More power to SPARQL: From paths to trees

Angelos Christos Anadiotis[1][0000−0002−9727−8513], Ioana Manolescu[2][0000−0002−0425−2462], and Madhulika Mohanty[2][0009−0004−9446−8663]

[1] Oracle, Switzerland
angelos.anadiotis@oracle.com
[2] Inria and Institut Polytechnique de Paris, France
{ioana.manolescu, madhulika.mohanty}@inria.fr

Abstract. Exploring Knowledge Graphs (KGs, in short) to discover facts and links is tedious even for experts with knowledge of SPARQL due to their unfamiliarity with the structure and labels of entities, classes and relations. Some KG applications require finding the connections between groups of nodes, even if users ignore the shape of these connections. However, SPARQL only allows checking if paths exist, not returning them. A recent property graph query language, GPML, allows also returning connecting paths, but not connections between three or more nodes.
We propose to demonstrate RelSearch, a system supporting *extended* SPARQL queries, featuring standard Basic Graph Patterns (BGPs) as well as novel Connecting Tree Patterns (CTPs); each CTP requests *the connections* (paths, or trees) between nodes bound to variables. RelSearch evaluates such extended queries using novel algorithms [2] which, unlike prior keyword search methods, return connections regardless of the edge directions and are independent of how we measure the quality (score) of each connection. We will demonstrate RelSearch's expressivity and efficiency using a variety of RDF graphs, user-selected score functions, and search exploration orders.

Keywords: Graph Queries · Keyword Search · Exploratory Search.

1 Introduction

Knowledge Graphs (KGs) like Yago, DBPedia and Freebase form the backbone of many applications ranging from search engines, business intelligence to question answering. The RDF data model is the most common way to represent the KGs. They comprise subject-predicate-object (SPO) triples where subjects and predicates are resources, while objects are either resources, e.g., ElvisPresley, or literal values like strings, numbers, and dates. Thus, a KG has subjects and objects as its nodes, connected by relations as directed edges. KGs can be queried using SPARQL, with Basic Graph Patterns (BGPs) at its core. As RDF graphs typically lack a prescriptive schema, their structure may be complex. This makes it difficult for users to query KGs when they ignore the structure of the relationships between node groups that interest them. For example, consider the KG in Figure 1. A journalist may be interested in finding the connections between three groups of nodes: (*i*) US entrepreneurs, (*ii*) French entrepreneurs and (*iii*) French politicians, regardless of the structure and directionality of these

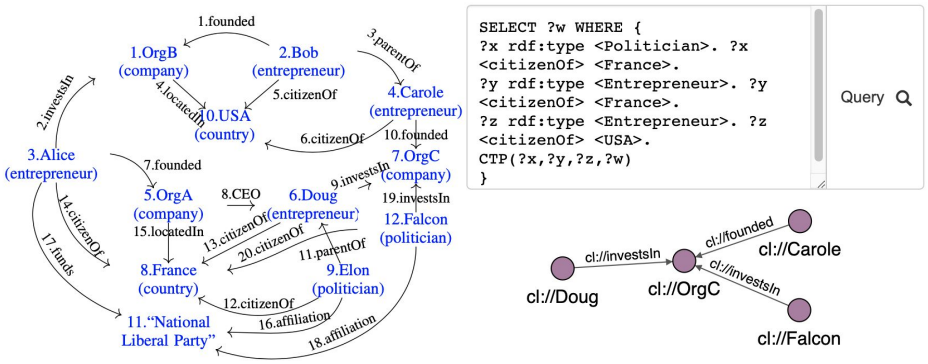

Fig. 1. Sample KG (left), and RelSearch demo screenshot (right).

connections. Current graph query languages, e.g., SPARQL, do not support expressing such a query, which is however useful when combining some criteria users have in mind ("US entrepreneurs", etc.) with graph structure discovery.

We propose to demonstrate the RelSearch system, enabling users to explore a KG, even when they cannot specify the exact labels or structure of these connections. RelSearch (*i*) extends the SPARQL syntax to support Connecting Tree Patterns (CTPs) alongside BGPs, (*ii*) efficiently executes the extended queries, and (*iii*) allows users to customize results by freely choosing the scoring function to use for ranking CTP results (thus, the extended query results), and several filters on their results. Scoring is important, because some paths are much more interesting than others, e.g., the journalist asking the above query is more interested to find that "Elon" is a parent of "Doug", than to find both are French (which was specified in the query). Being orthogonal wrt the score function is important, since it has been shown [4] that different applications and information needs are best served by different functions. These may privilege: node closeness, semantic coherence (or, on the contrary, diversity) on the edge labels in the connection, ranks of nodes along the connections, etc.

2 Framework: extending SPARQL for connection search

SPARQL queries may contain BGPs and/or Property Path Queries (PPQs). A BGP is a set of triple patterns, each of whose components may be a variable or a constant (URI or literal); BGP triples are connected by shared variables. A PPQ is a regular path query over the RDF graph; it allows checking (only) the presence of paths whose labels (property) match a regular expression, between query variables. Note that a PPQ does not allow the *any* regular expression, and can only combine user-specified properties. We extend SPARQL as follows:

CT Pattern A connecting tree pattern (CTP, in short) is a tuple of the form: $g = (g_1, g_2, \ldots, g_m, \underline{v_{m+1}})$ where each g_i, $1 \leq i \leq m$ is an URI or a variable and $\underline{v_{m+1}}$ is a variable. All variables $g_1, \ldots, g_m, \underline{v_{m+1}}$ are pairwise distinct.

CTPs are used to find connections among nodes: when replacing each g_i with a graph node, v_{m+1} is bound to a *subtree* of G, having the g_is as leaves. Formally:

Set-based CTP result Let $g = (g_1, \ldots, g_m, \underline{v_{m+1}})$ be a CTP pattern and $S_1, \ldots, S_m$ be sets of G nodes, called **seed sets**. The *result of g based on*

$S_1, \ldots, S_m$, denoted $g(S_1, \ldots, S_m)$, is the set of all $(s_1, \ldots, s_m, t)$ tuples such that $s_1 \in S_1$, ..., $s_m \in S_m$ and t is a *minimal* subtree of G containing the nodes $s_1, \ldots, s_m$. Here, minimal means: (i) removing any edge from t disconnects it and/or removes some s_i from t, and (ii) t contains only one node from each S_i.

Extended Query An extended query (EQ) consists of SPARQL BGPs, PPQs and/or CTPs. Its semantics is as follows. (i) When a g_i variable from a CTP appears also in some BGPs and/or PPQs, the respective seed set S_i is formed of all the nodes that (as per SPARQL semantics) match, simultaneously, the respective BGPs and PPQs. If a CTP variable g_j does not appear in any other place in the query, S_j consists of all the nodes in the KG. (ii) The EQ result is obtained by joining the BGP/PPQ results (seen as a table, binding variable to nodes) with the set-based results of all the CTPs in the EQ.

For example, Figure 1 shows the EQ seeking the connections between US entrepreneurs, French entrepreneurs and French politicians (at the top right). On the KG shown at left, the RelSearch screenshot in Fig. 1 also shows a sample connecting tree, matching the variable `?w`.

Our CTPs consider the graph *undirected*. This ensures that we do not miss any connections just because edge directions are not aligned. In our example, the CTP result connecting "Alice", "Falcon" and "Carole" via "National Liberal Party" and "OrgC" can only be found by considering edges in both directions.

CTP filters A CTP can have a very large number of results. Consider a KG of $2N$ triples over $N+1$ nodes (labeled `1`, `2`, and so on). Each node i is connected to $i+1$ by a triple whose property is `a`, and to the node $i-1$ through a property `b`. If v_1 is bound to `1` and v_2 to `N+1`, the CTP $(v_1, v_2, \underline{v_3})$, asking for all the connections between the end nodes, has 2^N solutions, or $2^{|E|/2}$, which grows exponentially in $|E| = 2N$, the number of KG triples. Observe that if we allowed only unidirectional paths, there would be only $N+1$ results, rooted at each node. Thus, matching CTPs regardless of the edge direction may drastically increase the number of CTP results; in some cases, computing all the CTP results may be **unfeasible**. To control the amount of effort spent evaluating CTPs, we also provide a set of orthogonal filters which allow to restrict set-based CTP results. Specifically, adding `UNI` for a CTP indicates that only *unidirectional* trees are sought, that is: a tree t, as in Def. 2, must have a *root* node, from which a *directed path* goes to each seed node in t. Specifying a set of labels $\{l_1, l_2, \ldots, l_k\}$ for a CTP indicates that the edges in any result of that CTP must have labels from the given set. Indicating a `MAX` n for a CTP indicates that only trees of at most n edges are sought. Further, a **score function** σ can be used to assign a real number $\sigma(t)$ (the higher, the better) to each connecting tree, measuring their interestingness to users. Using `TOP` k, one can restrict the CTP result to those having the k-highest σ scores. Finally, a practical way to limit the evaluation of a CTP is to specify a `timeout` T, that is, a maximum allowed evaluation time.

3 System and Demonstration Scenario

RelSearch relies on the ConnectionLens [3] system for storing the KG in a PostgreSQL table `graph(gID, s, p, o)`, heavily indexed. RelSearch extends a

SPARQL parser to incorporate the CTP atoms. The query execution engine is implemented in Java 11; the CTP evaluation algorithms [2] are integrated within ConnectionLens; BGPs are evaluated within Postgres.

We demonstrate RelSearch over two real-world datasets, a 6M edges subset of YAGO3, and a 18M edges subset of DBPedia. Users write their own queries and inspect the results, including connections shown as trees in the GUI. They can also select multiple filters for the CTPs – changing the number of results shown, the score function to be used to rank the trees, any specifications of permitted labels, direction of edges in the results and also limit the size of the results. `https://team.inria.fr/cedar/projects/relsearch/` outlines our demo.

4 Related Work

Many works address KG exploration; a recent categorization can be found in [7]. Such works have focused on: graph summarization, query by example, query suggestion and refinement, etc. Keyword search systems over KGs [8,9] return trees connecting nodes matching user-specified keywords. However, users cannot specify more conditions on nodes to be connected, e.g., "is of type Person, has age < 20, and their name matches *Jane*". Symmetrically, query languages do not currently support searching for connecting trees. SPARQL allows checking for (but not returning) paths connecting nodes; property graph languages such as GPML (not implemented) [5] and Neo4j's Cypher return paths, however, the latter does not scale [2]. RPQProv [6] uses recursive SQL to return path labels; JEDI [1] returns unidirectional paths (only). Going beyond paths, RelSearch combines SPARQL's expressive power with the ability to return trees connecting an arbitrary number of node sets, traversing edges in any direction, independent of a scoring function.

Acknowledgments This work has been funded by the AI Chair SourcesSay (ANR-20-CHIA-0015-01) project.

References

1. Aebeloe, C., Setty, V., Montoya, G., Hose, K.: Top-k diversification for path queries in knowledge graphs. In: ISWC (2018)
2. Anadiotis, A., Manolescu, I., Mohanty, M.: Integrating connection search in graph queries. In: ICDE (2023)
3. Chanial, C., Dziri, R., Galhardas, H., et al.: ConnectionLens: Finding connections across heterogeneous data sources (demonstration). PVLDB **11**(12) (2018)
4. Coffman, J., Weaver, A.C.: An empirical performance evaluation of relational keyword search techniques. IEEE TKDE **26**(1) (2014)
5. Deutsch, A., Francis, N., Green, A., Hare, K., Li, B., Libkin, L., et al.: Graph pattern matching in GQL and SQL/PGQ. In: SIGMOD (2022)
6. Dey, S.C., Cuevas-Vicenttín, V., Köhler, S., et al.: On implementing provenance-aware regular path queries with relational query engines. In: EDBT (2013)
7. Lissandrini, M., Mottin, D., Hose, K., Pedersen, T.B.: Knowledge graph exploration systems: are we lost? In: CIDR (2022)

8. Wang, H., Aggarwal, C.C.: A survey of algorithms for keyword search on graph data. In: Managing and Mining Graph Data, vol. 40. Springer (2010)
9. Yang, J., Yao, W., Zhang, W.: Keyword search on large graphs: A survey. Data Sci. Eng. **6**(2), 142–162 (2021)