gTop: An efficient SPARQL Query Engine

Yuqi Zhou[1], Lei Zou[1], and Gang Cao[2]

[1] Peking University, Beijing, China
{zhouyuqi,zoulei}@pku.edu.cn
[2] Beijing Academy of Artificial Intelligence, Beijing, China
caogang@baai.ac.cn

Abstract. In this demonstration, we present **gTop**, a top-k query engine based on gStore which supports SPARQL queries over RDF databases. gTop can answer top-k queries with high efficiency and scalability. We use the DP-B algorithm for top-k queries and the DP-any algorithm for any-k queries. We break cyclic queries into pieces of tree queries and use DP-any to solve queries generated before assembling the results to retrieve the origin answers. Experiments show the efficiency of gTop. We provide a demonstration website to show the usage of gTop, where users can query YAGO2 with top-k SPARQL queries.

Keywords: SPARQL · Top-k · Query Engine.

1 Introduction

Top-k queries can be expressed as queries with the 'LIMIT' and 'ORDER BY' clauses in SPARQL language. The 'ORDER BY' clause specifies the ranking criterion and the 'LIMIT' clause tells the number of top-ranked results users want. Fig. 1 shows an example of the SPARQL top-k query, which queries the top-5 cities having the largest GDP and is the birthplace of a musician. Top-k queries are a powerful tool to mine data information and are widely used by analysts.

```
SELECT * WHERE{
   ?artist rdf:type dbo:MusicalArtist .
   ?artist dbo:birthPlace ?place .
   ?place rdf:type dbo:City .
   ?place dbo:areaTotal ?area.
 }ORDER BY DESC(?area)  LIMIT 5
```

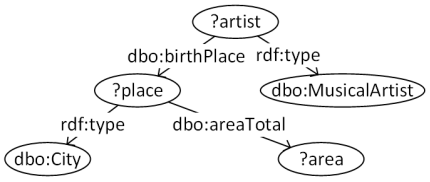

Fig. 1. An Example SPARQL Query

In this paper, we propose gTop, a new query engine specially designed for SPARQL top-k queries. gTop is built upon the gStore database [5], which is an open-source graph database using subgraph matching to answer SPARQL queries.

2 System Structure

The overview structure of gTop is depicted in Fig. 2. gTop uses different strategies to answer acyclic and cyclic top-k queries. Acyclic queries are directly forward into the top-k tree query solver, which uses the DP-B algorithm [2]. For cyclic queries, gTop first breaks them into tree queries through the query splitter and each sub-query will be evaluated by the any-k tree query solver. A controller takes charge of the enumeration process by dynamically deciding which sub-query to enumerate, and how many results should be outputted from the sub-query. The sub-results are assembled and then transferred to the users.

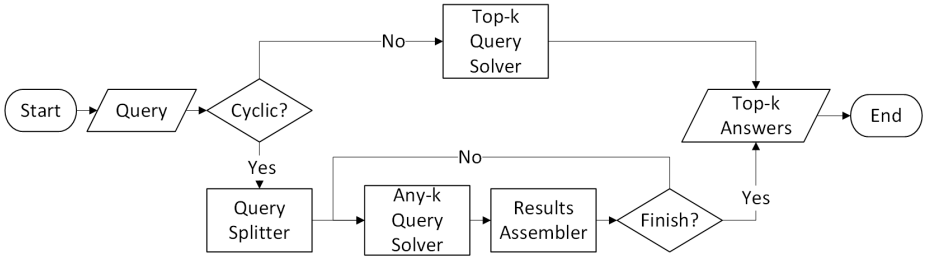

Fig. 2. The Working Process of gTop

Top-k Tree Query Solver The basic structure of a SPARQL query can be expressed as a graph, as shown in Fig. 1. If there is no cycle in the query graph, we refer to it as an acyclic query, otherwise, we call it a cyclic query. For acyclic top-k queries, the DP-B algorithm is the first linear algorithm proposed[2], which is used and adapted in gTop. The top-k solver completes this task in two steps. The solver first filters each variable's candidates from top to bottom and then enumerates the top-ranked result recursively from bottom to top.

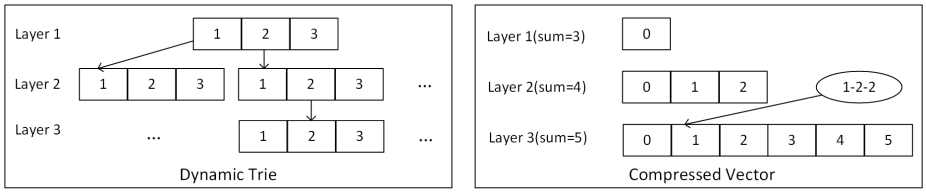

Fig. 3. An Example to illustrate Dynamic Trie and Compressed Vector

DP-B uses a dynamic trie to solve the **FQ-iterator Problem**. The problem focuses on enumerating top-ranked results from m ranked sub-lists, where each

combination of elements from the m sub-lists makes a valid result and the score is the sum of each element. For example, we want the min-3 results assembled from 3 ranked lists. We use a sequence of integers to identify a combination. Sequence $2-2-1$ refers to the result made of the two 2nd elements from the first two lists, and the 1st element of the last list. We observe that $1-2-1$ and $2-1-1$ are always no greater than $2-2-1$ and we call $1-2-1$ and $2-1-1$ parents of $2-2-1$.

In general, for sequences $x_1-x_2-...-x_m$ and $y_1-y_2-...-y_m$, if $\exists 1 \leq i \leq m, x_i = y_i - 1$, and $\forall 1 \leq j \leq m, j \neq i, x_j = y_j$, we call $x_1-x_2-...-x_m$ is the parent of $y_1-y_2-...-y_m$. Easy to see, a sequence is the potential to be the next result only when all its parents (if exists) have been enumerated. DP-B uses a Dynamic Trie to record whether all parents of a sequence have been enumerated. gTop adapts the DP-B algorithm for better performance in SPARQL queries, for example, it contains auxiliary indexes to support predicate variables of triples.

Any-k Tree Query Solver Solving the cyclic top-k query is an NP-hard problem. Some algorithms break the queries into tree sub-queries and assemble the sub-results back[1, 4], where the top-k tree query algorithms may not work well. For example, after getting the top-10 results, top-k algorithms may perform a whole search to the top-20 results, making no use of the previous effort. Our any-k tree solver treats k as dynamic, assuming no limit for the number of results users want and maintaining a scalable memory structure when running.

As mentioned before, the DP-B algorithm uses a dynamic trie that uses multi-layers of pointer arrays. The array size is fixed to k, limiting its ability to answer any-k queries. For an example shown in Fig. 3, to query sequence $2-2-1$, the dynamic trie first visits the 2nd pointer in the first layer which points to address $addr1$, and then the 2nd pointer in $addr1$, which maps to $addr2$, the information of the sequence in the first slot in $addr2$.

Our any-k tree queries solver uses another dynamic structure we proposed, Compressed Vector, to avoid the limitation. A compressed vector groups the sequence by the sum of the numbers in sequence. For example, in terms of 3 sub-lists, $1-2-2$ and $1-1-3$ are classified into one group Layer 3, which is composed of sequences summing to 5. A compressed vector maps each sequence into one non-negative number, that is, the lexicographical order of the sequence in the group.

Compressed Vector gets rid of the requirement to know k, avoids allocating too many memory pieces, and reduces the time to access RAM. The translation from sequences to integers (and the opposite direction also) is of low cost due to the efficient translation algorithm we implement. Such a mechanism allows the solver to continuously generate top-ranked results and is scalable to the data size and k.

Query Splitter and Results Assembler gTop treats SPARQL queries as subgraph isomorphism problems by treating the subject and object of a triple as nodes in a graph as in Fig. 1. In SPARQL, a user can define scoring functions

in literal values, which can only be objects of triples. According to this feature, we design a heuristic strategy to split queries. Once we find a cycle in the graph, we choose the edge with the worst selectivity to cut off. We keep cutting off edges until the query becomes a tree query. Then we run the DP-any algorithm to solve tree queries, once a result is enumerated, the results assembler checks whether the cut edges can be matched in the newly enumerated result, if not, ignore this result. The process stops if the any-k algorithm exhausts the results or we have already produced k valid results.

The correctness lies in the feature of SPARQL queries. The score of a result only comes from nodes, so ignoring some edges will not change the relative order of the origin top-k answers, but will bring more intermediate results. The query splitter chooses the edge with the worst selectivity to cut off to avoid overflooding intermediate results. The results assembler assures each output result is the correct match of the cyclic query.

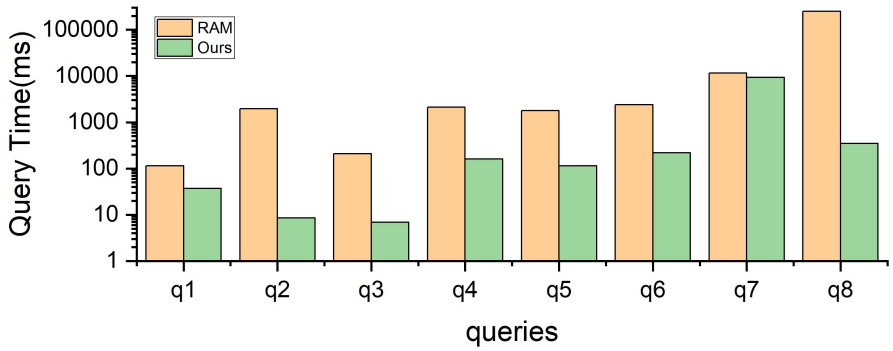

Fig. 4. The Answering Process of gTop

3 Experiments and Demonstration

We run our experiment on a server with an Intel(R) Xeon(R) CPU E5-2640 v3 2.60GHzCPU and 125GB memory running CentOS Linux. We evaluate all the algorithms by a real-world knowledge graph YAGO2[3]. YAGO2 is a high-quality knowledge graph extracted from Wikipedia, which is widely used in many real-world applications and is widely accepted as a criterion for performance in graph algorithms.

We test the algorithms with 8 queries and the results are shown in Fig. 4. In the figure, 'RAM' refers to the naive algorithm, which ranks and gets the top-k results after finding all the results by subgraph matching. The experiment shows that our strategy has a significant improvement over the naive implementation.

The queries have 2 to 10 query triples each and the results are ranked by the linear combination of up to 4 numeric literal variables.

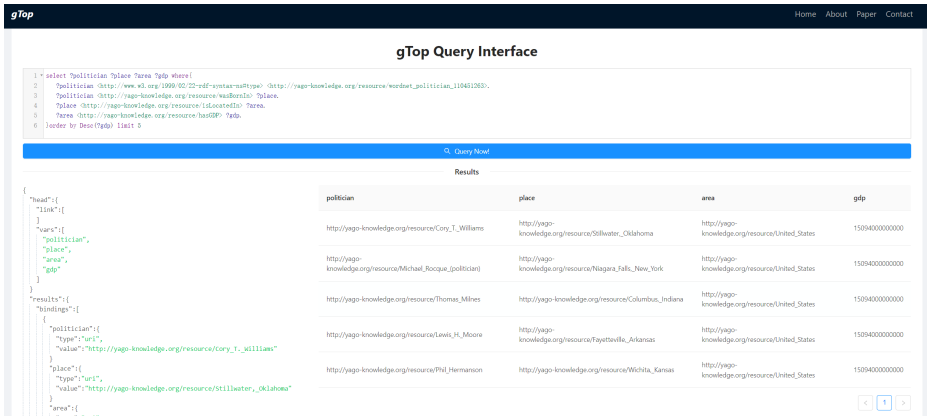

Fig. 5. The Answering Process of gTop

In the demonstration, we present a website for users to use gTop. We prebuild the YAGO2 databases and gTop processes query over YAGO2. Users can type in SPARQL top-k queries and get answers by pressing the 'Query Now' button on the website. The website is connected to an HTTP endpoint of gTop and the website will display the results returned by gTop, as shown in Fig. 5.

Acknowledgements This work was partially supported by National Key R&D Program of China (2020AAA0105200).

References

1. Cheng, J., Zeng, X., Yu, J.X.: Top-k graph pattern matching over large graphs. In: Jensen, C.S., Jermaine, C.M., Zhou, X. (eds.) 29th IEEE International Conference on Data Engineering, ICDE 2013, Brisbane, Australia, April 8-12, 2013. pp. 1033–1044. IEEE Computer Society (2013). https://doi.org/10.1109/ICDE.2013.6544895, https://doi.org/10.1109/ICDE.2013.6544895
2. Gou, G., Chirkova, R.: Efficient algorithms for exact ranked twig-pattern matching over graphs. In: Proceedings of the 2008 ACM SIGMOD international conference on Management of data. pp. 581–594 (2008)
3. Hoffart, J., Suchanek, F.M., Berberich, K., Weikum, G.: YAGO2: A spatially and temporally enhanced knowledge base from wikipedia. Artif. Intell. **194**, 28–61 (2013). https://doi.org/10.1016/j.artint.2012.06.001, https://doi.org/10.1016/j.artint.2012.06.001
4. Zeng, X., Cheng, J., Yu, J.X., Feng, S.: Top-k graph pattern matching: A twig query approach. In: Gao, H., Lim, L., Wang, W., Li, C., Chen, L. (eds.) Web-Age Information Management - 13th International Conference, WAIM 2012, Harbin, China, August 18-20, 2012. Proceedings. Lecture Notes in Computer Science, vol. 7418, pp. 284–295. Springer (2012). https://doi.org/10.1007/978-3-642-32281-5_28, https://doi.org/10.1007/978-3-642-32281-5_28

5. Zou, L., Mo, J., Chen, L., Özsu, M.T., Zhao, D.: gstore: Answering SPARQL queries via subgraph matching. Proc. VLDB Endow. **4**(8), 482–493 (2011). https://doi.org/10.14778/2002974.2002976, http://www.vldb.org/pvldb/vol4/p482-zou.pdf