Semantics for querying paths in graph databases: No-repeated-node or no-repeated-edge?

Daniel Hernández and Claudio Gutierrez

Computer Science Dept & CIWS, Universidad de Chile

Abstract. Patterns for walks in graphs usually include regular expressions, that may result in loops, thus producing infinite solutions. To overcome this issue, practical query languages bound result sets to ensure finiteness. On this paper we compare two bounding methods: edges are visited at most once; nodes are visited at most once. We show that despite these methods produce languages with different expressiveness, there is a duality, in the sense that the queries in one can be evaluated using the machinery used to evaluate the other.

1 Introduction

Most graph query languages support querying for walks (or paths) between nodes by using some fragment of regular path queries. Consider the expression ue^*v, where u, v are node and e is an edge; it can be considered a pattern that represents some type of query involving paths from u to v over edges with label e. The precise semantics for such query, though, has many possibilities. Some of them (see [1]) are the following: true if v is reachable from u through a path of edges with label e; the pairs of nodes (u, v) with the same previous condition; some of the walks from u to v fulfilling the above condition; the graphs that are induced by such walks, etc. In this paper we study the problem of returning some walks.

An issue when returning walks is that there are possibly infinitely many when allowing loops. Thus, for practical concerns current implementations limit the number of walks returned by choosing the shortest ones; or choosing those that do not repeat nodes; or those that do not repeat edges; or choosing the k-top "best" walks.

In this paper we will study two strategies: not repeating nodes and not repeating edges. The first is preferred in graph matching applications (see, e.g., [2]) where no nodes on the graph should be collapsed and was recently used in the context of property graph for pattern matching in social networks [3–5]. The second semantics is currently used by the Cypher query language [6].

Organization of the paper. In Sec. 2 we present the graph and query model. In Sec. 3 we compare the expressiveness of query languages induced by the two strategies and show how a machinery designed to evaluate queries in one language can be used for the other and viceversa. This is relevant to see when some languages can be used to simulate patterns of another.

2 Graph and query model

For our goals, it will be sufficient to work over a simple graph data model without direction on edges, and without properties in nodes and edges.

We assume the existence of three infinite disjoint sets $\mathbf{V}$, $\mathbf{E}$, and $\mathbf{X}$ denoting respectively nodes, edges, and variables. A *graph database* G (a *graph* in that follows) is a set (V, E, δ), where $V \subset \mathbf{V}$, $E \subset \mathbf{E}$, and $\delta : E \to [V]^2$ is a function that maps each edge $e \in E$ to a subset $\{u, v\} \in [V]^2$. Observe that according this model a graph database is a multigraph.

A *join* in a graph G, denoted as $[aub]$, is an unordered pair of edges a, b with a common endpoint u. Think of it as the pair $(u, \delta(u))$. Observe that a join is unordered (i.e., $[aub] = [bua]$).

A *walk* w in a graph G is an alternating sequence of nodes and edges $v_1e_1 \dots e_{n-1}v_n$ (i.e. beginning and ending in a node) such that $\delta(e_j) = \{v_j, v_{j+1}\}$. A *connection* c between two nodes u_1, u_2 in G is an alternating sequence of edges and nodes $e_1v_n \dots v_{n-1}e_n$ (i.e. beginning and ending with an edge) such that u_1cu_2 is a walk in G. A *walk pattern* P (a *pattern* in that follows) is a walk where some edges are replaced by variables. We will denote nodes and edges with lower case letters, and variables with upper case letters. For instance, $uXveu$ is a pattern where u, v are nodes, e is an edge, and X is a variable.

A *solution* of the evaluation of a pattern P on a graph G is a mapping μ from the variables occurring in P to connections in G such that the result, denoted as $\mu(P)$, is a walk obtained from replacing every variable X by $h(X)$ in P. We denote the set of all solutions of P on G as $[\![P]\!]_G$. Formally, a solution in $[\![P]\!]_G$ is an homomorphism from P to connections on G such that $h(P)$ is a walk in G. For example, consider the pattern $P = uXv$, and the graph G depicted by Fig. 1. Then, $\mu_1 : X \mapsto c$, and $\mu_2 : X \mapsto awb$ are solutions in $[\![P]\!]_G$, because $\mu_1(P) = uawbv$ and $\mu_2(P) = ucv$ are a walks in G.

3 No-repeated-node versus no-repeated-edge semantics

The semantics of patterns has problems when resulting sets may contain infinitely many solutions. For instance, evaluating the pattern uXv over the graph G depicted in Fig. 1 produces infinitely many solutions because there are infinitely many walks between u and v.

Most graph databases bound solution sets in some form [1]). We study two of the most common, based respectively in restricting solutions μ such that each node (respectively each edge) occurs at most once in $\mu(P)$. We will denote them as $[\![P \mid_V]\!]_G$ (respectively $[\![P \mid_E]\!]_G$), and we call them *non-repeating-node* and *non-repeating-edge* semantics [1].

It is easy to see that $[\![P \mid_V]\!]_G \subseteq [\![P \mid_E]\!]_G$ for every pattern P and a graph G. In fact, a walk w repeating and edge e with endpoints $\{u, v\}$ has to repeat at least v or u. If w does not repeat u, then w contains $veuev$, thus w repeats v. The converse is not true: a counterexample is the walk $ucvduev$ of the graph depicted in Fig. 1 that visits twice the nodes u and v without repeating edges. A corollary of this is that given an arbitrary pattern P, in general there does not exist a pattern P' such that $[\![P|_E]\!]_G = [\![P'|_V]\!]_G$ for every graph G. Furthermore, it is easy to see that this result is also true for the equality $[\![P|_V]\!]_G = [\![P'|_E]\!]_G$. Hence, these two semantics induce languages with different expressiveness. Moreover, they are incomparable, that is, no one is more expressive than the other.

Despite both semantics induce incomparable languages, we will show that the result of one semantics can be obtained using the machinery that compute the results of the other.

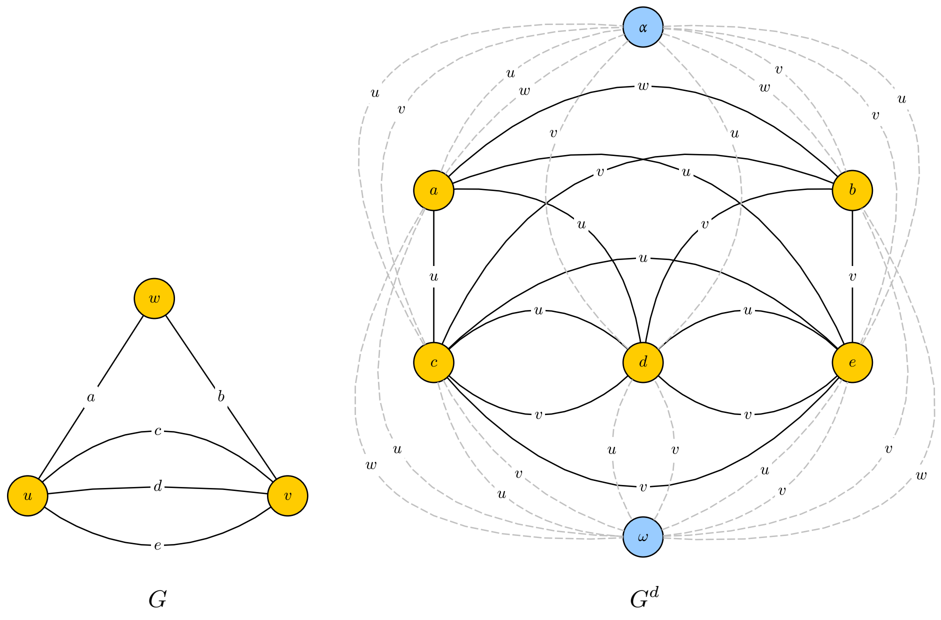

Fig. 1. A graph database G and its dual G^d.

We assume the existence of an injective function $(\cdot)^d$ that maps every edge to a node, and every join to an edge. Also, given a graph G, we denote by G' its dual graph: every node u' in G' correspond to an edge e in G such that $u' = e^d$, and every edge e' in G' correspond to a join $[aub]$ in G such that $e' = [aub]^d$, and the endpoints of e' in G' are a^d and b^d. Then, we will write G^d to denote the result of adding two fresh nodes α and ω to G' and connecting them two every node in G'. Abusing notation, we will call G^d the *dual* of G. For example, the graph G^d depicted by Fig. 1 is the dual of G[1].

The *dual* of a walk $w = u_1e_1 \dots e_{n-1}u_n$, denoted as w^d, will be the walk $\alpha[\alpha u_1 e_1]^d e_1^d \dots e_{n-1}^d [e_{n-1}u_n\omega]^d\omega$ in G^d.

Extending the notion of *duality* for patterns is more involving than for walks. Indeed, given the pattern uXv, then it is natural to assume that $(uXv)^d$ is the pattern $\alpha[\alpha u Y_1]^d X [Y_2 v\omega]^d\omega$, where Y_1 and Y_2 are two variables that represent

[1] For the sake of the readability, in G^d we denote a^d and $(awb)^d$ simply as u and w, respectively, because the actual symbols can be inferred from the figure.

edges holding that solutions μ of this dual pattern satisfy $\mu(X) = \mu(Y_1) \dots \mu(Y_2)$. This translation introduces the expressions $[\alpha, u, Y_1]^d$ and $[Y_2, v, \omega]^d$ whose semantics are not already formalized.

We will define the expression $[\alpha, u, Y]^d$ as the set of all possible replacements of Y by an edge (i.e., $\{[\alpha u e]^d \mid e \in \mathbf{E}\}$). Similarly, given a pattern P, then P^d will represent the set, denoted as $\mathrm{repr}(P^d)$, including every possible result of replacing each edge variable Y occurring in P^d by an edge $e \in \mathbf{E}$.

Now we are ready to present our main result.

Theorem 1 (main). *Let P be a pattern and G be a graph database. Then:*

$$[\![P \mid_E]\!]_G = \bigcup_{P' \in \mathrm{repr}(P^d)} \{\mu^{d^{-1}} : \mu \in [\![P' \mid_V]\!]_{G^d}.$$

Proof sketch. Each walk w in G has a corresponding walk w^d in G^d. Thus, for every solution $\mu \in [\![P \mid_E]\!]_G$ there is a walk w_μ such that $(w_\mu)^d$ is in G^d. It suffices proving that there is a pattern P' in $\mathrm{repr}(P^d)$ such that $(w_\mu)^d$ is a solution of evaluating $[\![P' \mid_V]\!]_G$. Determining P' can be done by choosing the representation of P^d that maps the edge variables to the values according to what μ maps around its respective sequence variables. For instance, let P be the pattern uXv and consider the graphs G and G^d depicted by Fig. 1. Then, a solution of $[\![P|_E]\!]_G$ is the mapping $\mu : X \mapsto awbvcue$ because $\mu(P) = uawbvcuev$ is a walk in G^d. The pattern P^d is $\alpha[\alpha u Y_1]^d X [Y_2, v, \omega]^d \omega$. A pattern $P' \in \mathrm{repr}(P^d)$ matching the ends of $\mu(X)$ is $\alpha[\alpha u a]^d X [e v \omega]^d \omega$. The mapping μ^d is a solution of $[\![P' \mid_V]\!]$. Hence, P' is the pattern needed. $\square$

Computing $\bigcup_{P' \in \mathrm{repr}(P^d)} [\![P' \mid V]\!]_G$ does not require to evaluate the infinitely many patterns in $\mathrm{repr}(P^d)$. In fact, it suffices to check substitutions producing not trivially empty results. For instance, $[\alpha u Y]^d$ is not trivial only for edges whose endpoints include u in G. As a consequence of this optimization, for data complexity, the time required for evaluating $[\![P \mid_E]\!]_G$ with the method of Theo. 1 is in the same complexity class than the one required to compute $[\![P \mid_E]\!]_G$ directly, and the space grows to at most $O(|E|^2)$

Finally, let us say that the dual result (*i.e.* evaluate $[\![\cdot \mid_V]\!]_G$ using $[\![\cdot \mid_E]\!]_G$) uses similar techniques and we do not show it here for space reasons.

References

1. R. Angles, M. Arenas, P. Barcelo, A. Hogan, J. Reutter, D. Vrgoc. *Foundations of Modern Graph Query Languages.* CoRR abs/1610.06264, 2016.
2. Horst Bunke. *Graph matching: Theoretical foundations, algorithms, and applications.* In Proceedings of Vision Interface, 2000.
3. Wenfei Fan, Jianzhong Li, Shuai Ma, Nan Tang, Yinghui Wu, and Yunpeng Wu. *Graph Pattern Matching: From Intractable to Polynomial Time.* PVLDB, 2010.
4. Wenfei Fan. *Graph pattern matching revised for social network analysis.* In 15th International Conference on Database Theory (ICDT), 2012.
5. Wenfei Fan, Xin Wang, and Yinghui Wu. *Incremental graph pattern matching. ACM Trans.* Database Systems, 2013.
6. The Neo4j Team. *The Neo4j Manual v3.0.* http://neo4j.com/docs/stable/. 2016.