Reinforcement Learning-Based SPARQL Join Ordering Optimizer

Ruben Eschauzier[1(✉)], Ruben Taelman[1], Meike Morren[2], and Ruben Verborgh[1]

[1] IDLab, Department of Electronics and Information Systems, Ghent University - imec, Ghent, Belgium
ruben.eschauzier@ugent.be

[2] Marketing, School of Business and Economics, Vrije Universiteit Amsterdam, Amsterdam, The Netherlands

Abstract. In recent years, relational databases successfully leverage reinforcement learning to optimize query plans. For graph databases and RDF quad stores, such research has been limited, so there is a need to understand the impact of reinforcement learning techniques. We explore a reinforcement learning-based join plan optimizer that we design specifically for optimizing join plans during SPARQL query planning. This paper presents key aspects of this method and highlights open research problems. We argue that while we can reuse aspects of relational database optimization, SPARQL query optimization presents unique challenges not encountered in relational databases. Nevertheless, initial benchmarks show promising results that warrant further exploration.

Keywords: SPARQL · Join Order Optimization · Reinforcement learning · Machine Learning

1 Introduction

Optimizing the order in which database management systems execute joins is a well-studied topic in database literature because it heavily influences the performance characteristics of queries [11]. SPARQL endpoints over consistently evolving datasets, like Wikidata, can benefit from an algorithm that optimizes queries based on previous experiences. Different signals exist to inform an appropriate choice of join order, such as cardinalities. One such signal is *previous experiences*. We use previous experiences as a predictor to produce better join plans for future queries.

In recent literature, reinforcement learning(RL)-based optimizers that use *greedy search procedures*, guided by a learned *value function*, achieve impressive results in relational databases. Neo [5] shows that learned optimizers can match and surpass state-of-the-art commercial optimizers.

In SPARQL, machine learning is primarily used to predict query performance. These approaches [2,3,12] use supervised machine learning with a static

C. Pesquita et al. (Eds.): ESWC 2023, LNCS 13998, pp. 43–47, 2023.
https://doi.org/10.1007/978-3-031-43458-7_8

dataset of query executions. Learned query optimizers use reinforcement learning to dynamically generate training data, complicating the use of existing query performance prediction methods. The April [10] optimizer uses reinforcement learning for query optimization, with a one-hot encoded [6] feature vector denoting the presence of RDF terms in joins. However, the paper does not report any performance characteristics.

We fill this gap in the literature by exploring a fully-fledged RL-based query optimizer for SPARQL join order optimization on SPARQL endpoints. Endpoints query over the same dataset, likely making the previous experience signal stronger for join order optimization. We model our approach after the RTOS [11] optimizer for relational queries, which uses Tree-LSTM neural networks [9] to predict the expected latency of a join plan.

2 Method

To iteratively build up an optimized join plan, the RL-based optimizer greedily adds the join that minimizes the estimated query execution time at each iteration. For the first iteration, we have the result sets of all triple patterns, and in each subsequent iteration, we join two result sets. We estimate the execution time of the query using a neural network, which we train to minimize the mean squared error between predicted and actual query execution time. We feed a numerical representation of the current join plan as an input to the neural network.

Join Plan Representation. Like in the optimizer RTOS [11], we represent join plans as a tree that we build from the bottom up. Each leaf node represents the result set of a triple pattern, and internal nodes represent join result sets. We represent result sets using their cardinality, the presence and location of variables, named nodes and literals, and a vector representation of the predicate. We learn the predicate representation vectors by applying the RDF2Vec [7] algorithm to the RDF graph.

RDF2Vec generates learned vector representations of RDF terms that encode information on what RDF terms co-occur often. RDF2Vec first generates random walks on the input RDF graph, then for each random walk, it randomly removes an RDF term and trains a neural network to predict the missing term. The weights obtained during the model training are the feature vectors of the RDF terms in the graph. RDF2Vec does not learn variable representations because an RDF graph has no variables. The subject and object of triple patterns are often variables, so we do not encode named nodes in these positions. We obtain the representations for intermediate joins by applying an N-ary Tree-LSTM [9] neural network on the result sets representations involved in the join. These representations are optimized during training, thus allowing the model to determine

which features of the result sets involved in the join are important. Finally, at the (partial) join plan root node, we apply the Child-Sum Tree-LSTM network [9] to all unjoined result sets to obtain the numerical join plan representation.

Data Efficiency and SPARQL-Specific Adjustments. Data generation using query execution is slow; we account for this by applying two data efficiency techniques. First, we include a *time-out* set according to existing optimizers. We effectively truncate our optimization variable while ensuring the optimal query plan will not reach the time-out. Second, we use *experience replay* [4] to store previous (expensive) query executions and reuse them for training.

Relational RL-based optimization approaches use *one-hot encoding* [6] of database attributes to create feature vectors. However, large graphs like Wikidata can contain over 100 million unique entries. One-hot encoding that many attributes would create unwieldy vectors and degrade performance. To improve scalability, we do not use one-hot encoding in our approach, instead, we use feature encoding techniques to capture state information in fixed-size vectors.

Open Challenges. We have not found a way to encode connections between triple patterns. To encode all information in the query graph, these encodings should reflect the possible connections between triple patterns, like object-object, subject-subject, object-subject, and subject-object. Which makes using a simple adjacency matrix infeasible. Furthermore, our approach can only optimize basic graph patterns; in future work, this approach should be extended to more complex SPARQL query operations. Finally, we do not learn feature representations for variables; to enrich our triple pattern representation, we should encode variables based on the other RDF terms in the triple pattern.

3 Initial Experiments

We implement our optimizer in the TypeScript-based Comunica query engine [8] and compare it to the default cardinality-based optimizer. We use the WatDiv benchmark [1] to test our method, and show the performance characteristics of a preliminary version of the model. Table 1 shows that the model can find better plans for 7 templates, which we believe we can improve using the data efficiency and SPARQL-specific adjustments mentioned in Sect. 2. The search time of our method is significantly longer than the standard comunica optimizer. However, we run these benchmarks on a dataset with only about 100,000 triples. For large RDF graphs, like Wikidata, we expect that the execution of the join plan dominates the total query execution time.

Table 1. Comparison of the query optimization and plan execution time, in seconds, of a previous version of our optimizer and the standard Comunica [8] optimizer, with the faster plan execution in bold.

Query Template	C1	C2	C3	F1	F2	F3	F4	F5	L1
Planning (RL)	0.5028	1.000	0.277	0.214	0.347	0.160	0.800	0.278	0.027
Execution (RL)	3.116	2.577	0.583	0.100	0.090	0.062	1.906	**0.059**	**0.006**
Planning (Comunica)	0.007	0.008	0.017	0.002	0.003	0.005	0.005	0.005	0.002
Execution (Comunica)	**0.076**	**0.001**	**0.490**	**0.001**	**0.005**	**0.008**	**0.012**	0.194	0.032
Query Template	L2	L5	S1	S2	S3	S4	S5	S6	S7
Planning (RL)	0.025	0.024	0.689	0.060	0.059	0.066	0.059	0.021	0.028
Execution (RL)	**0.001**	**0.002**	2.242	0.011	**0.005**	**0.000**	**0.002**	0.008	0.002
Planning (Comunica)	0.001	0.001	0.006	0.002	0.002	0.002	0.002	0.002	0.002
Execution (Comunica)	0.006	0.007	**0.139**	**0.009**	0.008	0.005	0.009	**0.001**	**0.000**

4 Conclusion

In this paper, we explore a novel RL-based join plan optimizer for SPARQL endpoint query execution. Initial experiments show that the model can generate better join plans than existing cardinality-based optimizers for 7 query templates of the WatDiv benchmark. We plan to improve the model by enhancing data efficiency during training. We propose to use query *time-outs* based on existing query optimizers to reduce the time spent executing bad query plans. Additionally, we propose to use *experience replay* to reuse query execution information during training. For future work, we should include information on how triple pattern result sets connect to other result sets in the query, encode the RDF terms present in the subject and object locations of a triple pattern, and extend our approach to more complex SPARQL operations.

Acknowledgments. This work is supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral fellow of the Research Foundation - Flanders (FWO) (1274521N).

References

1. Aluç, G., Hartig, O., Özsu, M.T., Daudjee, K.: Diversified stress testing of RDF data management systems. In: Mika, P., et al. (eds.) ISWC 2014. LNCS, vol. 8796, pp. 197–212. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11964-9_13
2. Casals, D., Buil-Aranda, C., Valle, C.: SPARQL query execution time prediction using deep learning
3. Hasan, R., Gandon, F.: A machine learning approach to SPARQL query performance prediction. In: 2014 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT) (2014)
4. Lin, L.J.: Self-improving reactive agents based on reinforcement learning, planning and teaching. Mach. Learn. **8**, 293–321 (1992)

5. Marcus, R., et al.: Neo: a learned query optimizer. arXiv preprint arXiv:1904.03711 (2019)
6. Müller, A.C., Guido, S.: Introduction to machine learning with Python: a guide for data scientists (2016)
7. Ristoski, P., Paulheim, H.: RDF2Vec: RDF graph embeddings for data mining. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9981, pp. 498–514. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46523-4_30
8. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a modular SPARQL query engine for the web. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11137, pp. 239–255. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00668-6_15
9. Tai, K.S., Socher, R., Manning, C.D.: Improved semantic representations from tree-structured long short-term memory networks. arXiv preprint arXiv:1503.00075 (2015)
10. Wang, H., et al.: April: an automatic graph data management system based on reinforcement learning. In: Proceedings of the 29th ACM International Conference on Information & Knowledge Management (2020)
11. Yu, X., Li, G., Chai, C., Tang, N.: Reinforcement learning with tree-LSTM for join order selection. In: 2020 IEEE 36th International Conference on Data Engineering (ICDE) (2020)
12. Zhang, W.E., Sheng, Q.Z., Qin, Y., Taylor, K., Yao, L.: Learning-based SPARQL query performance modeling and prediction. World Wide Web **21**, 1015–1035 (2018)