly at random from all nodes in the graph. The default parameter values are $k = 10$ and support threshold $\theta = 30\%$ of the total number of paths between the exemplar pair.

4.1.1 Datasets. Table 2 summarizes the datasets.

`Co-author` [40]: `Co-author` is a co-authorship network obtained from https://dblp.org/. Two authors (nodes) have an edge between them if they have collaborated in at least one paper. Each node label represents the venue where the author has published the most.

`Citation` [40]: In `Citation`, each node is a paper, and there is a edge from paper x to paper y, if x cites y. The label of a node is the conference venue where it was published.

`Freebase` [8]: `Freebase` is an open-source knowledge graph. Node labels in freebase indicate the entity type such as 'person', 'athlete', 'artist', etc. Each edge represents a semantic link between the entities with labels such as 'created', 'lives in', etc.

`StackOverflow` [19]: `StackOverflow` is a *dynamic* graph containing interactions on the communication forum of Stack Overflow. Each directed edge (u, v, t), denotes an interaction between users (nodes) u and v at time t. The label of an edge corresponds to its communication type. Since this dataset is dynamic, in addition to the query parameter, we also include a timestamp t_q. t_q is randomly sampled from the time range in `StackOverflow`. The query is answered with respect to the state of the graph at time t_q.

Table 2: Summary of datasets used.

Dataset	$\|\mathcal{V}\|$	$\|\mathcal{E}\|$	$\|\mathcal{L}\|$	Directed	Node Labels	Edge Labels	Effective Diameter
Co-author (DBLP)	$1.75M$	$13.6M$	17073		✓		6.7
Citation (DBLP)	$3.1M$	$8.4M$	11661	✓	✓		7.6
Freebase	$3.6M$	$57.7M$	7513	✓	✓		3.7
StackOverflow	$2.6M$	$67.5M$	3	✓		✓	4.0
Yeast	$4k$	$79k$	41		✓		4.0

`Yeast`: `Yeast` represents the protein-protein interaction network of the yeast organism. Each node is tagged with the gene ontology of the corresponding protein [6], which indicates its function.

4.1.2 Baselines. We consider the following baselines.

- **Optimal:** Optimal provides us the ground truth by implementing Alg. 1. Since Optimal is exorbitantly slow, in all experiments involving Optimal, we limit the search process to nodes that are at most diameter hops away from the source.
- **Arrival [45]:** Arrival is the state of the art for answering RPQs. For our problem, instead of enumerating all paths through Optimal, we sample paths through Arrival.

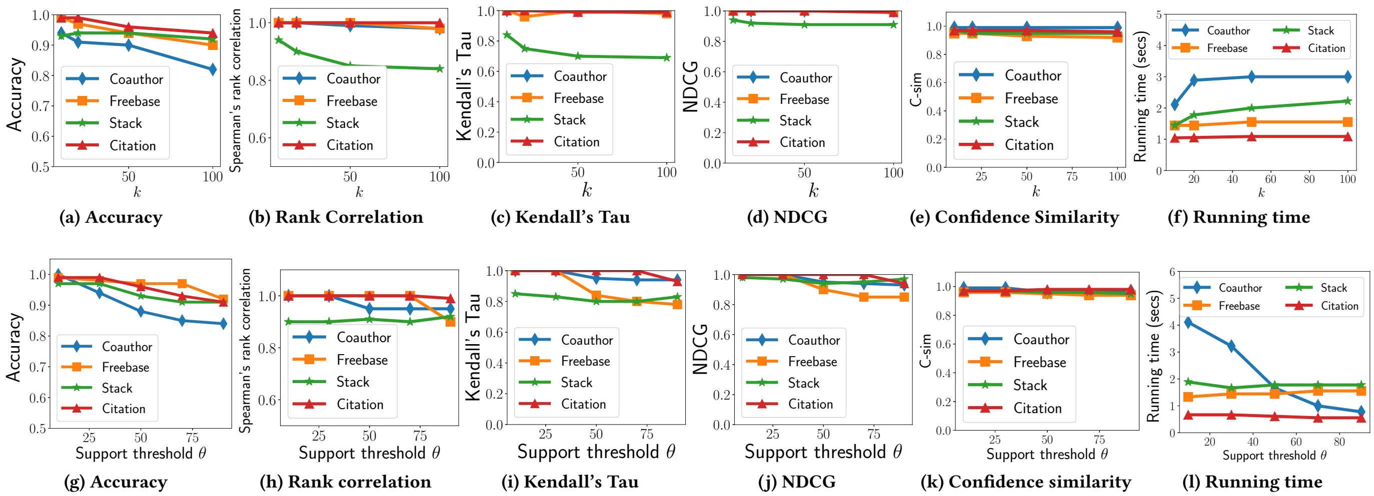
4.1.3 Hyper-parameters. The hyper-parameter values of RQuBE are identified through *grid-search* on 20 randomly generated queries in each dataset. They are mentioned in Table 3.

4.1.4 Metrics. To quantify performance, we use:

- **Accuracy:** Let $\mathcal{A}^*$ be the optimal top-k answer set and $\mathcal{A}$ the answer set returned by RQuBE (or a baseline). The accuracy is quantified as $\frac{\mathcal{A}^* \cap \mathcal{A}}{k}$.
- **Ranking Preservation:** To measure how well the ranking among nodes in the top-k set is preserved, we use *Spearman's rank correlation coefficient* (ρ), *Kendall's Tau*, and *NDCG* . In our case, the two ranked lists are $\mathcal{A}^*$ and $\mathcal{A}$. We compute the correlation between $\mathcal{A}^*$ and $\mathcal{A}$ by only considering the nodes in $\mathcal{A}^* \cap \mathcal{A}$.

Table 3: Hyper-parameter values.

Dataset	η	z	p	d
Co-author	0.001	500	20%	2
Citation	0.01	2000	20%	2
Freebase	0.01	2000	20%	2
StackOverflow	0.01	1000	20%	2
Yeast	0.01	1000	20%	2



(a) Accuracy (b) Rank Correlation (c) Kendall's Tau (d) NDCG (e) Confidence Similarity (f) Running time

(g) Accuracy (h) Rank correlation (i) Kendall's Tau (j) NDCG (k) Confidence similarity (l) Running time

Figure 6: (a-f) Impact of k on accuracy, ρ, τ, NDCG , C-sim, and running time. (g-l) Impact of support threshold θ on accuracy, ρ, τ, NDCG , C-sim, and running time.

- **Confidence Similarity:** The accuracy metric assumes any node in $\mathcal{A}^* \setminus \mathcal{A}$ is a false positive. In reality, a false positive may also be "good" if its confidence is almost as high as the true positive in its corresponding ranking order. We capture this aspect through:

$$\text{C-sim} = 1 - \frac{\sum_{i=1}^{k} |conf\,(\mathcal{M}^*, s, \mathcal{A}^*[i]) - conf\,(\mathcal{M}^*, s, \mathcal{A}[i])|}{k} \quad (6)$$

$\mathcal{A}[i]$ denotes the i^{th} ranked node in $\mathcal{A}$. Since $Conf(\mathcal{M}^*, s, v) \in [0, 1]$, C-sim $\in [0, 1]$ as well.

4.2 Comparison with Baselines

Efficiency: In Fig. 5a, we compare the running times of RQuBE with Optimal and Arrival. RQuBE is 4 and 3 orders of magnitudes faster on average than Optimal and Arrival respectively. For RQuBE, we further breakdown the running time into the three components of NFA-inference, candidate set identification, and iterative sampling. The largest consumption of time occurs in iterative sampling, while NFA-inference is the fastest; the contribution of NFA-inference on the overall time is extremely small and hence invisible in the bar plot. This behavior is consistent with our theoretical analysis. Specifically, the dominant factor in NFA-inference, is the length of each sampled path, which is bounded by the diameter of the graph. Candidate generation grows logarithmically with the number of nodes in the graph. Both these factors are significantly smaller than the $O(|C| \log(|C|))$ factor in iterative sampling.

Answer-set Quality: As visible in Figs. 5b-5d, RQuBE consistently achieves accuracy, precision, and C-sim above 0.9 across all datasets. In contrast, Arrival is inferior in quality. This result highlights the importance of candidate generation through NFA-guided random walks. More specifically, Arrival uniformly samples z paths from source to each node within *diam* hops, where *diam* is the effective diameter of the graph. In contrast, RQuBE focuses *only* on the candidate set formed through NFA-guided random walks. On this candidate set, RQuBE performs iterative sampling, where z paths are sampled in each iteration on the surviving (i.e., promising) nodes. Thus, the sampling strategy of RQuBE is non-uniform, wherein the higher the promise of a node, the more is the number of samples allocated for it. This strategy enables better balance between quality and efficiency when compared to Arrival.

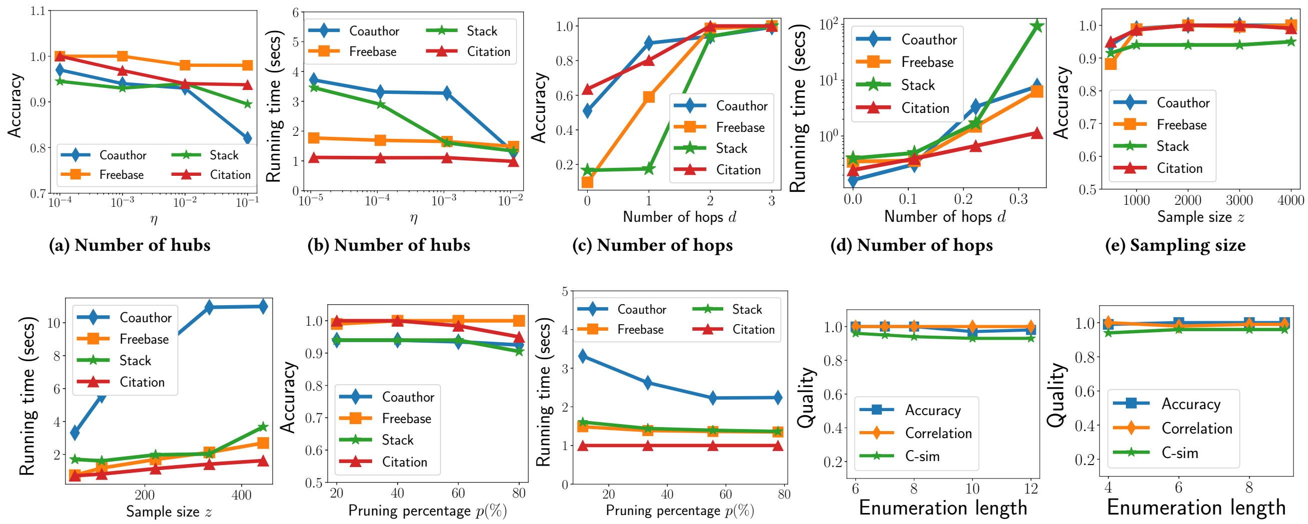
The accuracy of Arrival is the lowest in `Freebase`. `Freebase` is dense and has a large number of labels. In this scenario, for most nodes, when z paths are sampled, they contain less than θ NFA-accepting paths. Consequently, Arrival prunes them off. In RQuBE, a node gets pruned off due to support threshold only in the last iteration of iterative sampling. Thus, nodes that survive till the last iteration have iz paths sampled, where i is the number of iterations. This allows more reliable estimates of support.

4.3 Impact of Input Parameters

Impact of k: Figs 6a-6e present the impact on efficacy against k. We observe that with higher k, the accuracy of RQuBE drops marginally, with the effect being most pronounced in `Co-author`. `Co-author` is the most difficult dataset since it contains the largest number of labels. When the label set is large, there is more heterogeneity in the network, which in turn, decreases the efficacy of sampling. Despite these challenges, accuracy remains above 0.8 under all values of $k \in [10, 100]$. Note that C-sim is best in `Co-author`, which indicates that even though the answer set may be different from the ground-truth, their quality is almost equally good.

In the ranking preservation metrics of ρ, Kendall's Tau and NDCG, we observe that on all datasets except `StackOverflow`, the metrics are ≈ 1 (Figs 6b-6d). Distortion of ranking is the highest in `StackOverflow`. Since `StackOverflow` has the lowest number of unique labels (See Table 2), there are many nodes with similar scores and hence ranking preservation is more difficult.

In Fig. 6f, we analyze the impact of k on running time. As expected, the time increases marginally and saturates quickly. This indicates that scalability is not a concern even when $k > 100$. We



(a) Number of hubs (b) Number of hubs (c) Number of hops (d) Number of hops (e) Sampling size

(f) Sampling size (g) Pruning percentage (h) Pruning Percentage (i) `Citation` (j) `Freebase`

Figure 7: Impact of hyper-parameters (a-b) η, (c-d) d, (e-f) z and (g-h) p on accuracy and running time of RQuBE. (i-j) path enumeration length threshold during NFA-construction.

note here that NFA-inference and candidate set generation are independent of k. Only iterative sampling depends on k and since typically $k \ll |C|$, the impact of k on running time is minimal.

Support Threshold θ: The impact of θ on accuracy is similar to k (Fig. 6g); there is minor deterioration in accuracy as θ increases. A closer inspection on this behavior reveals that as θ increases, the candidate set reduces. When the candidate set size reduces, the skewness in the distribution of confidence values reduces, and this results in a drop in accuracy. When the distribution is highly skewed, even a low sample set of paths is able to better distinguish the top-k set from the candidate space. The impact on ranking preservation is also similar to what we observed against k (Figs. 6h-6j). Across all metrics, the performance remains above 0.8, with `StackOverflow` being the hardest dataset.

Next, we analyze the impact of θ on the running time in Fig. 6l. As expected, the running time either decreases (most pronounced in `Co-author`) or remains minimally impacted. While increasing θ reduces the candidates set size, that does not guarantee a reduction in running time. Recall that Alg. 2 terminates either when the candidate set size reduces to k, or the top-k set remains unchanged over two iterations. While the time per iteration reduces with reduction in candidate set size, the number of iterations may increase.

4.4 Impact of Hyper-parameters

Impact of η: As η reduces, more hubs are selected and hence the candidate set size grows. This results in increase in computation time (Fig. 7b) and improvement in the accuracy metrics (Fig. 7a). Note that datasets that observe large improvement in accuracy with reduction in η (`Co-author` and `StackOverflow`), also undergo significant increase in running time. This is purely due to the property that these two datasets also observe a significant increase in candidate set with reduction in η.

Number of hops d: To form the candidate set, we enumerate all nodes within d hops from any of the hubs. Thus, the higher the value of d, the larger is the candidate set size. With a larger candidate set size, the accuracy and the running time can only monotonically increase. This trend is reflected in Fig. 7c and Fig. 7d. Note that we the y-axis in the Fig. 7d is in log-scale to better reflect the growth rate. The increase in running time is the highest in `StackOverflow` since the density of `StackOverflow` is the highest. We also notice that beyond $d = 2$, the increase in accuracy is minimal, while the jump in running time is drastic.

Impact of z: We expect both the accuracy and running time to increase with sample size z. These trends are visible in Fig. 7e and Fig. 7f. We observe that across all datasets, a sample set of 1000 to 2000 paths is enough to achieve accuracy above 0.9.

Impact of p: p controls the percentage of candidate nodes pruned in each iteration of iterative sampling (Alg. 2). Clearly, the lower the value of p, the better should be the accuracy, and the larger the computation time. This behavior is visible in Fig. 7g and Fig. 7h. We, however, note that the variation in both accuracy and running time is more profound in `Co-author`. This happens since in `Freebase`, `StackOverflow`, and `Citation`, the average number of iterations per query are 15, 13, and 6 respectively. In contrast, in `Co-author`, the number of iterations is 30.

Path enumeration length in NFA construction: Since the number of paths between any pair of nodes may be exponential, we restrict path enumeration to only those paths that are at most as long as the graph diameter. While this allows computational tractability, it may compromise accuracy. In Figs. 7i-7j, we study this aspect in the datasets with the longest and shortest diameters (`Citation`, diameter=7.6 and `Freebase`, diameter=3.7). Specifically, we vary the path enumeration length threshold from graph diameter to a significantly larger value and measure the impact on the

Table 4: (a) Results from the user survey for each of the 15 subgraphs from `Yeast`. p-values less than 0.01 are denoted as ≈ 0. (b) Precision, Recall and Running times of BF and Raman et al. [34] in NFA-inference.

Metric	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12	#13	#14	#15
Yes%	79%	100%	76%	100%	64%	98%	98%	98%	64%	64%	100%	62%	100%	62%	100%
p-value	0	0	0	0	0.04	0	0	0	0.04	0.04	0	0.07	0	0.07	0

(a) User Survey

Algorithm	Mean Prec.	Median Prec.	Mean Recall	Median Recall	Running Time(sec)
BF	**0.90**	**1**	**0.90**	**1**	**0.012**
Raman	0.46	0.25	0.79	0.71	17.97

(b) Accuracy in NFA-inference

various quality metrics. As visible, there is minimal impact on the quality. This trend may be explained as follows. Let $\mathbb{P}$ be the set of all paths from the source to the destination. By thresholding, we select a subset of paths from $\mathbb{P}$. The larger the threshold, the larger is the subset. As long as the subsets are large, we do not expect the distribution of label sequences in each of these subsets to deviate much from the true distribution in $\mathbb{P}$. Since the NFA characterizes the label sequences, if the sequence distributions across the subsets are similar, the corresponding NFAs are also similar.

4.5 Case Studies on NFA-inference

4.5.1 User Survey. Does RQuBE infer the regex that the user intends to communicate through the exemplar? We ask this question in an user survey and measure the performance of NFA-inference.

To construct the user base, invitations were sent out to over 400 people who have a computer science or mathematics background. We restrict to these educational backgrounds to ensure that users are knowledgeable on regular expressions. The user base is diverse enough to include bachelors level students, masters and PhD students, full-time professors and experienced industry professionals. From our invitations, 58 people participated in the survey.

In our user survey, we randomly select 15 subgraphs from the `Yeast` dataset. We restrict the size of a subgraph to at most six nodes so that it is easy for human consumption. Two nodes are randomly selected within the subgraph as the source and destination. An user is then shown the inferred regex for the corresponding source-destination paths and asked whether it is an accurate representation of the paths. The users are clearly instructed to tag the regex as accurate only if it resembles the regex they would use to describe the paths and not simply because it accepts all paths between the source and the destination. As an example Σ^* would accept all paths, but may not be considered accurate. Their response is recorded in the form of a Yes/No answer. The exact subgraphs used in our survey are available at https://github.com/idea-iitd/RQuBE.

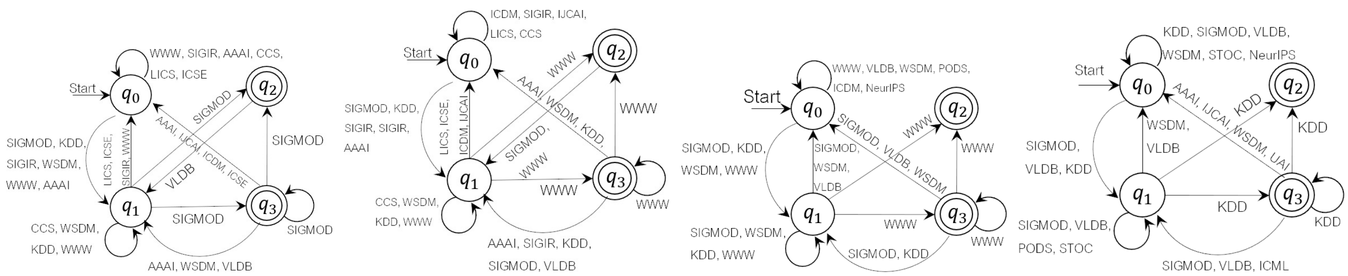
On average, 84% of the responses confirm that RQuBE reconstructs the regex that the users have in mind. Table 4a presents the accuracy per question and their p-values. The p-values are computed under the null hypotheses that both options of Yes or No are equally likely. As visible, the p-value is below 0.05 for 13 out of 15 subgraphs indicating a definite preference towards the inferred regex. In 5 out of 15 subgraphs, 100% of the users agree that the inferred regex correctly reconstructs their intended regex. We selected a random subset of users to obtain qualitative feedback on the cases where they selected "No". The feedback consistently indicated that even in these cases the inferred regex is accurate, but it could have been more specific. Overall, these results establish the efficacy of RQuBE in accurately modeling user's intentions.

4.5.2 Comparison against Ground-truth: Our goal in this experiment is two-fold: (1) Quantify the NFA-inference accuracy against a ground-truth, and (2) Compare BF algorithm with the an alternative MML-based algorithm.

To construct the ground-truth, we randomly choose 25 pairs of source-destination nodes from `Yeast`. For each pair, we manually inspect all paths of lengths up to 3-hops and employ a human labeler to construct the regex that best characterizes these paths. We next identify all nodes in `Yeast` that are connected to the source node through paths that satisfy the human-labeled regex (i.e., confidence=1). These nodes form our ground-truth answer set. We now try to recover this answer set by using the source-destination pair as the exemplar to RQuBE. Note that our earlier experiments on accuracy analyze the efficacy of Phase 2a and 2b against all path enumeration. In this experiment, we focus on the accuracy of the entire pipeline by also including Phase 1. Specifically, to recover the ground-truth, the inferred regex (NFA) must be similar to the human-labeled one. Furthermore, to benchmark inference quality and efficiency of BF [7], we consider a second method where we replace BF with Raman et al. [34], which is one of the prominent MML-based NFA-inference algorithms (§ 3.1.4). We limit ourselves to 25 queries and paths of lengths up to 3 since otherwise the task of constructing human-curated regex is too arduous.

Table 4b presents the results obtained. BF is able to recover the ground-truth for the majority of the pairs. In contrast, the accuracy of Raman is significantly inferior. A closer inspection of the results reveal that for some input exemplars, Raman is able to achieve high precision and recall. But for the majority, both the precision and recall numbers are low. Consequently, its median precision and recall are lower than the averages. As discussed in § 3.1.4, Raman does not have a parameter like ℓ in BF that has a monotonic relationship with the language accepted by the corresponding NFA. Hence, supervised parameter selection is not feasible. This limitation has been recognized in the literature, where the only solution is to either choose the NFA that "looks good" [10], or explore different parameter choices and choose the one where MML is minimized. We choose the second option for Raman.

In addition to quality, we also compare the NFA-construction times of the two algorithms in Table 4b. As visible, BF is ≈ 1500 times faster. Raman is slow since we need to re-compute the change in MML for each merging decision. Furthermore, Raman uses *beam search*, where we need to compare any new NFA with all the NFAs present in the beam at every step to avoid duplicates.



(a) Q2: Weikum→Garcia-Molina **(b) Q3: Garcia-Molina→Weikum** **(c) Q4: Kleinberg→Leskovec** **(d) Q6: Pei→Faloutsos**

Figure 8: Inferred NFAs of a subset of queries from Table 5. For brevity, we only show labels corresponding to A^* conferences [1].

4.5.3 Semantics. Does RQuBE return semantically meaningful results? To conduct this empirical investigation, we use the `Co-author` dataset. In Table 5, we show six queries and their top-1 matches. Fig. 8 shows the inferred NFAs of a subset of queries.

Semantics: Across all queries, the top answer contains an author who publishes in the same community as the destination node in the exemplar. This shows that the proposed NFA-inference mechanism is effective in modeling the intent behind the exemplars. But this is an expected result and our goal is something more sophisticated: to identify nodes that share the same relationship with the source, in terms of labels in connecting paths, as the exemplar destination. To this end consider Query 1. While both Jeffrey Xu Yu and Guoliang Li publish heavily in data analytics, with ICDE being the most prolific venue, they do not share a large number of neighbors. Yet, 91% of the paths from Weikum to Li are accepted by the NFA constructed over Weikum-Yu. Similarly in Query 6, we are able to identify Vazirani and Papadimitrou as being similar with respect to Kleinberg although Vazirani and Kleinberg have not published together and Vazirani and Papadimitrou have published together only twice.

Interpretability: The NFA characterizing an exemplar is interepretable and has its own independent importance in visualizing collaboration diffusion. To elaborate, the NFA not only shows the communities that collaborate, but also the sequentiality of collaborations. As an example, let us consider the Query 2 (Fig. 8a). It is clear that Weikum's direct collaborations are primarily with the Information Retrieval (IR) community (SIGIR, WWW, WSDM). However, his collaborators from IR have connections to the data base (DB) domain, which is evident from the fact that as we get closer to the accepting states, the proportion of DB conferences increase. On the other hand if we look at the NFA for Q4 (Fig. 8c) we see that the relationship from Kleinberg to Leskovec is largely expressed through the same set of conferences which form cycles in the NFA, thereby indicating a greater convergence of interests in the subnetwork connecting Kleinberg to Leskovec.

Asymmetry: Directionality in exemplars matters. To elaborate, let us compare Query 2 with Query 3. Although the exemplar nodes are the same, the source and the destinations are switched. This impacts the label sequence in paths, and hence the NFA inference (Compare Fig. 8a with Fig. 8b).

Impact of destination: Let us compare Query 1 with Query 2. Although the source is same in both, because the destinations are different, the identified best matches are also different.

Table 5: Answers to exemplar-based RPQs by RQuBE on the `Co-author` dataset.

Query #	Source	Destination	Answer	Confidence
1	Gerhard Weikum	Jeffrey Xu Yu	Guoliang Li	0.91
2	Gerhard Weikum	Hector Garcia-Molina	Surajit Chaudhuri	0.95
3	Hector Garcia-Molina	Gerhard Weikum	Stefano Ceri	0.94
4	Jon M. Kleinberg	Jure Leskovec	Ravi Kumar	0.6
5	Jon M. Kleinberg	Christos H. Papadimitriou	Vijay V. Vazirani	0.87
6	Jian Pei	Christos Faloutsos	Jiawei Han	0.79

5 CONCLUSION

Our work is motivated by the observation that in Regular Path Queries (RPQs), the need to express path constraints in the form of regular expressions is a barrier for both technically savvy as well as non-technical users. While non-technical users do not have the expertise to formulate regular expressions, technically savvy users require familiarity with semantic annotations, which are often cryptic and dynamic. We remove this barrier by introducing the *query-by-example* paradigm for RPQs. In our framework, the user is only required to provide an exemplar pair that characterizes the paths relationships a user is interested in. The proposed technique, RQuBE, uses Biermann and Feldman's algorithm [7] to automatically infer the optimal regex that best represents the relationship between the end points of the exemplar pair. This regex is, in effect, a semantically rich representation of the exemplar pair and may be of independent interest beyond the study of the RQuBE problem. Returning to RQuBE, to mitigate the scalability bottleneck of enumerating all possible paths across all nodes in the graph, a sampling based methodology is developed using the novel concept of *NFA-guided random walks with restarts* which may also be of independent interest. Extensive experiments on real, million-scale datasets establish that RQuBE is scalable, more than 3 orders of magnitude faster than baseline strategies, and effective in retrieving semantically meaningful answer sets.

ACKNOWLEDGMENTS

We acknowledge the contributions of Sarthak Singla and Gaurav Jain in conducting several key experiments that were requested during the revision phase.

REFERENCES

[1] [n.d.]. CORE Rankings Portal. http://portal.core.edu.au/conf-ranks/.

[2] 2002. Statistical mechanics of complex networks. *Reviews of Modern Physics* 74, 1 (Jan. 2002), 47–97.

[3] R. Angles, M. Arenas, P. Barcel, P. Boncz, G. Fletcher, C. Gutierrez, T. Lindaaker, M. Paradies, S. Plantikow, J. Sequeda, O. van Rest, and H. Voigt. 2018. G-CORE: A Core for Future Graph Query Languages. In *Proc. SIGMOD '18*. ACM, 1421–1432.

[4] Dana Angluin. 1987. Learning Regular Sets from Queries and Counterexamples. *Inf. Comput.* 75, 2 (Nov. 1987), 87–106. https://doi.org/10.1016/0890-5401(87)90052-6

[5] Marcelo Arenas, Sebastián Conca, and Jorge Pérez. 2012. Counting beyond a Yottabyte, or How SPARQL 1.1 Property Paths Will Prevent Adoption of the Standard. In *Proceedings of the 21st International Conference on World Wide Web* (Lyon, France) *(WWW '12)*. Association for Computing Machinery, New York, NY, USA, 629–638. https://doi.org/10.1145/2187836.2187922

[6] M Ashburner, C A Ball, J A Blake, D Botstein, H Butler, J M Cherry, A P Davis, K Dolinski, S S Dwight, J T Eppig, M A Harris, D P Hill, L Issel-Tarver, A Kasarskis, S Lewis, J C Matese, J E Richardson, M Ringwald, G M Rubin, and G Sherlock. 2000. Gene ontology: tool for the unification of biology. The Gene Ontology Consortium. *Nat Genet* 25, 1 (May 2000), 25–29. https://doi.org/10.1038/75556

[7] A. W. Biermann and J. A. Feldman. 1970. *On the synthesis of finite-state acceptors*. Technical Report AIM-114. Stanford University.

[8] Kurt Bollacker, Colin Evans, Praveen Paritosh, Tim Sturge, and Jamie Taylor. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In *Proc. SIGMOD '08* (Vancouver, Canada). ACM, New York, NY, USA, 1247–1250. https://doi.org/10.1145/1376616.1376746

[9] Angela Bonifati, Radu Ciucanu, and Aurélien Lemay. 2015. Learning Path Queries on Graph Databases. In *Proceedings of the 18th International Conference on Extending Database Technology, EDBT 2015, Brussels, Belgium, March 23-27, 2015*. 109–120.

[10] Matthew S. Collins and Jonathan J. Oliver. 1997. Efficient induction of finite state automata. In *Proc. 13th Conf. Uncertainty in Artificial Intelligence (UAI '97)*. 99–107.

[11] Javier De Las Rivas and Celia Fontanillo. 2010. Protein–Protein Interactions Essentials: Key Concepts to Building and Analyzing Interactome Networks. *PLOS Computational Biology* 6, 6 (06 2010), 1–8. https://doi.org/10.1371/journal.pcbi.1000807

[12] Daniel Christopher Esposito, Joseph Cursons, and Melissa Jane Davis. 2018. Inferring Edge Function in Protein-Protein Interaction Networks. (2018). To appear in *Bioinformatics*, Preprint doi:10.1101/321984.

[13] W. Fan, J. Li, S. Ma, N. Tang, and Y. Wu. 2011. Adding regular expressions to graph reachability and pattern queries. In *Proc. ICDE '11*. 39–50.

[14] Dániel Fogaras, Balázs Rácz, Károly Csalogány, and Tamás Sarlós. 2005. Towards scaling fully personalized PageRank: algorithms, lower bounds, and experiments. *Internet Math.* 2, 3 (2005), 333–358.

[15] G. D. Hachtel, J. K. Rho, F. Somenzi, and R. Jacoby. 1991. Exact and Heuristic Algorithms for the Minimization of Incompletely Specified State Machines. In *Proceedings of the Conference on European Design Automation* (Amsterdam, The Netherlands) *(EURO-DAC '91)*. IEEE Computer Society Press, Washington, DC, USA, 184–191.

[16] Ruoming Jin, Hui Hong, Haixun Wang, Ning Ruan, and Yang Xiang. 2010. Computing Label-constraint Reachability in Graph Databases. In *Proc. SIGMOD '10* (Indianapolis, Indiana, USA). ACM, New York, NY, USA, 123–134.

[17] Timothy Kam, Tiziano Villa, Robert K. Brayton, and Alberto Sangiovanni-Vincentelli. 1993. *A Fully Implicit Algorithm for Exact State Minimization*. Technical Report. USA.

[18] Mineichi Kudo and Masaru Shimbo. 1988. Efficient regular grammatical inference techniques by the use of partial similarities and their logical relationships. *Pattern recognition* 21, 4 (1988), 401–409.

[19] Jure Leskovec and Rok Sosič. 2016. SNAP: A General-Purpose Network Analysis and Graph-Mining Library. *ACM Transactions on Intelligent Systems and Technology (TIST)* 8, 1 (2016), 1.

[20] D. A. Levin, Y. Peres, and E. L. Wilmer. 2017. *Markov chains and mixing times* (2e ed.). American Mathematical Soc.

[21] József Marton, Gábor Szárnyas, and Dániel Varró. 2017. Formalising openCypher Graph Queries in Relational Algebra. In *Proc ADBIS '17*. Springer, 182–196.

[22] Miller McPherson, Lynn Smith-Lovin, and James M Cook. 2001. Birds of a Feather: Homophily in Social Networks. *Annual Review of Sociology* 27, 1 (2001), 415–444.

[23] Alberto O. Mendelzon and Peter T. Wood. 1995. Finding Regular Simple Paths in Graph Databases. *SIAM J. Comput.* 24, 6 (Dec. 1995), 1235–1258.

[24] Laurent Miclet. 1980. Regular inference with a tail-clustering method. *IEEE Transactions on Systems, Man, and Cybernetics* 10, 11 (1980), 737–743.

[25] Dheepikaa Natarajan and Sayan Ranu. 2016. A scalable and generic framework to mine top-k representative subgraph patterns. In *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 370–379.

[26] Dheepikaa Natarajan and Sayan Ranu. 2018. Resling: a scalable and generic framework to mine top-k representative subgraph patterns. *Knowledge and Information Systems* 54, 1 (2018), 123–149.

[27] Mark EJ Newman. 2000. Models of the small world. *Journal of Statistical Physics* 101, 3 (2000), 819–841.

[28] Arlindo L. Oliveira and Stephen Edwards. 1995. Inference of State Machines from Examples of Behavior.

[29] Anil Pacaci, Angela Bonifati, and M Tamer Özsu. 2020. Regular path query evaluation on streaming graphs. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 1415–1430.

[30] Jon D Patrick and KE Chong. 1987. Real-time inductive inference for analysing human behaviour. In *AI'87-Proceedings Australian Joint Artificial Intelligence Conference, Sydney*. 305–322.

[31] You Peng, Ying Zhang, Xuemin Lin, Lu Qin, and Wenjie Zhang. 2020. Answering billion-scale label-constrained reachability queries within microsecond. *Proceedings of the VLDB Endowment* 13, 6 (2020), 812–825.

[32] Arneish Prateek, Arijit Khan, Akshit Goyal, and Sayan Ranu. 2020. Mining top-k pairs of correlated subgraphs in a large network. *Proceedings of the VLDB Endowment* 13, 9 (2020), 1511–1524.

[33] Eric Prud'hommeaux and Andy Seaborne. 2008. SPARQL Query Language for RDF. W3C Recommendation. http://www.w3.org/TR/rdf-sparql-query/

[34] A. Raman, P. Andreae, and J. Patrick. 1998. A Beam Search Algorithm for PFSA Inference. *Pattern Anal. Appl* 1 (1998), 121–129.

[35] Sayan Ranu, Minh Hoang, and Ambuj Singh. 2013. Mining discriminative subgraphs from global-state networks. In *Proceedings of the 19th ACM SIGKDD international conference on knowledge discovery and data mining*. 509–517.

[36] Neha Sengupta, Amitabha Bagchi, Maya Ramanath, and Srikanta Bedathur. 2019. ARROW: Approximating Reachability using Random-walks Over Web-scale Graphs. In *Proc. of the 35th IEEE Intl. Conference on Data Engineering (ICDE)*.

[37] Qawi K Telesford, Karen E Joyce, Satoru Hayasaka, Jonathan H Burdette, and Paul J Laurienti. 2011. The ubiquity of small-world networks. *Brain connectivity* 1, 5 (2011), 367–375.

[38] Frank Tetzel, Wolfgang Lehner, and Romans Kasperovics. 2020. Efficient Compilation of Regular Path Queries. *Datenbank-Spektrum* 20, 3 (2020), 243–259.

[39] Ken Thompson. 1968. Programming Techniques: Regular expression search algorithm. *Commun. ACM* 11, 6 (June 1968), 419–422. https://doi.org/10.1145/363347.363387

[40] Jithin Vachery, Akhil Arora, Sayan Ranu, and Arnab Bhattacharya. 2019. RAQ: Relationship-Aware Graph Querying in Large Networks. In *The World Wide Web Conference*. 1886–1896.

[41] Lucien D.J. Valstar, George H.L. Fletcher, and Yuichi Yoshida. 2017. Landmark Indexing for Evaluation of Label-Constrained Reachability Queries. In *Proc. SIGMOD '17* (Chicago, Illinois, USA). ACM, New York, NY, USA, 345–358. https://doi.org/10.1145/3035918.3035955

[42] Oskar van Rest, Sungpack Hong, Jinha Kim, Xuming Meng, and Hassan Chafi. 2016. PGQL: A Property Graph Query Language. In *Proc. 4th Intl. Wkshp. on Graph Data Management Experiences and Systems (GRADES '16) (GRADES '16)*. Article 7, 1-6 pages.

[43] F Vernadat. 1982. Regular grammatical inference by a successor method. In *IEEE Symposium on Pattern Recognition*.

[44] Denny Vrandečić and Markus Krötzsch. 2021. Wikidata:Database reports/List of properties/all . https://www.wikidata.org/wiki/Wikidata:Database_reports/List_of_properties/all

[45] Sarisht Wadhwa, Anagh Prasad, Sayan Ranu, Amitabha Bagchi, and Srikanta Bedathur. 2019. Efficiently answering regular simple path queries on large labeled networks. In *SIGMOD*. 1463–1480.

[46] Moshé M. Zloof. 1975. Query-by-Example: The Invocation and Definition of Tables and Forms. In *Proceedings of the 1st International Conference on Very Large Data Bases*. 1–24.

[47] Lei Zou, Kun Xu, Jeffrey Xu Yu, Lei Chen, Yanghua Xiao, and Dongyan Zhao. 2014. Efficient processing of label-constraint reachability queries in large graphs. *Information Systems* 40 (2014), 47 – 66. https://doi.org/10.1016/j.is.2013.10.003