Linear Recursion in G-CORE

Valentina Urzua and Claudio Gutierrez

Department of Computer Science, Universidad de Chile and IMFD.

Abstract. G-CORE is a query language with two key characteristics: It is closed under graphs and incoporates paths as first-class citizens. Currently G-CORE does not have recursion. In this paper we propose this extension and show how to code classical polynomial graph algorithms with it.

Keywords: G-CORE, Recursion, Graph Query Language

1 Introduction

G-CORE is a graph database query language designed by a working group belonging to the *Linked Data Benchmark Council*[1]. One of its most novel features is the ability to express paths as first-class citizens. However, path queries are still not enough to express a wide variety of "natural" queries, particularly classical graph algorithms like topological sort, BFS, Eulerian circuit, etc.

We propose to extend G-CORE with (linear) recursive functionalities by introducing a syntax and a semantics that essentially follow the ideas of similar constructs in SQL and SPARQL[1]. What is more novel is that we take the core basic polynomial algorithms in graph theory and show, first, that they cannot be coded in standard G-CORE; and second, how to write queries that code them in G-CORE extended with linear recursion. Finally, we tested the code in an evaluator for G-CORE that is being developed by Roberto García at the University of Talca.

Basic Notions. A *Path Property Graph* (PPG) is an edge and node labeled graph where edges and nodes additionally have property-value pairs. In addition, a PPG may also have a collection of paths, where a path is a concatenation of existing, adjacent, edges in the graph.

A basic *G-CORE query* is an expression of the form:

$$\texttt{CONSTRUCT}\ f\ \texttt{MATCH}\ \gamma\ \texttt{ON}\ G\ \texttt{WHERE}\ \xi \tag{1}$$

where f is a full construct pattern, γ is a full graph pattern and ξ a Boolean condition [2]. The core of a G-CORE query consists in the complete graph pattern γ that defines the content of the `MATCH` clause. The `MATCH` clause is evaluated against the PPG G and returns a set of bindings that are filtered with the `WHERE` clause, to finally build a new PPG H with the `CONSTRUCT` clause.

[1] Ongoing research

Related Work. Adding recursion to database query languages has been extensively studied both from a theoretical [3] as well as from a practical point of view (e.g. SQL, SPARQL [4, 5]). In SQL it was included in the SQL-99 standard and was developed via common table expressions (CTE'S) that have a base SELECT statement and a recursive SELECT statement, that allows to express graph queries like DFS, BFS, topological sort, connected components, etc. Since queries must be linear and many interesting queries are not [6], optimizations have been proposed in [7, 8] (r-sql proposal) that allow only linear recursion and not the explicit negation that is a limitation when implementing recursion [3].

The graph algorithms BFS and DFS were implemented in [9] with the recursive SQL operator, where only trees were allowed as input since otherwise the query would loop infinitely. The topological order was studied in [10], making a BFS starting from the node whose outdegree is zero. In the case of the connected components it is shown in [11] how to obtain them adding recursion.

For SPARQL, Reutter et al [5] proposed a recursion operator based on SQL and make a comparison with property paths. In their study they formalize the syntax of the recursive operator and develop algorithms for evaluating it in practical scenarios. Also, a comparative study of the expressiveness of property paths and the recursive SQL operator was made in [6].

2 Adding a Recursive Operator to G-CORE

Among the main issues when adding recursion to query languages are the complexity of the evaluation, the expressiveness and the efforts to keep the declarative character of queries. From a theoretical point of view, recursive operators are based on the theory of least fixed points [3], that is, the increasing accumulation of results until eventually the evaluation does not add anything else and stops. Our proposal follows these ideas and is based both on the recursive SPARQL operator defined by Reutter et al [5] and the recursive SQL operator [4].

Proposed Syntax of linear recursive queries.

$$\texttt{WITH RECURSIVE } t \texttt{ AS } \{\ q_{base} \texttt{ UNION } q_{rec}\ \}\ q_{out}, \tag{2}$$

where t is a temporary PPG, q_{base} is a usual G-CORE query, q_{rec} is a positive G-CORE query that can use the temporary graph t and q_{out} a recursive G-CORE query itself.

Semantics of recursive queries. First, recall that the queries q_{base} and q_{rec} are usual G-CORE queries, therefore, are of the form (1). Second, recall that the evaluation of a usual query of the form (1) against a PPG G outputs a binding table $M(G)$, and from it the `CONSTRUCT` clause returns the PPG consisting of the union of $f(\ell)$ for each $\ell \in M(G)$. In what follows we will denote by M_b the binding table and by $C_b(\ell)$ the PPG returned by the `CONSTRUCT` clause of over the row ℓ of M_b of the base query q_{base}. Similarly we denote C_r and M_r respectively for the query q_{rec}.

Proposed Semantics of linear recursive queries. Let q be a recursive G-CORE query (like (2)) and G be a PPG. The answer $\text{ans}(q, G)$ of the query corresponds to the least fixed point of the sequence given by:

$$G_0 = G, \quad G_{-1} = \emptyset,$$
$$G_{i+1} = G_i \cup \{\text{ans}(q_{rec}, G + (G_i - G_{i-1}))\},$$

where: (1) the difference $(G_i - G_{i-1})$ is similar to the G-CORE difference operator defined in [2], except that the difference of the sets of nodes is redefined as $(N_i \backslash N_{i-1}) \cup \{a \in N_{i-1} : a \text{ is adjacent in } G_i \text{ to a node in } N_i \backslash N_{i-1}\}$; and (2) the semantics of $\text{ans}(q, G + H)$ means the match of q can use G or H or both (note that $\text{ans}(q, G \cup H)$ would not the same).

The answer $\text{ans}(q, G)$ of the recursive G-CORE query q over G is given by the following procedure:

Algorithm 1 Computing the answer of a recursive query in G-CORE

Data: PPG G, Queue of PPG'S $Q = \emptyset$, PPG $t = \emptyset$
for ***each*** *row l of the binding table $M_b(G)$* **do**
  `Insert`$(C_b(l), Q)$
while $Q \neq \emptyset$ **do**
  `Set` $r \leftarrow$ `Extract`(Q)
  $t \leftarrow t \cup r$
  for ***each*** *row l of the binding table $M_r(G + r)$* **do**
    `Insert`$(C_r(l), Q)$
return $ans(q_{out}, t + G)$

The idea is simple: first with the base query we construct the base case. Then with the recursive query we recursively construct a temporary graph which codes the solution that the output query will use.

Thus, first we begin with the queue Q empty and the temporary graph t empty. Then we evaluate the matching clause of q_{base} and for each row of the produced binding table $M_b(G)$, insert $C_b(l)$ into Q. Then we enter the while loop. As long as $Q \neq \emptyset$, we extract the top element of Q and update the temporary graph with it. Then for each row ℓ of the binding table $M_r(G+r)$ obtained from the query q_{rec} against $G+r$ (recall that this means that it could use the original graph G as well as the recently obtained graph r), we insert into the queue Q the new results obtained by the recursive construct $C_r(l)$. When we exhaust the queue Q the temporary graph t is finally ready to be used, and the query q_{out} is evaluated against $t+G$. If q_{out} is a standard G-CORE query the process outputs the result. If q_{out} is a recursive G-CORE query, we proceed as before again.

Notice that the fixed point exists whenever the query is monotone. On the form of query definition in (2) and tis semantics restricts queries to be linear [3–5, 11], that is, you can consult only the new elements added to `t`.

3 An Example: Topological Sort

In graph theory there are problems that have been widely studied and can be solved recursively in polynomial time. Due to space restriction, we will show as an example how the case of topological sort can be coded in recursive G-CORE.

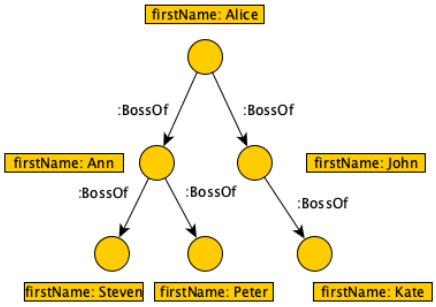

Fig. 1. A PPG G that represents hierarchy of a company

The classic algorithm to obtain the topological sort given an acyclic directed graph is Kahn algorithm (1962). It is not difficult to see that topological sort cannot be expressed in G-CORE. The next query illustrates with the graph of Fig. 1 how to code topological sort in G-CORE:

```
WITH RECURSIVE t AS (
          CONSTRUCT (n) SET n.depth :=0,
          MATCH (n) ON G,
          WHERE NOT EXISTS (CONSTRUCT (y),
                            MATCH (n)-[:BossOf]->(y) ON G)
          UNION{
          CONSTRUCT (x) SET x.depth:=n.depth+1,
          MATCH (x) ON G,
                (n) ON t
          WHERE EXISTS (CONSTRUCT (x),
                        MATCH (x)-[:BossOf]->(n) ON G)}
SELECT z.id,
MATCH (z) ON t,
ORDER BY MAX(z.depth) DESC
```

The above query first looks for those nodes which have no outgoing edges (base case) and are assigned with depth equal to 0 as a property. Then the recursive case comes and the nodes which have out edges to the nodes of the base case are added with property depth equal to 1 and so on until all the nodes in G have been added. Finally, we order by depth and we take the maximum value of the property depth.

References

1. LBDC. http://ldbcouncil.org/.
2. Renzo Angles, Marcelo Arenas, Pablo Barceló, Peter Boncz, George Fletcher, Claudio Gutierrez, Tobias Lindaaker, Marcus Paradies, Stefan Plantikow, Juan Sequeda, Oskar Van, Alex Averbuch, Hassan Chaa, Irini Fundulaki, Alastair Green, Josep Lluis, Larriba Pey, Jan Michels, Raquel Pau, Arnau Prat, Tomer Sagi, and Yinglong Xia. G-CORE A Core for Future Graph query Languages Designed by the LDBC Graph query Language Task Force *. *In: SIGMOD*, 2017.
3. Richard Hull Serge Abiteboul and Victor Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
4. Jim Melton and Alan R Simon. *SQL: 1999-Understanding Relational Language Components*. Morgan Kaufmann, 2001.
5. Juan L. Reutter, Adrián Soto, and Domagoj Vrgoč. Recursion in SPARQL. pages 19–35, Berlin, Heidelberg, 2015. Springer-Verlag.
6. N Yakovets, P Godfrey, and J Gryz. Evaluation of SPARQL property paths via recursive SQL. *CEUR Workshop Proceedings*, 1087, 01 2013.
7. Gabriel Aranda, Susana Nieva, Fernando Sáenz-Pérez, and Jaime Sánchez-Hernández. Formalizing a broader recursion coverage in SQL. pages 93–108, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.
8. Carlos Ordonez. Optimization of linear recursive queries in SQL. *IEEE Transactions on Knowledge and Data Engineering*, 22, 2010.
9. Devin W. Homan. Transitive Closure in SQL. http://dwhoman.com/blog/sql-transitive-closure.html.
10. FusionBox. Graph Algorithms in a Database . https://www.fusionbox.com/blog/detail/graph-algorithms-in-a-database-recursive-ctes-and-topological-sort-with-postgres/620/.
11. Torsten Grust. Advanced SQL. https://db.inf.uni-tuebingen.de/staticfiles/teaching/ss17/advanced-sql/slides/advanced-sql-05.pdf.