Saint Petersburg State University

Georgiy Belyanin

Educational practice

Development of linear algebra-based regular path query algorithm

Education level: bachelor

Speciality *02.03.03 "Software and Administration of Information Systems"*

Scientific supervisor:
C.Sc, docent S.V. Grigoriev

Saint Petersburg
2025

Contents

Introduction

Graph database management systems nowadays are becoming more popular in a variety of different fields, including analysis of knowledge bases [13], biological data [12], the semantic web [10], and source code [22]. Edge-labeled directed graphs give a solid framework for data representation if you take vertices as the objects and edges as the relation between them. This approach appears to be versatile enough to be described in a few data model standards, such as the W3C Resource Description Framework (RDF) [11].

Language-constrained path querying provides a way to search for paths in edge-labeled graphs where constraints are formulated in terms of formal languages. A path is accepted if the sentence formed by the concatenation of path edge labels is in the desired language. If such constraints are defined by regular languages, the query is called a regular path query (RPQ). In practice it can be handy to find kind of pseudo-paths allowing the movement in the direction opposite to the existing edges. This problem is solved by two-way regular path queries (2-RPQs) [19, 17]. Queries containing specific starting or final vertices are called single-source and single-destination correspondingly. Single-source and single-destination 2-RPQs are adopted in a few graph query languages such as GQL, SQL/PGQ, and Cypher. For performance considerations Graph DBMS provide different query semantics support. The user or an execution engine may determine what is needed to complete the query and select the most appropriate algorithm. Sometimes only a set of nodes reachable by a path satisfying the constraints is needed, i.e. reachability problem. The user may need only one of the shortest paths, i.e. any shortest paths semantic, or all paths, i.e. all simple paths, and all trails semantics [9].

Efficient evaluation of RPQs and 2-RPQs is still an open problem, and new methods are actively studied [6, 23, 3]. One of the most promising approaches in graph processing is linear algebra-based algorithms [15, 14]. Turns out a lot of graph algorithms, e.g., BFS, triangle count, and Bellman-Ford, could be expressed using operations over graph adjacency matrices [14]. At first glance, this approach seems inefficient due to memory consumption and algebraic operations time complexity when using dense matrix representations. A directed graph with n vertices has at most n^2 edges but for a lot of real-world graphs amount of edges appear to be many times less than n^2 and their adjacency matrices turn out to be sparse allowing us to use special sparse matrix representations. Utilizing sparsity allows one to achieve compact memory representation and high performance due to parallelization of matrix operations [15].

Linear algebra-based algorithms over sparse matrices demonstrate a decent performance in evaluating two-way regular path queries. Mainly, there are two different approaches: breadth-first-search-based approach [5] which is capable of solving the problem under different semantics and Diego Arroyuelo et al. approach [3] based on folding the abstract syntax tree representing the desired regular constraints that can be used to solve a reacha-

bility problem. Though they are not studied enough: they still demonstrate performance loss in some queries against other approaches and it is not clear which of them is better to use for different query kinds. Performance of using linear algebra for seeking all paths satisfying 2-RPQ has not been compared to other approaches yet.

In this work we improve the performance of the breadth-first-search approach for solving reachability 2-RPQs. We provide an extensive comparison of two different linear algebra 2-RPQ algorithms: BFS-based and Diego Arroyuelo et al. by comparing performance on different query kinds. We analyze capability of solving all simple paths and all trails 2-RPQ semantics by linear algebra against state-of-the-art MillenniumDB DBMS [16].

1 Problem statement

The goal of the scientific work is to develop a linear algebra-based regular path query algorithm. The following tasks have been formulated in order to achieve it.

1. Improve the BFS-based RPQ reachability algorithm evaluation time on simpler queries which affect only a few vertices.
2. Compare different linear algebra-based approaches and determine the best query kinds for each of them.
3. Compare linear algebra-based algorithms for all simple paths and all trails semantics against MillenniumDB.

2 Preliminaries

First we define the graph that we use as a data model.

Definition 2.1 (Edge-labelled graph)**.** A quadruple $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ is called an *edge-labelled graph* (or a *graph*) if:

- V is a finite set of vertices;
- $E \subseteq V \times V$ is a finite set of edges;
- L is a finite set of labels;
- $\lambda_{\mathcal{G}} : E \mapsto 2^L$ represents edge labels.

Any finite set can be enumerated by natural numbers from 1 to n. For the rest of the work, we will assume that the vertices of the graph $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ are enumerated, and without loss of generality $V = \{1, 2, ..., |V|\}$.

Let $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ be an edge-labeled graph. Introduce symbols and sets:

- $a^- \notin L$ for $a \in L$, $(a^-)^- = a$ and $a = b \Leftrightarrow a^- = b^-$
- $e^- = (v, u)$ where $e = (u, v)$.
- $E^- = \{e^- \mid e \in E\}$.
- $L^- = \{a^- \mid a \in L\}$.
- $\lambda_{\mathcal{G}}^- : E^- \mapsto 2^{L^-},\ \lambda_{\mathcal{G}}^-(e^-) = \{a^- \mid a \in \lambda_{\mathcal{G}}(e)\}$.

We also need these sets to generalize the definition of the directed graph to be able to traverse it in both directions:

- $E^{\leftrightarrow} = E \cup E^-$.
- $L^{\leftrightarrow} = L \cup L^-$.
- $\lambda_{\mathcal{G}}^{\leftrightarrow} : E^{\leftrightarrow} \mapsto 2^{L^{\leftrightarrow}},\ \ \lambda_{\mathcal{G}}^{\leftrightarrow}(e) = \lambda_{\mathcal{G}}(e) \cup \lambda_{\mathcal{G}}^-(e)$.
- $\mathcal{G}^{\leftrightarrow} = \langle V, E^{\leftrightarrow}, L^{\leftrightarrow}, \lambda^{\leftrightarrow} \rangle$.

Definition 2.2 (Path)**.** A *path* $\pi = (e_1, e_2, ..., e_n)$ of length n in the graph $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ from u_1 to v_n is a finite sequence of edges $e_i = (u_i, v_i) \in E$ s.t. $\forall 1 \leq i \leq n-1;\ v_i = u_{i+1}$.

We say there are *zero-length paths* represented by an empty sequence from v to v for all $v \in V$.

Definition 2.3 (Simple path)**.** Let $\pi = (e_1, e_2, ..., e_n)$ and $e_i = (u_i, v_i)$. Recall a path π *simple* if $\forall 1 \leq i, j \leq n;\ v_i = u_j \Rightarrow j = i + 1$. In other words, a simple path is a path without repeated vertices except the first and the last one.

Definition 2.4 (Trail)**.** Recall a path $\pi = (e_1, e_2, ..., e_n)$ a *trail* if $\forall 1 \leq i, j \leq n;\ e_i = e_j \Rightarrow i = j$. In other words, a trail is a path without repeated edges.

Definition 2.5 (2-way path)**.** A *2-way path* (2-path) $\pi = (e_1, e_2, ..., e_n)$ in the graph $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ is a path in $\mathcal{G}^{\leftrightarrow} = \langle V, E^{\leftrightarrow}, L^{\leftrightarrow}, \lambda^{\leftrightarrow}_{\mathcal{G}} \rangle$.

2-way simple path (2-simple path) and *2-way trail* (2-trail) are introduced the same way as they're defined for paths.

The ω maps paths to words as defined below.

- $\omega_{\mathcal{G}}(\pi) = \{a_1 \cdot a_2 \cdot ... \cdot a_n \mid a_i \in \lambda_{\mathcal{G}}(e_i)\}$ for path $\pi = (e_1, e_2, ..., e_n)$ in $\mathcal{G}$.
- $\omega^{\leftrightarrow}_{\mathcal{G}}(\pi) = \{a_1 \cdot a_2 \cdot ... \cdot a_n \mid a_i \in \lambda^{\leftrightarrow}_{\mathcal{G}}(e_i)\}$ for 2-path $\pi = (e_1, e_2, ..., e_n)$ in $\mathcal{G}$.

where $\cdot$ denotes concatenation.

We will use *extended regular expressions* to represent the constraints in 2-RPQs.

Definition 2.6 (2-way extended regular expressions)**.** *2-way (extended) regular expressions* (2RE) over the alphabet Σ can be defined inductively as follows:

- ε is a 2RE.
- a is a 2RE if $a \in \Sigma^{\leftrightarrow}$ where $\Sigma^{\leftrightarrow} = \Sigma \cup \Sigma^{-}$.
- $R_1\ R_2$ (concatenation) is RE if R_1 and R_2 are 2REs.
- $R_1 \mid R_2$ (conjunction) is RE if R_1 and R_2 are 2REs.
- R^* if R is a 2RE.
- R^+ if R is a 2RE.

2-way extended regular expressions define regular languages over $\Sigma^{\leftrightarrow}$ recursively by a map σ.

- $\sigma(\varepsilon) = \{\varepsilon\}$.
- $\sigma(a) = \{a\}$ for $a \in \Sigma^{\leftrightarrow}$.
- $\sigma(R_1\ R_2) = \{r_1\ r_2 \mid r_1 \in \sigma(R_1), r_2 \in \sigma(R_2)\}$.
- $\sigma(R_1 \mid R_2) = \sigma(R_1) \cup \sigma(R_2)\, \varnothing$.
- $\sigma(R^*) = \bigcup_{n=0}^{\infty} \sigma(R^n)$
- $\sigma(R^+) = \bigcup_{n=1}^{\infty} \sigma(R^n)$

, where $\sigma(R^n) = \sigma(\underbrace{R\ R\ ...\ R}_{n \text{ times}})$, and $\sigma(R^0) = \{\varepsilon\}$.

Regular languages (RLs) represent the set of all languages that are accepted by finite automata. However, we are going to look for 2-way paths for which we need an NFA modification to be introduced.

Definition 2.7 (2-way non-deterministic finite automaton)**.** A *2-way non-deterministic finite automaton* (2-NFA) is a tuple $\mathcal{N} = \langle Q, \Sigma, \Delta, \lambda_{\mathcal{N}}, Q_S, Q_F \rangle$, where:

- Q is a finite set of states;
- Σ is a finite alphabet;
- $\Delta \subseteq Q \times Q$ is transition relation;
- $\lambda_{\mathcal{N}} : \Delta \mapsto 2^{\Sigma^{\leftrightarrow}}$ maps transitions to labels;
- $Q_S \subseteq Q$ is a set of starting states;
- $Q_F \subseteq Q$ is a set of final states.

Definition 2.8 (2-way deterministic finite automaton)**.** A *2-way non-deterministic finite automaton* (2-DFA) is a special case of 2-NFA $\mathcal{N} = \langle Q, \Sigma, \Delta, \lambda_{\mathcal{N}}, Q_S, Q_F \rangle$ having $|Q_S| = 1$ and $\forall\ (q, q'), (q, q'') \in \Delta; q' \neq q''; \lambda_{\mathcal{N}}((q, q')) \cap \lambda_{\mathcal{N}}((q, q'')) = \varnothing$.

In practice special kinds of strings called *regular expressions* provide a useful facility to define regular languages. The algorithms supplying the transformations between regular expressions, RLs and 2-NFAs are out of the scope of this paper.

Notice that 2-NFA can be seen as a graph $\mathcal{G}_{\mathcal{N}} = \langle Q, \Delta, \Sigma^{\leftrightarrow}, \lambda_{\mathcal{N}} \rangle$ equipped with the set of starting states $Q_S \subseteq Q$ and the set of final states $Q_F \subseteq Q$. Analogously introduce sets Δ^{-}, $\lambda_{\mathcal{N}}^{-}$ and the map $\omega_{\mathcal{N}} = \{a_1 \cdot a_2 \cdot ... \cdot a_n \mid a_i \in \lambda_{\mathcal{N}}(\delta_i)\}$ for the paths $\pi = (\delta_1, \delta_2, ..., \delta_n)$ in the graph $\mathcal{G}_{\mathcal{N}}$.

Vice versa a graph $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ equipped with some $V_S \subseteq V$ can bee seen as 2-NFA $\mathcal{N}_{\mathcal{G}} = \langle V, L, E, \lambda_{\mathcal{G}}^{\leftrightarrow}, V_S, V \rangle$. Such 2-NFA accepts the language:

$$[\![\mathcal{N}_G]\!] = \bigcup_{\substack{\text{2-path } \pi_{\mathcal{G}} \text{ in } \mathcal{G} \\ \text{from } v_S \in V_S}} \omega_{\mathcal{G}}^{\leftrightarrow}(\pi_{\mathcal{G}})$$

We have all the required preliminaries to formally state the problem solved by the 2-RPQ algorithm.

Definition 2.9 (Two-way regular path query)**.** Recall *two-way regular path query* (2-RPQ) a triple $\langle \mathcal{G}, \mathcal{R}, v_s \rangle$ where $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ is a graph, $\mathcal{R}$ is a regular language over the alphabet $L^{\leftrightarrow}$ and $v_s \in V$ is a starting node.

Frequent practical cases are queries with fixed starting or final vertex. At first, we describe a way to solve . For 2-RPQ $\mathcal{Q} = \langle \mathcal{G}, \mathcal{R}, v_s \rangle$ we are interested in an efficient way of evaluating these maps.

- **Single-source reachability**:

$$[\![\mathcal{Q}]\!]_{SSR} = \left\{ u \in V \;\middle|\; \exists \text{ 2-path } \pi \text{ in } \mathcal{G} \text{ from } v_s \text{ to } u \text{ and } \omega^{\leftrightarrow}_{\mathcal{G}}(\pi) \cap \mathcal{R} \neq \varnothing \right\}.$$

- **Single-destination reachability**:

$$[\![\mathcal{Q}]\!]_{SDR} = \left\{ u \in V \;\middle|\; \exists \text{ 2-path } \pi \text{ in } \mathcal{G} \text{ from } u \text{ to } v_s \text{ and } \omega^{\leftrightarrow}_{\mathcal{G}}(\pi) \cap \mathcal{R} \neq \varnothing \right\}.$$

- **Single-source all simple paths**:

$$[\![\mathcal{Q}]\!]_{SSASP} = \left\{ \pi \;\middle|\; \pi \text{ is a 2-simple path in } \mathcal{G} \text{ from } v_s \text{ and } \omega^{\leftrightarrow}_{\mathcal{G}}(\pi) \cap \mathcal{R} \neq \varnothing \right\}.$$

- **Single-destination all simple paths**:

$$[\![\mathcal{Q}]\!]_{SDASP} = \left\{ \pi \;\middle|\; \pi \text{ is a 2-simple path in } \mathcal{G} \text{ to } v_s \text{ and } \omega^{\leftrightarrow}_{\mathcal{G}}(\pi) \cap \mathcal{R} \neq \varnothing \right\}.$$

- **Single-source all trails**:

$$[\![\mathcal{Q}]\!]_{SSAT} = \left\{ \pi \;\middle|\; \pi \text{ is a 2-trail in } \mathcal{G} \text{ from } v_s \text{ and } \omega^{\leftrightarrow}_{\mathcal{G}}(\pi) \cap \mathcal{R} \neq \varnothing \right\}.$$

- **Single-destination all trails**:

$$[\![\mathcal{Q}]\!]_{SDAT} = \left\{ \pi \;\middle|\; \pi \text{ is a 2-trail in } \mathcal{G} \text{ to } v_s \text{ and } \omega^{\leftrightarrow}_{\mathcal{G}}(\pi) \cap \mathcal{R} \neq \varnothing \right\}.$$

In order to operate with graphs in terms of linear algebra we need to see how graphs could be represented using Boolean matrices.

At first, introduce a connection between relations and Boolean matrices. For two enumerated finite sets $A = \{1, 2, ...n\}$, $B = \{1, 2, ..., m\}$ for some $n, m \in \mathbb{N}$ a binary matrix T of size $|A| \times |B|$ can be used to represent a relation $\mathcal{T} \subseteq A \times B$: $T_{ij} = 1 \Leftrightarrow (i, j) \in \mathcal{T}$. Recall this matrix T *a matrix representing the relation* $\mathcal{T}$.

Let $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ be an arbitrary graph and $\mathcal{N} = \langle Q, \Sigma, \Delta, \lambda_{\mathcal{N}}, Q_S, Q_F \rangle$ an arbitrary 2-NFA. Introduce the sets:

- $E^a = \{e \mid e \in E, a \in \lambda^{\leftrightarrow}_{\mathcal{G}}(e)\}$ for all $a \in L^{\leftrightarrow}$.
- $\Delta^a = \{\delta \mid \delta \in \Delta, a \in \lambda_{\mathcal{N}}(\delta)\}$ for all $a \in \Sigma^{\leftrightarrow}$.

These sets consist of the edges of the graph $\mathcal{G}$ and the transitions of 2-NFA $\mathcal{N}$ marked with a label a. They can be seen as binary relations over the sets V and Q correspondingly.

Definition 2.10 (Adjacency matrix of the label)**.** Let G^a be the matrix representing the binary relation E^a. G^a is called *an adjacency matrix of the label* a.

Definition 2.11 (Boolean decomposition of the adjacency matrix)**.** A *Boolean decomposition of the adjacency matrix* G is a set of Boolean matrices $\{G^a \mid a \in L\}$.

Boolean decomposition of the adjacency matrices of some real-world data represented by graphs is a set of **sparse** matrices. This fact strictly leads to the idea of exploiting it for an efficient representation and using sparse matrix operation algorithm implementations.

Also note that for the given graph $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$, $E^{a^-} = (E^a)^-$ and $G^{a^-} = (G^a)^T$.

We need to define the following linear algebra operations over the Boolean matrices. Let $A = (a_{ij})$, $B = (b_{ij})$, $C = (c_{ij})$ be the matrices over Boolean algebra $\langle \mathcal{B} = \{0, 1\}, \vee, \wedge, \neg, 0, 1 \rangle$. Introduce the following operations and constants.

- **Sum**: $A_{n\times m} \oplus B_{n\times m} = (a_{ij} \vee b_{ij})_{n\times m}$.
- **Product**: $A_{n\times m} \otimes B_{m\times k} = \left(\bigvee_{k=1}^{n} a_{ik} \wedge b_{kj} \right)_{n\times k}$.
- **Transposition**: $A^T_{n\times m} = (a_{ji})_{m\times n}$.
- **Zero-matrix**: $\mathbf{0}_{n\times k} = (0)_{n\times k}$.
- **Mask**: $A_{n\times m}\langle C_{n\times m} \rangle = (a_{ij} \wedge c_{ij})_{n\times m}$
- **Complement**: $\neg A = (\neg a_{ij})$

These are all the preliminaries required to describe the existing linear algebra-based algorithms.

2.1 Linear algebra BFS-based 2-RPQ algorithm

Algorithm results with a vector P^F with the following property.

$$\begin{cases} P_{q_F v} = 1 \\ q_F \in Q_F \end{cases} \Leftrightarrow \begin{cases} \exists \pi_{\mathcal{G}} \text{ in } \mathcal{G} \text{ from } v_s \text{ to } v \\ \omega^{\leftrightarrow}_{\mathcal{G}}(\pi_{\mathcal{G}}) \cap \mathcal{R} \neq \varnothing \,. \end{cases}$$

It has been also shown [4], that the LARPQ algorithm is capable of solving all paths, all simple paths, all trails 2-RPQ problems. Though using Boolean matrices is not sufficient for it. It requires matrix object having sets as having entries and. The detailed description of the LARPQ algorithm for solving all path semantics is out of the scope of the work.

BFS-based LARPQ algorithm has been implemented inside the LAGraph [14] linear algebra graph-processing algorithms infrastructure and is available in the stable version

Algorithm 1: Single Source 2-RPQ using linear algebra

input : $\mathcal{N} = \langle Q, \Sigma, \Delta, \lambda_{\mathcal{N}} Q_S, Q_F \rangle$, $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$, $v_s \in V$
output Vector P^F of size $1 \times |V|$
:

$\{N^a\} \leftarrow$ Boolean decomposition of the $\mathcal{N}$ adjacency matrix;
$\{G^a\} \leftarrow$ Boolean decomposition of the $\mathcal{G}$ adjacency matrix;
$P_{|Q|\times|V|} \leftarrow \mathbf{0}_{|Q|\times|V|}$;
$M_{|Q|\times|V|} \leftarrow M$ s.t. $M_{qv} = 1$ if $q \in Q_S$, $v = v_s$, otherwise 0;
$F_{1\times|Q|} \leftarrow F$ s.t. $F_{1q} = 1$ if $q \in Q_F$, otherwise 0;
while $M \neq \mathbf{0}$ **do**
    $M \leftarrow \bigoplus_{a \in \Sigma^{\leftrightarrow} \cap L^{\leftrightarrow}} ((N^a)^T \otimes M \otimes G^a)\langle \neg P \rangle$; `// Update` $\mathcal{M}$
    $P \leftarrow P \oplus M$
end
return $P^F = F \otimes P$

of the collection[1]. It uses SuiteSparse:GraphBLAS [8] library to work with sparse matrices which employs compressed-sparse-row (CSR) and compressed-sparse-columns (CSC) sparse matrix formats and their variants for different levels of sparsity. The implementation works with matrices in the MatrixMarket [2] format. Previous evaluations of LARPQ showed that it outperforms various graph database management systems [5]. A prototype of the linear algebra 2-RPQ under other semantics has been also implemented inside LAGraph and currently it is only available in its fork[3]. Benchmarking utilities for both approaches are available separately [4].

2.2 Diego Arroyuelo et al. 2-RPQ reachability algorithm

Diego Arroyuelo et al. in [3] propose another linear-algebra-based 2-RPQ evaluation algorithm called *RPQ-matrix* that is based on a direct translation of regular expression to operations over Boolean matrices.

The authors introduce the recursive schema $\mathcal{T}$ to translate 2-RPQ $Q = \langle \mathcal{G}, \mathcal{R}, V_s, V_f \rangle$ over graph $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$ into matrix operations that are defined as follows.

[1]LAGraph RPQ reachability algorithm: https://github.com/GraphBLAS/LAGraph/blob/stable/experimental/algorithm/LAGraph_RegularPathQuery.c
[2]MatrixMarket format: https://math.nist.gov/MatrixMarket/formats.html
[3]LAGraph 2-RPQ for different semantics: https://github.com/SparseLinearAlgebra/LAGraph/blob/2-rpq-all-paths/experimental/algorithm/LAGraph_2Rpq.c
[4]LAGraph benchmarking facilities: https://github.com/georgiy-belyanin/RPQ-bench/tree/main

$$
\begin{aligned}
\mathcal{T}() &= I \\
\mathcal{T}(a) &= G^a, \text{ for } a \in L \\
\mathcal{T}(a^-) &= G^{a^-}, \text{ for } a \in L \\
\mathcal{T}(R_1 \mid R_2) &= \mathcal{T}(R_1) \otimes \mathcal{T}(R_2) \\
\mathcal{T}(R_1\ R_2) &= \mathcal{T}(R_1) \oplus \mathcal{T}(R_2) \\
\mathcal{T}(R^+) &= \mathcal{T}(R)^+ \\
\mathcal{T}(R^*) &= I + \mathcal{T}(R)^+ \\
\mathcal{T}(Q) &= \mathcal{T}(\mathcal{R}) \text{ if } V_s = V \text{ and } V_f = V \\
\mathcal{T}(Q) &= \langle x \rangle \mathcal{T}(\mathcal{R}) \text{ if } V_s = \{x\} \text{ and } V_f = V \\
\mathcal{T}(Q) &= \mathcal{T}(\mathcal{R}) \langle y \rangle \text{ if } V_s = V \text{ and } V_f = \{y\} \\
\mathcal{T}(Q) &= \langle x \rangle \mathcal{T}(\mathcal{R}) \langle y \rangle \text{ if } V_s = \{x\} \text{ and } V_f = \{y\}
\end{aligned}
$$

Direct translation is not enough to achieve high performance; thus, the authors propose several optimizations. The first part of optimizations is a set of resulting formula (or execution plan) transformations aimed to move row and column constraints to the earliest possible steps.

$$
\begin{aligned}
\langle r \rangle (M \oplus M) \langle c \rangle &= \langle r \rangle M \langle c \rangle \oplus \langle r \rangle M \langle c \rangle \\
\langle r \rangle (M \otimes M) \langle c \rangle &= (\langle r \rangle M) \otimes (M \langle c \rangle) \\
M_1^+ \otimes M_2 \langle c \rangle &= (M_1 \otimes \ldots (M_1 \otimes (M_1 \otimes M_2 \langle c \rangle)) \ldots) \\
&\quad \text{, similarly for } M_1^* \\
\langle r \rangle M_1 \otimes M_2^+ &= (\ldots ((\langle r \rangle M_1 \otimes M_2) \otimes M_2) \ldots \otimes M_2) \\
&\quad \text{, similarly for } M_2^*
\end{aligned}
$$

The second optimization involves reordering matrix operations at query execution time based on sparsity patterns.

For Boolean matrices, the sum operation is both commutative ($A \oplus B = B \oplus A$) and associative ($(A \oplus B) \oplus C = A \oplus (B \oplus C)$), while the product operation is associative, but not commutative. Since computational complexity depends on the number of nonzero elements (denoted $|A|$ for matrix A), we optimize operation order to minimize intermediate matrix sizes.

- **Sum operation**: on given matrices $E_1 \oplus \cdots \oplus E_m$

  1. Initialize $M = \{A_1, \ldots, A_m\}$ where $A_i = \mathcal{T}(E_i)$
  2. While $|M| > 1$

- Find A and B matrices from initialized sequence with minimal $|A| + |B|$
- Compute $C = A \oplus B$
- Update $M \leftarrow (M \setminus \{A, B\}) \cup \{C\}$

- **Product operation**: on given matrices $E_1 \otimes \cdots \otimes E_m$

1. Initialize $M = \{A_1, \ldots, A_m\}$ where $A_i = \mathcal{T}(E_i)$
2. While $|M| > 1$
   - Find A_i and A_{i+1} matrices from initialized sequence with minimal $|A_i| + |A_{i+1}|$
   - Compute $C = A_i \otimes B_{i+1}$
   - Update $M \leftarrow (M \setminus \{A_i, A_{i+1}\}) \cup \{C\}$

In summary, the algorithm consists of the following steps.

1. Build an abstract syntax tree (evaluation plan) representing the desired regular expression in which leaves represent labels and other nodes represent operations such as concatenation, Kleene star, and conjunction, w.r.t. $\mathcal{T}$ defined above.
2. Match nodes with matrices: each leaf label is matched with an adjacency matrix representing it, and every inner node is matched with a matrix that can be computed based on the children matrices depending on the operation the node represents.
3. Optimize the evaluation plan using the transformations described above.
4. Compute the matrix representing the root element with operations reordering applied to optimize computations.

RPQ-matrix has been implemented using sparse matrix implementation written from scratch [5]. It provides two different matrix format approaches including CSR/CSC and k^2-tree formats and uses custom file formats for representing matrices in this formats. Experiments show that the latter k^2-tree implementation demonstrates the best memory consumption whereas CSR/CSC demonstrate rather good performance compared to other graph DBMS [3].

[5] RPQ-matrix repository: github.com/adriangbrandon/rpq-matrix

3 Solution

Since graph databases are actively studied conducting the experiments require extra supplementary work to be done due to the immature infrastructure.

3.1 Infrastructure for benchmarking

Most of the existing 2-RPQ benchmark datasets contain graphs in the RDF data format [11] and SPARQL [2] queries with it. Many database systems use other data models apart from RDF such as property graph [1] and use query languages different from SPARQL. At the same time, linear algebra-based approaches use adjacency matrices and their representations. Even though there are so many different formats there are no any publicly available tools for pre-processing and converting the datasets between them. We implement a set of the following tools to allow one to prepare RDF graphs to be used for benchmarking in various environments.

- A script to remove RDF prefixes to reduce the influence of DBMS behavior not related to the graph algorithms.
- A script to filter out properties from RDFs. RDF data model represents data as a large graph with vertices with properties which are not very important when studying language-constraint path querying.
- A script to convert a graph to a set of adjacency matrices in MatrixMarket format that can be used by various linear algebra-based approaches.
- A script to convert a graph to a set of CSV files that can be imported by various property graph databases.
- A script to convert an arbitrary RDF dataset to MillenniumDB-specific format.
- A script for removing duplicating edges since not all of the systems handle parallel edges the same way.
- A tool for semi-automatic conversion of SPARQL queries to Cypher [7] which is commonly used with the property graph data model.

Combined this tool create a solid framework for preparing datasets to be used for benchmarking different approaches and graph database management systems independently to the data model they use.

3.2 Improving BFS-based algorithm performance on simple queries

Previous experiments from [4] showed that the BFS-based LARPQ algorithm implemented using SuiteSparse:GraphBLAS demonstrated higher execution time on simpler queries with fewer answers than RPQ-matrix and MillenniumDB. This slowdown is related to the sparse matrix library internal and the way it handles matrices with a few entries. SuiteSparse:GraphBLAS is not able to utilize parallel computations and properly use cache when working with a few very sparse matrices. We introduce a new version of the BFS-based algorithm without using extra traversal matrix M described in algorithm 2.

Algorithm 2: Single Source 2-RPQ using linear algebra without an extra traversal matrix

input : $\mathcal{N} = \langle Q, \Sigma, \Delta, \lambda_{\mathcal{N}} Q_S, Q_F \rangle$, $\mathcal{G} = \langle V, E, L, \lambda_{\mathcal{G}} \rangle$, $v_s \in V$
output Vector P^F of size $1 \times |V|$
:

$\{N^a\} \leftarrow$ Boolean decomposition of the $\mathcal{N}$ adjacency matrix;
$\{G^a\} \leftarrow$ Boolean decomposition of the $\mathcal{G}$ adjacency matrix;
$P_{|Q| \times |V|} \leftarrow P$ s.t. $P_{qv} = 1$ if $q \in Q_S$, $v = v_s$, otherwise 0;
$F_{1 \times |Q|} \leftarrow F$ s.t. $F_1 q = 1$ if $q \in Q_F$, otherwise 0;
while P *changes* **do**
  $P \leftarrow P \oplus \bigoplus_{a \in \Sigma^{\leftrightarrow} \cap L^{\leftrightarrow}} ((N^a)^T \otimes P \otimes G^a)$; `// Update` $\mathcal{P}$
end
return $P^F = F \otimes P$

Although it may seem that this algorithm should perform some extra calculations because P contains more entries than M on each step in algorithm 1, this way uses fewer matrices and utilizes caches better.

The algorithm maintains a bit different invariant from the original one when evaluating the main traversal loop. The proof of its correctness is the same straightforward induction as in [5].

$$(q, v) \in \mathcal{P}_n \Leftrightarrow \begin{cases} \exists \text{ 2-path } \pi_{\mathcal{G}} \text{ of length } \leq n \text{ in } \mathcal{G} \text{ from } v_s \text{ to } v \\ \exists \text{ path } \pi_{\mathcal{N}} \text{ of length } \leq n \text{ in } \mathcal{N} \text{ from } q_s \in Q_S \text{ to } q \\ \omega_{\mathcal{G}}^{\leftrightarrow}(\pi_{\mathcal{G}}) \cap \omega_{\mathcal{N}}(\pi_{\mathcal{N}}) \neq \varnothing \,. \end{cases}$$

One might solve a 2-RPQ problem by running algorithm 2 at first and then switching to algorithm 1 when the matrix P becomes dense enough. The most straightforward heuristic to determine the exact moment to perform a switch is when P gets some constant number of non-zero entries within it. Though this constant should be determined empirically for each dataset.

4 Experiments

We would like to compare new implementation of LARPQ and RPQ-matrix to other approaches and between each other. Whereas we are focusing on an efficient way of evaluating RPQs in memory, there are not many candidates to compare. Popular database management systems primarily focus on general availability and ensure availability of the concurrent access at the same time the suggested algorithm implementation is suited only for solving the reachability problem. We have selected the following systems as the most effective and the most related to our use case.

LARPQ-old: is an old version of LARPQ reachabilitiy algorithm without optimizations for simpler queries.

RPQ-matrix (GrB)[6] is an adapted version of Diego Arroyuelo et al. RPQ-matrix algorithm where matrix representations and operations are substituted with SuiteSparse:GraphBLAS equivalents in order to analyse performance impact of basic primitives implementation.

MillenniumDB [16] is a graph-oriented database management system with RDF-model and SPARQL support. It supports a synthetic way to carry out calculations without using disk storage by caching query data. MillenniumDB demonstrates state-of-the-art performance on evaluating regular path queries. Moreover, it is capable of evaluating 2-RPQs under all simple paths and all trails semantics.

FalkorDB (previously RedisGraph [20]) is an in-memory property graph database that also employs SuiteSparse:GraphBLAS for query evaluation. However, it uses OpenCypher modification and supports only a subset of regular path queries (e.g., it is impossible to evaluate repeated-path queries in the form of $(a\ b)^*$). To deal with this, we have carried out the measurements of cypher-compatible and non-cypher-compatible queries separately.

Blazegraph [16] is another graph-oriented database management system using RDF data model and SPARQL query language. It is used by the Wikidata project and is based on more classical B-trees.

All experiments are conducted on a work station with Ryzen 9 7900X 4.7 GHz 12-core, 128 Gb of DDR5 RAM and running Ubuntu 22.04.

4.1 Implementation Details

As mentioned above, for better performance it is important to determine the constant to switch between the BFS-based 2-RPQ reachability approaches to achieve better performance on queries involving fewer vertices. This constant is empirically determined for each graph and for the studied datasets the most suitable value is 1 000. This allows the

[6]SuiteSparse:GraphBLAS-based implementation of RPQ-matrix: https://github.com/suvorovrain/rpq-matrix/tree/gbmod

LARPQ reachability algorithm to provide strong performance on simple queries with very few answers and on analytical queries involving a lot of resulting vertices at the same time.

When it comes to semantics other than reachability SuiteSparse:GraphBLAS used within LAGraph turns to be a very tough choice. The library only supports fixed size semirings which strictly leads to the fact the only way to implement matrices with arbitrary size entries is to introduce alien memory allocations and to store the pointers as entries. LARPQ for solving various all paths semantics needs to store arbitrary amount of arbitrary length paths to guarantee to work properly. It results with the fact that linear algebra algorithms require excessive memory management. To deal with it we limit the maximum amount of paths and the maximum length and exclude the queries not fitting within the limitations. It is an interesting future research question whether it is possible to combine CSR/CSC efficient parallel computations with non-fixed-size structures.

4.2 Dataset Description

For evaluation, we choose the Wikidata dataset from the snapshot provided in terms of MillenniumDB path query challenge [9]. The resource contains both the graph and the set of 660 different 2-RPQs in SPARQL format taken from the Wikidata query log. The second dataset is Yago-2S evaluated with 7 different complex queries taken from [21].

We also want to compare different linear algebra-based approaches and determine the best of them for different query kinds. Real-world datasets are not suitable for it due to complex graph topology. Instead, we employ the synthetic RPQBench dataset generator [18] for controlled performance evaluation across query categories. The structure of the graph ensures reproducible and predictable results when similar queries are executed. The original generator produces arbitrary sized RDF datasets and offers 10 different query kinds without starting or final nodes specified in SPARQL format. We generate a graph and supply these query kinds with randomly generated source and destination vertices.

For systems that do not support the RDF format, the datasets have been converted to an edge-labeled graph. Original SPARQL path queries have been deprefixed, converted to the corresponding minimized DFAs. Queries have been converted to Cypher queries of the form `MATCH ... COUNT (DISTINCT ...)` when it is possible. Queries without starting or final vertex, broken queries involving missing entities have been removed. Final dataset statistics can be summarised as follows.

Wikidata

- 610 million edges, 91 million vertices, 1400 distinct labels.
- 578 queries filtered out from 660 from the query dump.

RPQBench

- Synthetic dataset, 150 million edges, 57 million vertices, 9 distinct labels.

Table 1: Execution time of the new and old LARPQ implementations on different query kinds from the RPQBench dataset. (S) denotes single-source queries, (D) denotes single-destination queries

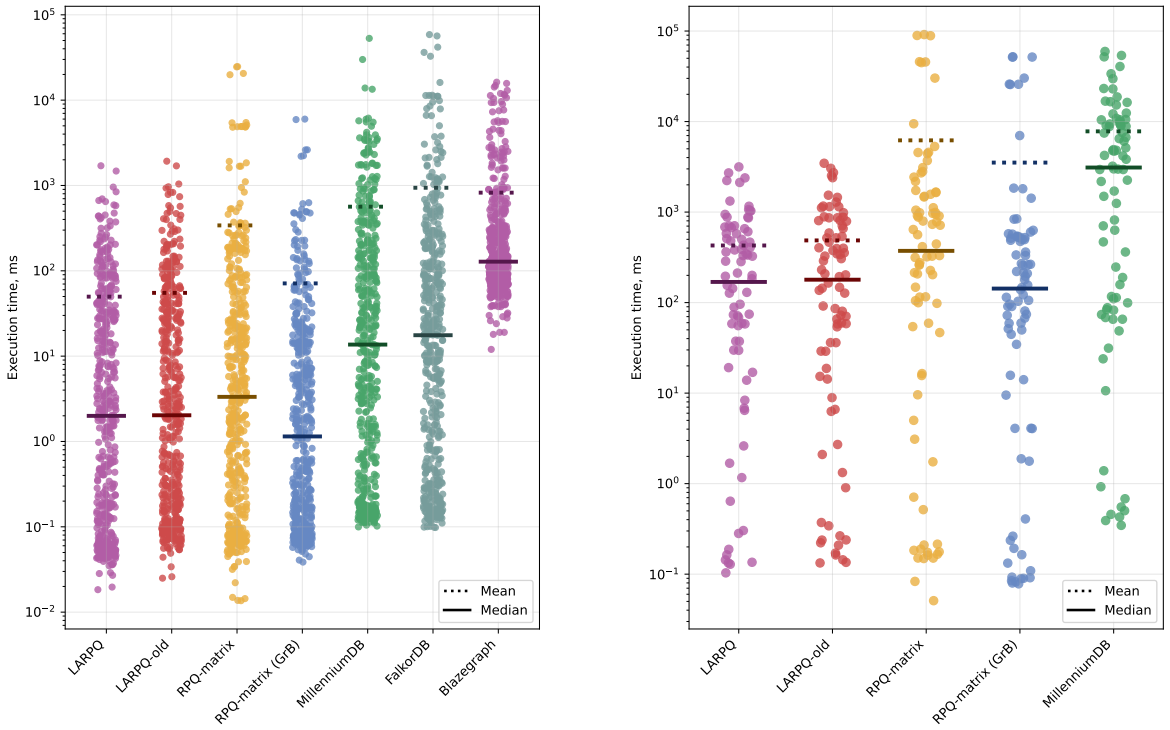
	QUERY	LARPQ	LARPQ-OLD	SPEEDUP
1	$a\ b\ c$, (S)	51	50	1.0
2	$a\ b\ c$, (D)	32	43	1.3
3	$(a\ b\ c) \mid (c\ d\ d)$, (S)	58	62	1.1
4	$(a\ b\ c) \mid (c\ d\ d)$, (D)	46	58	1.3
5	$d*$, (S)	22	49	2.2
6	$d*$, (D)	18	38	2.1
7	$d*\ e$, (S)	21	32	1.6
8	$d*\ e$, (D)	6	9	1.5
9	$d+$, (S)	22	50	2.2
10	$d+$, (D)	17	42	2.4
11	$(a\ b)*$, (S)	1 350	1 355	1.0
12	$(a\ b)*$, (D)	3 618	3 643	1.0
13	$f\ g\ (d \mid e)$, (S)	92 699	64 811	0.7
14	$f\ g\ (d \mid e)$, (D)	2 192	2 127	1.0
15	$f\ g\ (d \mid e)*$, (S)	149 643	119 825	0.8
16	$f\ g\ (d \mid e)*$, (D)	6 166	5 558	0.9
17	$(c \mid g)\ (d \mid e)$, (S)	36	35	1.0
18	$(c \mid g)\ (d \mid e)$, (D)	15	16	1.1
19	$(c \mid g)\ (d \mid e)*$, (S)	929	762	0.8
20	$(c \mid g)\ (d \mid e)*$, (D)	99	154	1.6

- 20 different query kinds supplied 1000 randomly generated source/destination vertices.
- Trivial queries are filtered out (e.g. a^* has ≥ 1 answer).

4.3 Comparison of the new and old BFS-based algorithms

We start from analyzing the optimizations made to LARPQ algorithm for speeding up simple queries on the synthetic RPQBench dataset. The results are provided in table 1.

As it is observed, the new version of the BFS-based algorithm implementation evaluates simpler queries 5–10 1.5–2.4× faster. Even more, it improves performance of more complex queries such as 1–4 and 20. On the other hand, it demonstrates a slowdown of 1.1–1.4× on such queries as 13, 14, and 19. Such slowdown is rather small and is likely to happen due to the specific query structure for the synthetic RPQBench dataset.



(a) Queries which are supported and succeeded on all of the competitors

(b) Queries which are not supported in Cypher or timed out on FalkorDB or Blazegraph

Figure 1: Wikidata dataset per-query evaluation time

4.4 Reachability algorithms on real-world data

To compare various reachability approaches we start from evaluating all of the competitors on real-world Wikidata. Resulting per-query evaluation time is provided in figure 1a for queries that are supported on all of the competitors. Remaining complex queries ended up with a timeout on slower competitors or not expressible in Cypher are represented separately in figure 1b. The dotted lines represent means and the straight lines represent medians. These numeric values of and total execution time are available in the table 2 for the relatively simple and complex queries (C) separately. For linear algebra-based competitors the size of the dataset loaded in-memory, and memory byte-per-triple (BPT) memory consumption values are provided.

For simple queries, LARPQ demonstrates the best mean with a speedup of 1.7× to 18.9×. A newer version also demonstrates mean improvement of 1.1× against the old one. However, RPQ-matrix implemented with SuiteSparse:GraphBLAS demonstrates the best median time that is better 1.7× than that of LARPQ. It means that RPQ-matrix evaluates some of the queries faster whereas the BFS-based algorithm works better in general cases. Our hypothesis is that the LARPQ algorithm does not utilize the sparsity of some edge kinds enough. Both linear algebra-based approaches demonstrate better time in mean an in median in comparison to all of database management systems.

Table 2: Wikidata query execution results. The results are supplied for simple and for complex (C) queries separately. For in-memory algorithms the memory consumption for whole dataset and bytes-per-triple values (BPT) are presented. Mean and median speedup relatively the proposed solution is a ratio of mean and median over query set for respective systems having LARPQ as a baseline

	LARPQ	LARPQ-old	RPQ-matrix	RPQ-matrix (GrB)	MillenniumDB	FalkorDB	Blazegraph
Total, ms	24 403	27 026	166 882	34 891	276 527	460 606	403 738
Mean, ms	49.8	55.2	340.6	71.2	564.3	940.0	824.0
Median, ms	2.0	2.0	3.3	1.1	13.7	17.6	128.0
Mean speedup	1.00	0.90	0.15	0.70	0.09	0.05	0.06
Median speedup	1.00	1.0	0.60	1.74	0.15	0.11	0.02
Total C, ms	35 061	39 884	508 378	288 766	639 193	—	—
Mean C, ms	427.6	486.4	6 199.7	3 521.5	7 795.0	—	—
Median C, ms	169.1	179.1	372.9	142.9	3 105.6	—	—
Mean speedup C	1.00	0.88	0.07	0.12	0.05	—	—
Median speedup C	1.00	0.94	0.45	1.18	0.05	—	—
Memory, Gb	9.2	9.2	6.9	9.2	—	—	—
BPT	16.3	16.3	12.1	16.3	—	—	—

For complex queries, relations between different competitors are the same except that LARPQ mean is drastically lower than other competitors being 14.5×, 8.3×, 18.3× less than those of the competitors. It means the LARPQ algorithm is capable of executing the most complex out of the complex algorithm faster than other approaches. LARPQ also outperforms its old implementation on complex queries 1.1× which means the suggested optimization is rather successful.

4.5 Linear algebra-based reachability approaches on synthetic data

We continue a detailed comparison of linear algebra-based approaches capability of executing different query kinds by running a synthetic RPQBench. Its evaluation results are available in the table 3. Each row represents distinct query kind. Total execution time of the randomly generated queries are presented in seconds for each competitor separately. Due to the lower overall performance demonstrated on the real-world datasets, MillenniumDB, FalkorDB, and Blazegraph are excluded from subsequent comparisons. The same holds for the old version LARPQ since it is in general slower than the new algorithm. We also provide edge statistics for each label kind in table 4.

As it is observed, the execution time of the proposed BFS-based algorithm for simple queries such as d^*, d^+, and d^*e is quite similar to that of other linear algebra-based implementations. RPQ-matrix ×2 speedups are likely to happen due to the CSR/CSC matrix implementation since RPQ-matrix (GrB) demonstrates evaluation time close to LARPQ.

The greatest performance improvement over other approaches is achieved when the patterns contain compound parts involving dense edges that are not adjacent to the starting or final node. For instance, the single-destination query 20, $(c \mid g)\,(d \mid e)^*$, yields speedups of 9,600× and 2,300×. The same holds for queries 11, 15, and 18.

Table 3: RPQBench dataset evaluation time of queries with randomly generated sources and destinations per each query kind in seconds

	Query pattern, single-source (S) or single-destination (D)	LARPQ	RPQ-matrix	RPQ-matrix (GrB)
1	$a\ b\ c$, (S)	51	83	11
2	$a\ b\ c$, (D)	32	11	12
3	$(a\ b\ c) \mid (c\ d\ d)$, (S)	59	89	24
4	$(a\ b\ c) \mid (c\ d\ d)$, (D)	47	14	22
5	d^*, (S)	22	21	37
6	d^*, (D)	19	19	23
7	$d^*\ e$, (S)	21	12	18
8	$d^*\ e$, (D)	6	3	5
9	d^+, (S)	23	14	29
10	d^+, (D)	18	14	20
11	$(a\ b)^*$, (S)	1 350	30 156	4 904
12	$(a\ b)^*$, (D)	3 619	15 706	2 978
13	$f\ g\ (d \mid e)$, (S)	92 700	5 917	7 745
14	$f\ g\ (d \mid e)$, (D)	2 193	301	486
15	$f\ g\ (d \mid e)^*$, (S)	149 644	2 870 709	2 053 090
16	$f\ g\ (d \mid e)^*$, (D)	6 167	810	1 053
17	$(c \mid g)\ (d \mid e)$, (S)	36	3	7
18	$(c \mid g)\ (d \mid e)$, (D)	16	173 017	41 713
19	$(c \mid g)\ (d \mid e)^*$, (S)	930	55	232
20	$(c \mid g)\ (d \mid e)^*$, (D)	99	955 891	229 035

Table 4: RPQBench edge stats per label

Edge label	Count	Edge label	Count
a	343 660	e	36
b	4 209 447	f	4 928 456
c	114 742 222	g	223 656
d	186		

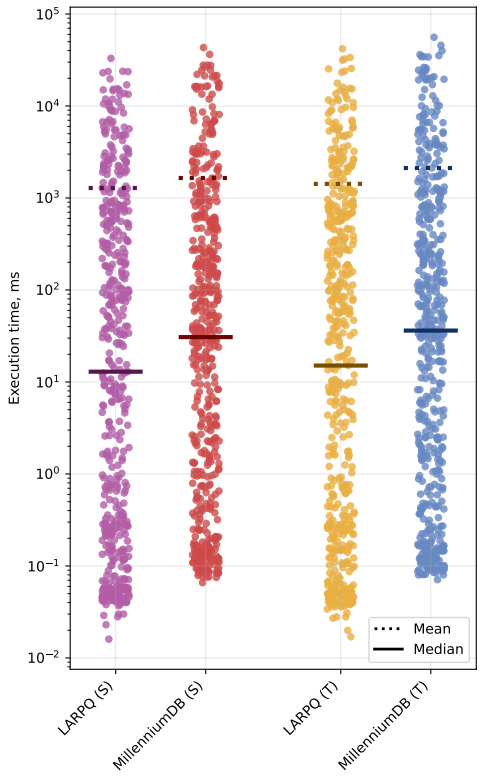


Figure 2: Wikidata execution time under different all paths semantics. (S) denotes all simple paths semantics, (T) denotes all trails semantics

For the remaining queries, such as 1–4, the proposed algorithm demonstrates a relative slowdown of 2× to 4×, since it does not utilize an efficient evaluation order. For queries 14, 16, 17, and 19, LARPQ is 4× to 17× slower than the competitors because it does not exploit the fact that edges with labels d or e are very rare.

Conclusion about various linear algebra-based algorithms might be summarized as follows.

- LARPQ is a better choice for queries that involve compound operations over labels corresponding to many edges and not having source/destination vertices close to them.
- LARPQ algorithm tends to be more stable whereas RPQ-matrix might demonstrate drastic slowdown for some kinds of complex queries.
- Both algorithms are efficient enough for simple queries.
- RPQ-matrix is a better choice if the query contains rare labels, long concatenations or disjunctions allowing to perform optimizations.

Table 5: Wikidata execution time under different all paths semantics. (S) denotes all simple paths semantics, (T) denotes all trails semantics. Mean and median speedup relatively the proposed solution is a ratio of mean and median over query set for respective systems having MillenniumDB as a baseline

	LARPQ (S)	MillenniumDB (S)	LARPQ (T)	MillenniumDB (T)
Total, s	669 249	859 476	742 018	1 103 160
Mean, ms	1 285	1 650	1 424	2 117
Median, ms	13	31	15	36
Mean speedup	1.3	1.0	1.5	1.0
Median speedup	2.4	1.0	2.4	1.0

4.6 All paths semantics evaluation

As mentioned previously, the only competitor that is is MillenniumDB. It is also important to mention, that it has dropped a support of all paths semantics in favor to SPARQL support. Thus, we refer to the old implementation used within [16].

We compare linear algebra-based approach with MillenniumDB under all simple paths and all trails semantics. We drop the queries that have failed or timeouted on any of the competitors and provide per-query evaluation results in figure 2 and numerical data in 5.

As observed LARPQ demonstrates a mean speed up of 1.3–1.5× against DBMS which means linear algebra turns to be a rather perspective competitor against other more classical approaches.

Conclusion

Results of the educational practice can be summarized as follows.

- A set of tools for preparing RDF datasets to be used in a variety of graph-processing approaches has been implemented.
- Linear algebra BFS-based 2-RPQ reachability algorithm performance on simple queries has been approximately 2× improved.
- A detailed comparison of two linear algebra-based algorithms and use cases for each of them has been determined.
  - RPQ-matrix by Diego Arroyuelo at al. is faster for complex 2-RPQs not involving compound statements over labels corresponding to many edges.
  - BFS-based linear algebra approach shows more stable performance in general. It is capable for better evaluation of complex queries.
  - Both approaches are efficient for evaluating simple queries.
- It has been shown that linear algebra is a promising approach for evaluating 2-RPQs under different semantics. Though infrastructure for a complete implementation of such algorithm has not been implemented since it would require sparse matrix libraries to support arbitrary size semirings.

The source code of the project and detailed information on evaluation is available at https://github.com/SparseLinearAlgebra/la-rpq under GPL-2.0 license.

Future work

The future work on the regular path query algorithm will be continued in the following directions in the next term.

- Combining two different linear algebra-based 2-RPQ algorithms to get advantages of both approaches.
- Extending sparse linear algebra libraries with capability to work with arbitrary size semirings.
- Researching of push-pull multiplication optimizations [24] in query evaluation.

References

[1] Angles Renzo. The Property Graph Database Model // Proceedings of the 12th Alberto Mendelzon International Workshop on Foundations of Data Management, Cali, Colombia, May 21-25, 2018 / Ed. by Dan Olteanu, Barbara Poblete. — Vol. 2100 of CEUR Workshop Proceedings. — CEUR-WS.org, 2018. — URL: https://ceur-ws.org/Vol-2100/paper26.pdf.

[2] Angles Renzo, Gutierrez Claudio. The Expressive Power of SPARQL // The Semantic Web - ISWC 2008 / Ed. by Amit Sheth, Steffen Staab, Mike Dean et al. — Berlin, Heidelberg : Springer Berlin Heidelberg, 2008. — P. 114–129.

[3] Arroyuelo Diego, Gómez-Brandón Adrián, Navarro Gonzalo. Evaluating Regular Path Queries on Compressed Adjacency Matrices. — 2024. — 2307.14930.

[4] Belyanin Georgiy, Grigoriev Semyon. Development of linear algebra-based regular path query algorithm. — 2024. — URL: https://se.math.spbu.ru/thesis_download?thesis_id=1403.

[5] Belyanin Georgiy, Grigoriev Semyon. Single-Source Regular Path Querying in Terms of Linear Algebra. — 2024. — 2412.10287.

[6] Casel Katrin, Schmid Markus L. Fine-Grained Complexity of Regular Path Queries // Log. Methods Comput. Sci. — 2023. — Vol. 19, no. 4. — URL: https://doi.org/10.46298/lmcs-19(4:15)2023.

[7] Cypher: An Evolving Query Language for Property Graphs / Nadime Francis, Alastair Green, Paolo Guagliardo et al. // Proceedings of the 2018 International Conference on Management of Data. — SIGMOD '18. — New York, NY, USA : Association for Computing Machinery, 2018. — P. 1433–1445. — URL: https://doi.org/10.1145/3183713.3190657.

[8] Davis Timothy A. Algorithm 1037: SuiteSparse:GraphBLAS: Parallel Graph Algorithms in the Language of Sparse Linear Algebra // ACM Trans. Math. Softw. — 2023. — Vol. 49, no. 3. — P. 28:1–28:30. — URL: https://doi.org/10.1145/3577195.

[9] Farias Benjamín, Rojas Carlos, Vrgoc Domagoj. MillenniumDB path query challenge (short paper) // Proceedings of the 15th Alberto Mendelzon International Workshop on Foundations of Data Management (AMW 2023), Santiago de Chile, Chile, May 22-26, 2023 / Ed. by Benny Kimelfeld, Maria Vanina Martinez, Renzo Angles. — Vol. 3409 of CEUR Workshop Proceedings. — CEUR-WS.org, 2023. — URL: https://ceur-ws.org/Vol-3409/paper13.pdf.

[10] Foundations of Modern Query Languages for Graph Databases / Renzo Angles, Marcelo Arenas, Pablo Barceló et al. // ACM Comput. Surv. — 2017. — sep. — Vol. 50, no. 5. — 40 p. — URL: https://doi.org/10.1145/3104031.

[11] Gutièrrez Claudio. RDF as a Data Model // Fourth IFIP International Conference on Theoretical Computer Science- TCS 2006 / Ed. by Gonzalo Navarro, Leopoldo Bertossi, Yoshiharu Kohayakawa. — Boston, MA : Springer US, 2006. — P. 7–7.

[12] Highly accurate protein structure prediction with AlphaFold / John Jumper, Richard Evans, Alexander Pritzel et al. // Nature. — 2021. — Aug. — Vol. 596, no. 7873. — P. 583–589. — URL: https://doi.org/10.1038/s41586-021-03819-2.

[13] Knowledge Graphs / Aidan Hogan, Eva Blomqvist, Michael Cochez et al. // ACM Comput. Surv. — 2021. — jul. — Vol. 54, no. 4. — 37 p. — URL: https://doi.org/10.1145/3447772.

[14] Szárnyas Gábor, Bader David A., Davis Timothy A. et al. LAGraph: Linear Algebra, Network Analysis Libraries, and the Study of Graph Algorithms. — 2021. — 2104.01661.

[15] Mathematical Foundations of the GraphBLAS / Jeremy Kepner, Peter Aaltonen, David A. Bader et al. // CoRR. — 2016. — Vol. abs/1606.05790. — arXiv : 1606.05790.

[16] MillenniumDB: A Persistent, Open-Source, Graph Database / Domagoj Vrgoc, Carlos Rojas, Renzo Angles et al. // CoRR. — 2021. — Vol. abs/2111.01540. — arXiv : 2111.01540.

[17] Bienvenu Meghyn, Calvanese Diego, Ortiz Magdalena, Simkus Mantas. Nested Regular Path Queries in Description Logics. — 2014. — 1402.7122.

[18] RPQBench: A Benchmark for Regular Path Queries on Graph Data / Hui Wang, Xin Wang, Menglu Ma, Yiheng You // Web Information Systems Engineering – WISE 2024 / Ed. by Mahmoud Barhamgi, Hua Wang, Xin Wang. — Singapore : Springer Nature Singapore, 2025. — P. 351–367.

[19] Reasoning on regular path queries / Diego Calvanese, Giuseppe De Giacomo, Maurizio Lenzerini, Moshe Vardi // ACM SIGMOD Record. — 2003. — 12. — Vol. 32. — P. 83–92.

[20] RedisGraph GraphBLAS Enabled Graph Database / Pieter Cailliau, Tim Davis, Vijay Gadepally et al. // IEEE International Parallel and Distributed Processing Symposium Workshops, IPDPSW 2019, Rio de Janeiro, Brazil, May 20-24, 2019. — IEEE, 2019. — P. 285–286. — URL: https://doi.org/10.1109/IPDPSW.2019.00054.

[21] TASWEET: Optimizing Disjunctive Path Queries in Graph Databases / Zahid Abul-Basher, Nikolay Yakovets, Parke Godfrey et al. // Proceedings of the 20th International Conference on Extending Database Technology, EDBT 2017, Venice, Italy, March 21-24, 2017 / Ed. by Volker Markl, Salvatore Orlando, Bernhard Mitschang et al. — OpenProceedings.org, 2017. — P. 470–473. — URL: https://doi.org/10.5441/002/edbt.2017.47.

[22] Urma Raoul-Gabriel, Mycroft Alan. Source-code queries with graph databases—with application to programming language usage and evolution // Science of Computer Programming. — 2015. — Vol. 97. — P. 127–134. — Special Issue on New Ideas and Emerging Results in Understanding Software. URL: https://www.sciencedirect.com/science/article/pii/S0167642313002943.

[23] Vrgoč Domagoj. Evaluating regular path queries under the all-shortest paths semantics. — 2023. — 2204.11137.

[24] Yang Carl, Buluç Aydin, Owens John D. Implementing Push-Pull Efficiently in GraphBLAS // Proceedings of the 47th International Conference on Parallel Processing, ICPP 2018, Eugene, OR, USA, August 13-16, 2018. — ACM, 2018. — P. 89:1–89:11. — URL: https://doi.org/10.1145/3225058.3225122.