A comparison between Cypher and conjunctive queries

Jaime Castro and Adrián Soto

Pontificia Universidad Católica de Chile, Santiago, Chile

1 Introduction

Graph databases are one of the most popular type of NoSQL databases [2]. Those databases are specially useful to store data with many relations among the entities, like social networks, provenance datasets or any kind of linked data. One way to store graphs is property graphs, which is a type of graph where nodes and edges can have attributes [1].

A popular graph database system is Neo4j [4]. Neo4j is used in production by many companies, like Cisco, Walmart and Ebay [4]. The data model is like a property graph but with small differences. Its query language is Cypher. The expressive power is not known because currently Cypher does not have a formal definition. Although there is not a standard for querying graph databases, this paper proposes a comparison between Cypher, because of the popularity of Neo4j, and conjunctive queries which are arguably the most popular language used for pattern matching.

1.1 Preliminaries

Graph Databases. Graphs can be used to store data. The nodes represent objects in a domain of interest and edges represent relationships between these objects. We assume familiarity with property graphs [1]. Neo4j graphs are stored as property graphs, but the nodes can have zero or more labels, and edges have exactly one type (and not a label). Now we present the model we work with.

A Neo4j graph G is a tuple $(V, E, \rho, \Lambda, \tau, \Sigma)$ where:

1. V is a finite set of nodes, and E is a finite set of edges such that $V \cap E = \emptyset$.
2. $\rho(\cdot, \cdot, \cdot) \subseteq V \times E \times V$ is a relation such that if $(v_1, e, v_2) \in \rho$ there exists a directed edge e from node v_1 to v_2.
3. $\Lambda(\cdot, \cdot) \subseteq V \times Lab$ is a relation such that if $(x, l) \in \Lambda$ then the label of x is l. Λ is a relation because the Neo4j documentation states that a node can have multiple labels. Lab denotes a set of labels.
4. $\tau(\cdot) : E \to Lab$ is a total function that maps every edge to its type.
5. Σ is a set of partial functions $\sigma_{property} : (V \cup E) \to Lab$. Each function in the set is such that if $\sigma_p(v) = l$, then the property p in v has the value l.

Conjunctive queries. We assume familiarity with first order logic and conjunctive queries (CQ). In this paper CQs are stated using the vocabulary of Neo4j graphs, that is, using predicates $(V, E, \rho, \Lambda, \tau, \Sigma)$ and may also use equalities between terms. Conjunctive queries with inequalities ($CQ^{\neq}$) adds to CQs the posibility of using inequalities between terms, and conjunctive queries with negation ($CQ^{\neq}_{\neg}$) extend $CQ^{\neq}$ with negated relational atoms.

2 Pattern matching in Cypher

Roughly, a Cypher query describes a pattern we want to find in a graph. The core of Cypher is based on `MATCH` and `RETURN` statements. In order to give the syntax and semantics for the core of Cypher, we need to define a **basic graph pattern** (**bgp**). Our definition of basic graph patterns is founded on the given in [1], but Neo4j also has undirected edges in its query language.

Definition 1. *Let Var be a set of symbols representing variables, and Lab be a set of strings representing labels. A **Basic Graph Pattern** (**bgp**) is defined as as a tuple $(V', E', \rho', \bar{\rho}', \Lambda', \tau', \Sigma')$. Each element in the tuple is explained below.*

1. $V' \subseteq Var, E' \subseteq Var$, where $V' \cap E' = \emptyset$.
2. $\rho'(\cdot, \cdot, \cdot) \subseteq V' \times E' \times V'$ is a relation that represents directed edges.
3. $\bar{\rho}'(\cdot, \cdot, \cdot) \subseteq V' \times E' \times V'$ is a relation that represents undirected edges.
4. $\Lambda'(\cdot, \cdot) \subseteq V' \times Lab$ is a relation that represent labels.
5. $\tau' : E' \to Lab$ is a total function that maps every edge to the respective type.
6. Σ' is a set of partial functions $\sigma'_{property} : (V' \cup E') \to Lab$, where each one assigns properties to elements. For example, if $\sigma'_p(x) = l$ then the property p of x has the value l.

The semantic for a **bgp** is defined using **mappings**. Intuitively a mapping M represents an homomorphism from the **bgp** to the Neo4j graph, where constants are mapped to themselves and variables are mapped to elements in the domain. When M is applied to Q, the result is a sub-graph of G. The semantic used in Cypher is a *No-repeated-edge* semantics [1], where "edge variables" must be assigned one-to-one, and the nodes, labels, attribute, properties and values do not need to be injective. Thus, the **result** for a single **bgp** is the set S of mappings from Q to G with an injective assignment of edges.

MATCH queries. In Cypher, the way to declare a pattern that we want to find in a graph is to declare a `MATCH` query, which is defined below.

Definition 2. *Let $bgp_1, \ldots, bgp_n$ be basic graph patterns. A `MATCH` query is an expression with the form `MATCH` bgp_1 ... `MATCH` bgp_n `RETURN` $v_1, \ldots, v_m$.*

The **semantics** for a pattern matching query in Cypher is defined as to apply the `RETURN` expression over the set of mappings S resulting from the `MATCH`

expression. Let S_i be the set of mappings satisfying the basic graph pattern bgp_i. The result of the expression `MATCH` bgp_1 ... `MATCH` bgp_n is $S_1 \bowtie \ldots \bowtie S_n$[1], where the join $\bowtie$ is defined over the notion of compatible mappings [5].

The result of the `RETURN` $v_1, \ldots, v_m$ expression applied over the set of mappings S is defined as the inclusion of only the variables $v_1, \ldots, v_m$ for each mapping in S (i.e. is the projection of such variables). Also in Cypher it is possible to have unnamed variables that will not be returned. This behaviour can be captured by variables in the **bgp** that are not returned.

3 Comparision with conjunctive queries

In this chapter we provide our results about the comparison between conjunctive queries and pattern matching in Cypher. First we show an example of how a conjunctive query looks like.

Example 1. The following query represents all nodes receiving an edge from a node with label 'Montevideo'.

$$\exists x_1 \exists y_1 V(x_1) \wedge E(y_1) \wedge V(x_2) \wedge \rho(x_1, y_1, x_2) \wedge \\ \Lambda(x_1, \text{'City'}) \wedge \sigma_{name}(x_1) = \text{'Montevideo'}$$

Proposition 1. *For each conjunctive query over the vocabulary* $(V, E, \rho, \bar{\rho}, \Lambda, \tau, \Sigma)$*, there exists an equivalent* `MATCH` *query.*

The Proposition 1 holds because an answer for a **bgp** uses the *No-repeated-edge* semantic. For that reason we cannot transform the conjunctive query into a Cypher query with a single `MATCH`. So for each appearance of $\rho(x_1, x_2, x_3)$ in the formula, we need a `MATCH` statement with a different **bgp**.

Proposition 2. *There exists a* `MATCH` *query that is not equivalent to any conjunctive query over the vocabulary* $(V, E, \rho, \bar{\rho}, \Lambda, \tau, \Sigma)$*.*

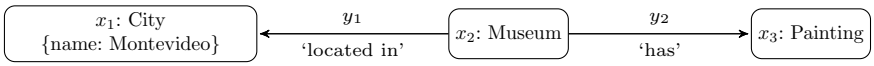
Since conjunctive queries preserves homomorphisms and a `MATCH` query does not preserve homomorphisms, there exists queries which are not expressible as conjunctive queries. Then we need to add inequalities to conjunctive queries, in order to express all `MATCH` queries as a conjunctive query.

Proposition 3. *For each* `MATCH` *query, there exists an equivalent conjunctive query with inequalities over the vocabulary* $(V, E, \rho, \bar{\rho}, \Lambda, \tau, \Sigma)$*.*

The formula that represents the conjunctive query enforces each one of the necessary conditions to satisfy the `MATCH` query, including the *No-repeated-edge* semantics. This is because now we can explicitly demand that edges must be different in the conjunctive query. Now we present an example.

[1] Note that a query with more than one `MATCH` is useful to avoid the constraint of no repeat edges

Example 2. To obtain the paintings in museums located in "Montevideo", we create the following **bgp** *painting_bgp*:



The query is `MATCH` *painting_bgp* `RETURN` x_3, and its equivalent CQ is:

$$\exists x_1 \exists x_2 \exists y_1 \exists y_2 \; V(x_1) \wedge V(x_2) \wedge V(x_3) \wedge E(y_1) \wedge E(y_2) \wedge y_1 \neq y_2 \wedge \rho(x_2, y_1, x_1) \\ \wedge \rho(x_2, y_2, x_3) \wedge \Lambda(x_1, \text{'City'}) \wedge \Lambda(x_2, \text{'Museum'}) \wedge \Lambda(x_3, \text{'Painting'}) \\ \wedge \tau(y_1) = \text{'located in'} \wedge \tau(y_2) = \text{'has'} \wedge \sigma_{name}(x_1) = \text{'Montevideo'}$$

4 Adding inequalities and negation

Since we added inequalities to CQs in order to express `MATCH` queries, we wanted to know the results of the comparison when `MATCH` queries have also inequalities. This can be done in Cypher through the `WHERE` statement. This statement filters or puts a constraint to the patterns, which can be any inequality or equality between labels or variables, ask for a **bgp** representing an eulerian path[2], a restriction for only selecting nodes with a fixed label, or a negation[3].

MATCH - WHERE queries. A `MATCH - WHERE` query in Cypher has at least one `MATCH` statement, a single `WHERE` statement and a single `RETURN`. The definition is as follows.

Definition 3. *Let $bgp_1, ..., bgp_n$ be **bgps**. A `MATCH - WHERE` query is an expression with the form `MATCH` bgp_1 ... `MATCH` bgp_n `WHERE` $\Phi_1 \ldots \Phi_p$ `RETURN` $v_1, \ldots, v_m$.*

The intuition is that only mappings that comply all the restrictions are considered. The `RETURN` statement is applied as before. Also, it is quite straightforward to construct a conjunctive query with inequalities and negation from a `MATCH - WHERE` query. Our main result is the following:

Theorem 1. *`MATCH - WHERE` queries without negation are equivalent to conjunctive queries with inequalities over the vocabulary $(V, E, \rho, \bar{\rho}, \Lambda, \tau, \Sigma)$.*

Considering that any conjunctive query can be expressed as a `MATCH` query, it is not hard to note that if we add inequalities to a query, these can be written in the `WHERE` statement. If we include negation to the Cypher queries and to the conjunctive queries the result does not hold in one direction.

Proposition 4. *Each `MATCH - WHERE` query (including negation and inequalities) can be expressed as a conjunctive query with inequalities and negation over the vocabulary $(V, E, \rho, \bar{\rho}, \Lambda, \tau, \Sigma)$.*

[2] We use the eulerian path restriction because Neo4j forces the **bgps** in `WHERE` clauses to be written in a single statement.

[3] Note that Cypher permits more constraints. In particular, Cypher allows nesting of boolean conditions, which will be studied in a future work.

In the other direction, the equivalence is weaker.

Proposition 5. *Each query in* $CQ^{\neq}_{\neg}$ *can be expressed as a* `MATCH - WHERE` *query, as long as their negated statements ask for graphs (regardless the direction of the edges) representing an eulerian path.*

Finally, we conjecture that there exists a conjunctive query with inequalities and negation over the vocabulary $(V, E, \rho, \bar{\rho}, \Lambda, \tau, \Sigma)$ that is not equivalent to any `MATCH - WHERE` query. The intuition behind is that in Cypher it is impossible to specify a **bgp** without an eulerian path in the `WHERE` statement, while a CQ does not have this restriction. We want to prove such proposition in future work.

5 Conclusions

The comparison between Cypher and conjunctive queries can be a starting point for a future formalization for that query language. Despite the difference between the *No-repeated-edge* semantic of Cypher and the homomorphisms of conjunctive queries, the core of Cypher based on the `MATCH`, `WHERE` without negation, and `RETURN` statements was proved to be equivalent to conjunctive queries with inequalities. However, the non existence of an equivalence between Cypher with respect to conjunctive queries with union, intersection and difference is a disadvantage respect to other languages that are defined over a formal model.

As future work we will study all the open problems presented through the paper. Also, we are interested in studying another Neo4j functions not covered by this work, for instance, path functions. It is not clear the semantics given to a path function in Cypher, but paths are one of the most distinctive features of this query language with respect to other ones. We would like to compare path queries against regular path queries, since regular path queries are considered the core of languages made for querying semistructured data [3].

References

1. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern graph query languages. CoRR abs/1610.06264 (2016), http://arxiv.org/abs/1610.06264
2. Angles, R., Gutierrez, C.: Survey of graph database models. ACM Computing Surveys (CSUR) 40(1), 1 (2008)
3. Calvanese, D., De Giacomo, G., Lenzerini, M., Vardi, M.Y.: Reasoning on regular path queries. SIGMOD Record 32(4), 83–92 (2003), http://doi.acm.org/10.1145/959060.959076
4. Neo Technology, I.: Neo4j, the world's leading graph database (2016), http://neo4j.com
5. Pérez, J., Arenas, M., Gutierrez, C.: Semantics and Complexity of SPARQL, pp. 30–43. Springer Berlin Heidelberg, Berlin, Heidelberg (2006), http://dx.doi.org/10.1007/11926078_3