Interactive Path Query Specification on Graph Databases

Angela Bonifati Radu Ciucanu Aurélien Lemay
University of Lille & INRIA, France
{angela.bonifati, radu.ciucanu, aurelien.lemay}@inria.fr

ABSTRACT

Graph databases are becoming pervasive in several application scenarios such as the Semantic Web, social and biological networks, and geographical databases, to name a few. However, specifying a graph query is a cumbersome task for non-expert users because graph databases (i) are usually of large size hence difficult to visualize and (ii) do not carry proper metadata as there is no clear distinction between the instances and the schemas. We present GPS, a system for interactive path query specification on graph databases, which assists the user to specify path queries defined by regular expressions. The user is interactively asked to visualize small fragments of the graph and to label nodes of interest as positive or negative, depending on whether or not she would like the nodes as part of the query result. After each interaction, the system prunes the uninformative nodes i.e., those that do not add any information about the user's goal query. Thus, the system also guides the user to specify her goal query with a minimal number of interactions.

1. INTRODUCTION

Graph databases [8] are becoming pervasive in several application scenarios such as the Semantic Web, social and biological networks, and geographical databases, to name a few. Many mechanisms have been proposed to query a graph database, which, although being very expressive, are difficult to understand by non-expert users who are unable to specify their queries with a formal syntax.

The problem of *assisting non-expert users to specify their queries* has been recently raised by Jagadish et al. [6, 7]. More concretely, they have observed that "constructing a database query is often challenging for the user, commonly takes longer than the execution of the query itself, and does not use any insights from the database". While they have mentioned these problems in the context of relational databases, we argue that they become even more difficult to tackle for graph databases. Indeed, graph databases usually do not carry proper metadata as there is no clear distinction between the instances and the schemas. The absence of metadata and the difficulty of visualizing possibly large graphs make unfeasible traditional query specification paradigms for non-expert users e.g., query by example [9].

In this paper, we address the problem of assisting non-expert users to specify their graph queries and propose GPS, "a system for interactive Graph Path query Specification". The user is interactively asked to visualize small fragments of the graph and to label nodes of interest as *positive* or *negative* examples, depending on whether or not she would like the nodes as part of the query result. After each interaction, the system prunes the uninformative nodes i.e., those nodes that do not provide any useful information about the user's goal query. Thus, the system also guides the user to specify her goal query with a minimal number of interactions.

In [2], we have studied the theoretical challenges of such a scenario and empirically shown the improvements of using an interactive approach on biological and synthetic datasets. As a natural extension, we are interested next in applying our algorithms to scenarios where human users provide the positive and negative examples. Both in [2] and in this demo, we focus on the class of path queries defined by regular expressions, where a node is selected if it has a path in the language of a given regular expression. The objective of this demo is thus to let real users interact with GPS to specify different path queries that they could have in mind, while minimizing the number of interactions with the system.

The rest of the paper is organized as follows. In Section 2 we present some key ingredients of our system via a motivating example, while in Section 3 we describe our demonstration scenario. Due to space restrictions, in this paper we provide only a glimpse of the techniques employed by GPS. However, we refer to our full research paper [2] for algorithmic details and for more elements of related work.

2. SYSTEM OVERVIEW

In this section we present a brief overview of our system. To this purpose, we first introduce a *motivating example*. Then, we describe the *interactive scenario* for path query specification on graph databases.

Motivating example

We depict in Figure 1 a graph representing a geographical database having as nodes the neighborhoods of a city area (N_1 to N_6), along with cinemas (C_1 and C_2), and restaurants (R_1 and R_2) in such neighborhoods. The edges represent public transportation facilities from a neighborhood to another (using labels *tram* and *bus*), along with other

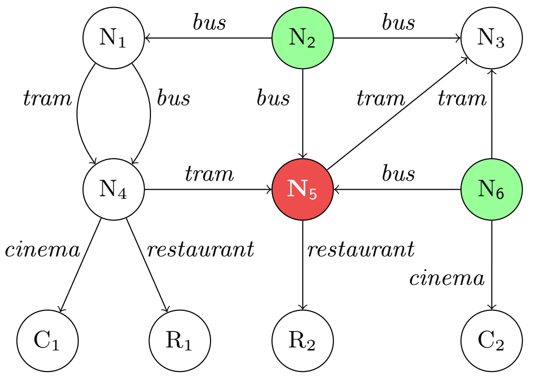


Figure 1: A geographical graph database.

kind of facilities (using labels *cinema* and *restaurant*). For instance, the graph indicates that one can travel by bus between the neighborhoods N_2 and N_3, that in the neighborhood N_4 there is a cinema C_1, and so on. Next, imagine that a user wants to know from which neighborhoods in the city she can reach cinemas via public transportation. These neighborhoods can be retrieved using a *path query* defined by the following *regular expression*:

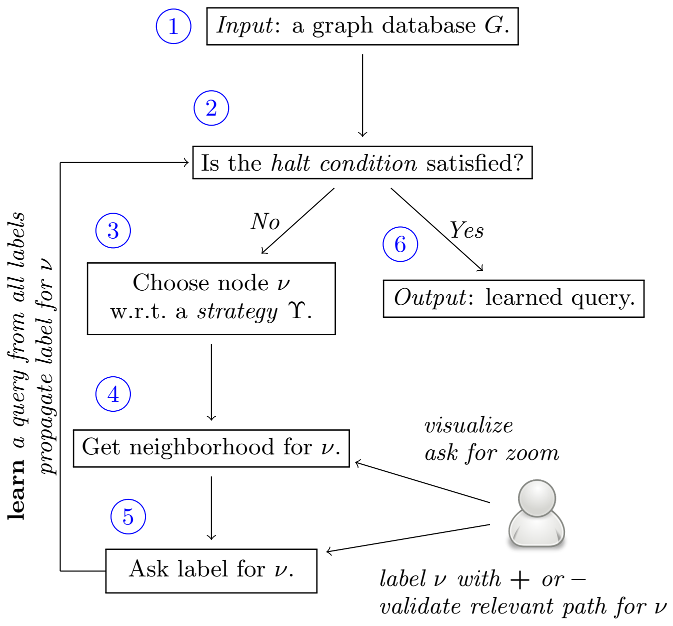
$$q = (tram + bus)^* \cdot cinema$$

The query q selects the nodes N_1, N_2, N_4, and N_6 as they are entailed by the following *paths* in the graph:

$$N_1 \xrightarrow{tram} N_4 \xrightarrow{cinema} C_1,$$
$$N_2 \xrightarrow{bus} N_1 \xrightarrow{tram} N_4 \xrightarrow{cinema} C_1,$$
$$N_4 \xrightarrow{cinema} C_1,$$
$$N_6 \xrightarrow{cinema} C_2.$$

We assume that the user is not familiar with any formal syntax of query languages, while she still wants to specify the above query on the graph database in Figure 1 by providing examples of the query result. In particular, she would positively or negatively label some graph nodes according to whether or not they would be selected by the targeted query. For instance, the user could label the nodes N_2 and N_6 as *positive examples*, and the node N_5 as a *negative example*, thus willing to have all nodes but the last as part of the query result. Indeed, there is no path starting in N_5 through which the user can reach a cinema, while there are paths for the first two nodes. We also observe that the query q above is *consistent* with the user's examples because q selects all positive examples and none of the negative ones.

To construct a query that is consistent with the examples provided by the user, we have proposed in [2] a *learning algorithm* that essentially consists of the following two steps: (i) for each positive example, find a path that is not covered by any negative, and (ii) construct an automaton recognizing precisely the paths found at the previous step and generalize it by state merges while no negative example is covered. By continuing on our running example, take the graph database from Figure 1, the positive examples N_2 and N_6, and the negative example N_5. Assuming that at step (i) we have found the paths *bus·tram·cinema* and *cinema* for N_2 and N_6, respectively, by generalizing the disjunction of these two paths we are able to construct the aforementioned query q, which corresponds to the user's goal query. In the next



1
Input: a graph database G.
2
Is the *halt condition* satisfied?
3
No
Choose node ν w.r.t. a *strategy* Υ.
6
Yes
Output: learned query.
4
Get neighborhood for ν.
5
Ask label for ν.
visualize
ask for zoom
label ν with + or −
validate relevant path for ν
learn *a query from all labels*
propagate label for ν

Figure 2: Interactive scenario.

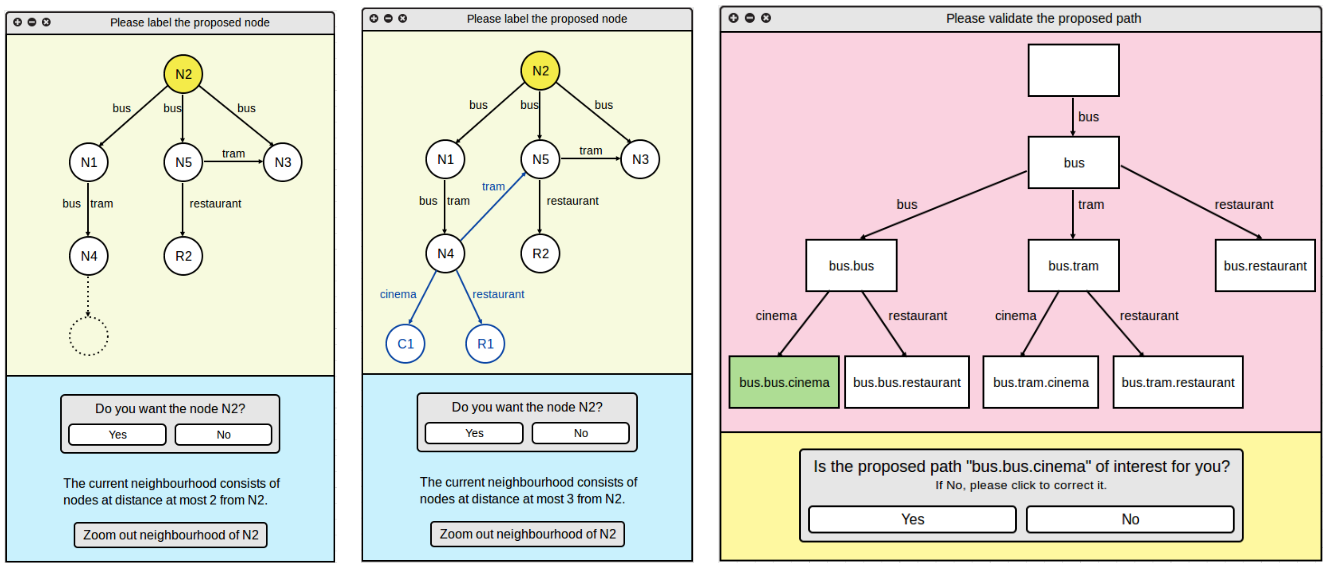
section, while describing the interactive scenario, we explain in more details how our system is able to find path queries via simple user interactions.

Interactive scenario

Even though our demonstration scenario consists of several types of interactions with the user (cf. Section 3), in this section we concentrate exclusively on the core of GPS, the *interactive scenario* for path query specification. This scenario is inspired by the well-known framework of *learning with membership queries* [1]. Recently, we have formalized it as a general paradigm for learning queries on big data [3] and also employed it for learning join queries on relational databases [4, 5]. In Figure 2, we depict the current instantiation for path queries on graphs and we detail next its different steps.

(1) (2) We consider as input a graph database G. Initially, we assume an empty set of examples that we enrich via simple interactions with the user. The interactions continue until a *halt condition* is satisfied. A natural condition is to stop the interactions when there is exactly one consistent query with the current set of examples. However, we also allow weaker conditions e.g., the user may stop the process earlier if she is satisfied by some candidate query proposed at some intermediary stage during the interactions.

(3) We propose nodes of the graph to the user according to a *strategy* Υ i.e., a function that takes as input a graph G and a set of examples S, and returns a node from G. Since our goal is to minimize the amount of effort needed to learn the user's goal query, a smart strategy should avoid proposing to the user those nodes that do not bring any information to the learning process. Intuitively, a node is uninformative if all its paths are covered by negative nodes. A good practical strategy should have two essential properties: (i) be time-efficient i.e., the user does not have to wait too much



Do you want the node N2?

The current neighbourhood consists of nodes at distance at most 2 from N2.

Do you want the node N2?

The current neighbourhood consists of nodes at distance at most 3 from N2.

Is the proposed path "bus.bus.cinema" of interest for you?
If No, please click to correct it.

(a) (b) (c)

Figure 3: Interactions with the user for node labeling and path validation.
(a) Proposing node N_2 to the user and showing the neighborhood of nodes at distance at most 2.
(b) Proposing node N_2 to the user and showing the neighborhood of nodes at distance at most 3.
(c) Proposing a path of node N_2 for user validation and showing all paths of N_2 of length at most 3.

between two consecutive interactions, and (ii) attempt to minimize the number of user interactions by asking the user to only label the most informative nodes. In [2], we have developed such strategies, which intuitively seek the nodes having an important number of paths that are shorter than a fixed bound and not covered by any negative node.

4 5 6 A node by itself does not carry enough information to allow the user understand whether it is part of the query result or not. Therefore, we have to enhance the information of a node by zooming out on its *neighborhood* before actually showing it to the user. This step has the goal of producing a small, easy to visualize fragment of the initial graph, which possibly contains the nodes that the user would label as positive or negative.

In our system, we initially compute the neighborhood of a node ν as the graph consisting of all nodes and edges at distance at most 2 from ν. For instance, given the graph database from Figure 1, let us assume that we want to ask the user to label the node N_2. Thus, the user is first presented with the graph in Figure 3(a). Notice that we have depicted by "..." the parts of the graph that are reachable from the current node N_2, but are not shown because they are not in the current neighborhood. The user can label the proposed node as a positive or negative example (i.e., answer "Yes/No") or she may ask for zooming out the neighborhood to be able to decide whether or not the node is of interest for her. For instance, if the user decides to zoom out the neighborhood of N_2, we show her the graph from Figure 3(b), where we highlight (by drawing in blue) the nodes and edges that have been added w.r.t. the previously presented graph fragment. On our example, the user is able now to see that she can reach a cinema from N_2 and thus to label this proposed node as a positive example.

Next, if the user has labeled a given node as a positive example, we want to find out which of the paths of that node is of interest for her. For this purpose, the system builds all paths of the current node that are not yet covered by negative examples and of length at most the size of the last neighborhood. We present these paths to the user as a prefix tree and we highlight the path that the system believes is the path of interest for her. The user can thus validate this path or correct it further by choosing a different path. In Figure 3(b) of our running example, we have shown the neighborhood of size 3 of the node N_2, which is the node that the user has labeled as a positive example. Consequently, the system shows to the user the paths of N_2 of length at most 3 in the prefix tree of Figure 3(c). The system highlights the path *bus·bus·cinema* as a candidate path of interest for the user because (i) it has length equal to 3 and (ii) the system inferred that a path of this length better fits the user's will as the latter zoomed out the neighborhood of length 2 in Figure 3(a).

After path validation by the user, the system seamlessly propagates to the rest of the graph the labels provided by the user at this stage, while at the same time pruning the nodes that become uninformative. Our learning algorithm outputs in polynomial time either a query q consistent with all labels provided by the user, or instead the next node to label if such a query cannot be constructed efficiently. We have shown in [2] that constructing in polynomial time a query consistent with the examples is not always possible, but, after a certain number of examples (this number being polynomial in the size of the query), the learning algorithm is guaranteed to return in polynomial time a query equivalent to the user's goal query.

When the halt condition is satisfied, we return the latest output of the learning algorithm to the user. In particular, the halt condition may take into account an intermediary learned query q e.g., when the user is satisfied by the output of q on the instance and wants to stop the interactions.

3. DEMONSTRATION SCENARIO

The demonstration scenario consists of three parts. First, we would like that the attendee appreciates the difficulties that one can encounter when labeling directly the graph database instance. Next, we propose to the attendee an interactive scenario where she is prompted with small fragments of the graph that can be easily visualized. On these fragments, the user can label nodes as positive or negative examples and the system infers her queries. Third, we present the core of our system, where we additionally show to the user the set of relevant paths entailed by the positive examples for further validation. This extra step guarantees that the system generalizes the interesting paths for the user and thus the constructed query indeed corresponds to what the user had in mind. In the demo, we plan to show our algorithms on real geographical data. Such data combines the information about networks of public transportation in France (e.g., Transpole[1]) with other facilities in the spirit of our motivating example.

Static labeling

To illustrate why it is important to use an interactive approach in proposing nodes of interest to the user, we progressively show to the attendees the different types of interactions our system can handle. In this first part of the demonstration, we let the attendee visualize the graph and label nodes of interest in the order she prefers. Then, the system proposes a query consistent with the provided examples or, alternatively, points out that the labeled nodes are inconsistent. The attendee must observe that this kind of approach is not user-friendly as the user is asked to (i) possibly visualize a large graph database instance and (ii) inspects interesting fragments by herself. This clearly requires more effort than visualizing small fragments of the graph and simply answering "Yes/No" to nodes proposed by the system. However, we think that it is still important to show this static labeling scenario to the user to appreciate the differences with respect to the interactive scenario, which we discuss next. Moreover, the static labeling scenario is the only one where we let the user to make mistakes by labeling nodes inconsistently because in the other scenarios we show informative nodes only hence any labeling is consistent.

Interactive labeling (without path validation)

In this part of the demonstration, we present the interactive scenario illustrated in the previous section, but without including the step of path validation. For instance, at each interaction the system computes the most informative node and shows it to the user together with its neighborhood as in Figure 3(a). Then, the user may label it as a positive or negative example, or she can ask for zooming out the neighborhood, which yields a graph as in Figure 3(b). When the user labels a proposed node as a positive example, the system computes for that node a path that is not covered by any negative example, the latter path being used by the learning algorithm afterwards. The goal of this scenario is to show the importance of the step of path validation. Although the interactive scenario without this step finally produces a query that is consistent with the examples provided by the user, this query is not necessarily always the query that the user expects. As an example, on the graph and the labeled examples in Figure 1, notice that the query *bus* selects both positive examples C_2 and C_6, and not the negative example C_5. This query is clearly not the user's goal query. Therefore, even though the user has to perform an additional click to validate or to correct the path of interest for a positive node, this step is necessary to make sure that the learned query is constructed using for each positive node the paths of interest for her.

[1]`http://www.transpole.fr/`

Interactive labeling (with path validation)

In this last part of the demonstration, we illustrate the core of our system i.e., the interactive scenario described in Section 2. As a difference w.r.t. the second demonstration scenario, the user can now additionally choose as in Figure 3(c) the path of interest instead of letting the learning algorithm choose such a path. The goal of this third scenario is to show the actual difference between "learning" a query that is consistent with the node examples provided by the user and assisting the user to "specifying" her query, also via node examples. When the user also validates the paths of interest for each positive node, this guarantees that the system generalizes the interesting paths for the user and the constructed query is indeed the user's goal query. In conclusion, by using GPS, a non-expert user desiring to query a graph database has neither to visualize all the graph that can be potentially large, nor to look by herself for interesting nodes, as the system guides her throughout small, easy to visualize fragments of the graph and prompt her with nodes to label on these fragments.

4. REFERENCES

[1] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1988.

[2] A. Bonifati, R. Ciucanu, and A. Lemay. Learning path queries on graph databases. In *EDBT*, 2015.

[3] A. Bonifati, R. Ciucanu, A. Lemay, and S. Staworko. A paradigm for learning queries on big data. In *Data4U*, pages 7–12, 2014.

[4] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive inference of join queries. In *EDBT*, pages 451–462, 2014.

[5] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive join query inference with JIM. *PVLDB*, 7(13):1541–1544, 2014.

[6] H. V. Jagadish, A. Chapman, A. Elkiss, M. Jayapandian, Y. Li, A. Nandi, and C. Yu. Making database systems usable. In *SIGMOD Conference*, pages 13–24, 2007.

[7] A. Nandi and H. V. Jagadish. Guided interaction: Rethinking the query-result paradigm. *PVLDB*, 4(12):1466–1469, 2011.

[8] P. T. Wood. Query languages for graph databases. *SIGMOD Record*, 41(1):50–60, 2012.

[9] M. M. Zloof. Query by example. In *AFIPS National Computer Conference*, pages 431–438, 1975.