Regular Path Queries in Graph Databases

Lewis Dyer

March 28, 2023

- Formally define Regular Path Queries, and give an example or two.
- Formally introduce the main semantics of regular path queries, defining the main algorithmic questions in play.
- Give some results so far.

1 Preliminaries

As a model for graph databases, we consider directed edge-labelled graphs. Given an alphabet Σ, a graph $G = (V, E)$ consists of a set of vertices V, along with a set of edges $E \subseteq (V \times \Sigma \times V)$, with (u, l, v) denoting at edge from vertex u to vertex b with label l. Note that this model requires that every edge is labeled, and that we may omit the label if this label is irrelevant or clear from context. We further note that loops and multiple edges are permitted in this model.

Given two vertices u and v, a path p is defined by a sequence of edges of the form $(v_0, l_1, v_1), (v_1, l_2, v_2), \ldots, (v_{n-1}, l_n, v_n)$ where $u = v_0$ and $v = v_1$, and we say this path has length n. We denote $\mathcal{P}_(G, u, v)$ as the set of all such paths in G between u and v, occasionally omitting the subscript where this is clear from context. We say a path is *simple* if any two vertices v_i, v_j in p are pairwise distinct.

2 Introducing regular path queries

Given a path p in a graph G and vertices u and v, we consider the word w_p formed in Σ^* by concatenating each edge label in p. More formally, the function $w : \mathcal{P}_{(G,u,v)} \to \Sigma^*$ with $w\left((v_0, l_1, v_1), (v_1, l_2, v_2), \ldots, (v_{n-1}, l_n, v_n)\right) = l_1 l_2 \ldots l_n$ produces the unique word associated with a given path p. Regular path queries (also denoted as RPQs) consist of a regular expression r along with a set of *semantics* for that RPQ, though we frequently refer to an RPQ solely by the regular expression where the semantics being used are clear from context. Evaluating an RPQ over a graph G between vertices u and v consists of two steps: first, we consider paths $p \in \mathcal{P}_{(G,u,v)}$ such that $w(p)$ is accepted by r, and we denote these paths as the set of *candidate paths* of the RPQ. Then, we determine

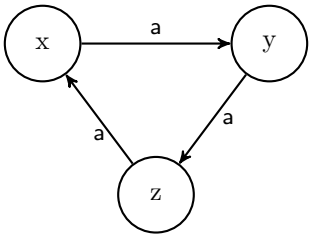

Figure 1: A directed cyclic graph on 3 vertices, where every edge has label a.

which of these candidate paths to output depending on the semantics being used. Three typical sets of semantics are:

- **Arbitrary semantics**, where all candidate paths are returned.
- **Shortest path semantics**, where all candidate paths of minimum length are returned.
- **Simple path semantics**, where all simple candidate paths are returned.

Each of these sets of semantics presents different challenges. When using arbitrary semantics, care must be taken to avoid paths of infinite length, which can occur in cyclic directed graphs. For example, evaluating a^* over the graph in Figure 2 from vertex x to vertex y using arbitrary semantics will result in an infinite set of paths with words given by the set $\{a^{3k+1} | k \in \mathbb{N}\}$.

Shortest path semantics avoid this problem, but can be counter-intuitive even for common queries. For example, when counting the number of paths between two vertices x and y that are accepted by some regular expression r, adding an additional path between x and y can actually reduce the overall count, if this newly added path is accepted by r and is shorter than any other paths.

Simple paths avoid this lack of intuition while also avoiding paths of infinite length, and correspond to many natural constraints on paths in common queries. For example, in a transport network where vertices correspond to locations and edges correspond to different forms of travel between locations, finding simple paths between two locations means that visiting the same location more than once is not permitted.

3 Complexity of evaluating RPQs

To begin, we first formalise the set of evaluation problems on RPQs we are considering. We define the decision RPQ evaluation problem with arbitrary path semantics as follows:

PATH
Input: A graph G with a starting vertex s and an ending vertex e, and a regular expression r.

Output: `true` if there exists a path p in G matching s and r such that $w(p)$ is accepted by r, and `false` otherwise.

Note that this defines the decision problem for RPQ evaluation under arbitrary path semantics. Analogous variants of this decision problem exist for the other forms of semantics introduced in Section 2, which we shall denote as ShortPath and SimPath for shortest path and simple path semantics respectively. Similarly, we may also define enumeration variants of these problems, where all such paths from s to r whose words are accepted by r should be returned. We denote these problems as EnumPaths, EnumShortPaths and EnumSimPaths.

4 Characterising RPQs for simple path semantics

As discussed in Section 3, SimPath and EnumSimPaths are in general intractable over all possible regular expressions, with SimPath being NP-complete with the regular expression $(aa)^*$ and EnumSimPaths being #P-complete with the regular expression a^*. A natural further question is whether a particular class of regular expressions allows for tractable decision and enumeration, and moreover whether this class is sufficiently powerful to handle regular path queries that are used in practice.