Exploring Monte Carlo Tree Search for Join Order Selection

Ji Zhang[§ζ], Kun Wan[§], Sebastian Schelter[ζ], Ke Zhou[§]
[§]Huazhong University of Science and Technology, [ζ]New York University
{jizhang,k.wan,k.zhou}@hust.edu.cn,sebastian.schelter@nyu.edu

1 INTRODUCTION

Query optimization remains a difficult problem, and existing database management systems (DBMSs) often miss good execution plans [1]. Identifying an efficient join order is key to achieving good performance in database systems. A primary challenge in join order selection is enumerating a set of candidate orderings and identifying the most effective ordering [2]. Searching in larger candidate spaces increases the potential of finding well working plans, but also increases the cost of query optimization. In this paper, we explore the benefits of *Monte Carlo Tree Search* (MCTS) for the join order selection problem.

MCTS [3] is a search method usually used in games to predict the set of moves that should be taken to reach a final winning solution with high likelihood. It simulates the game many times and tries to predict the most promising move based on the simulation results. For example, in the game of Go [4], MCTS simulates the game several times to select the actions and orders which have the highest probability to win. Inspired by this method, we explore the application of MCTS to join order selection. Our main idea is to simulate many possible join orders, and to apply MCTS to select the order to execute with the highest estimated performance. We design a neural network to predict the query execution time of a given plan, to which we refer as *Order Value Network* (OVN). MCTS leverages this network to score candidate query plans.

2 PROPOSED APPROACH

Figure 1 gives an overview of our proposed approach, which includes two encoding methods (SQL-encoding and Plan-encoding), two neural networks (Order Value Network and Decision Network), and the proposed optimizer which applies MCTS.

SQL-encoding encodes the table and attribute information contained in the SQL query. Similar to previous work [5], the representation of each query consists of two components: the first component encodes the join graph of the query in an adjacency matrix as shown in Figure 1. A one in the matrix corresponds to the join predicate connecting two tables. The second component is a simple "one-hot encoding" of the attributes involved in any SQL predicate.

Plan-encoding represents a partial execution plan. The difference to SQL-encoding is that we represent the join order instead of just the join graph in the encoding matrix (in contrast to [2]). We show two encoding examples in Figure 1.

Optimization with MCTS consists of two parts: the first part is the Order Value Network (OVN) which is trained based on historical query plans and their corresponding runtime using a standard convolutional neural network. The second part is the MCTS. We construct the Monte Carlo Tree for join order selection by simulating different join orders, where we apply the OVN to score their performance, and use the "Upper Confidence bounds applied to

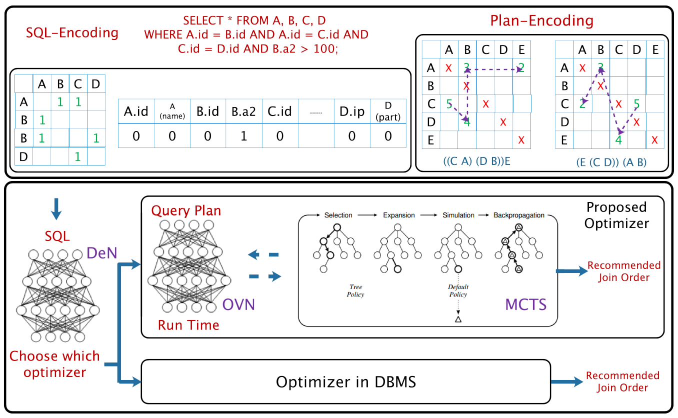

Figure 1: Overview of our proposed approach which includes two encoding methods (SQL-encoding and Plan-encoding), two trained neural networks (Order Value Network (OVN) and Decision Network (DeN)), and an optimizer which applies Monte Carlo Tree Search (MCTS).

Trees (UCT)" algorithm to balance exploration and exploitation. Note that the path from each child node to the root node in the tree represents a complete join order.

3 PRELIMINARY RESULTS

We investigate the performance of our method via the Join Order Benchmark (JOB), a set of queries used in previous assessments of query optimizers [1]. We configure Postgres to execute the generated query plans. In initial attempts, we applied our MCTS-based optimizer to replace the optimizer in PostgreSQL. Although our approach achieved an overall better performance than the optimizer in Postgres, we found that some queries optimized by our method performed much worse than if they would have been optimized with the optimizer in Postgres. This attributes to the uncertainty of the UCT algorithm in our method. In order to alleviate this situation, we train another neural network to which refers to as the Decision Network (DeN, shown in Figure 1) to choose between our optimizer and the Postgres optimizer for a given query. DeN is learned from a labeled dataset of historical executions times of the optimizer in PostgreSQL and our MCTS-based optimizer.

We present preliminary experiments that indicate that our method can generate join orders with lower runtimes than the ones generated by the PostgreSQL optimizer and the state-of-the-art method *Neo* [6]. The results are shown in Figure 2, where the y-axis denotes the execution time (including the MCTS search time) and the x-axis of the right figure shows the test queries ordered by their execution time. Our approach is able to achieve not only better overall performance but performs well for individual queries, and

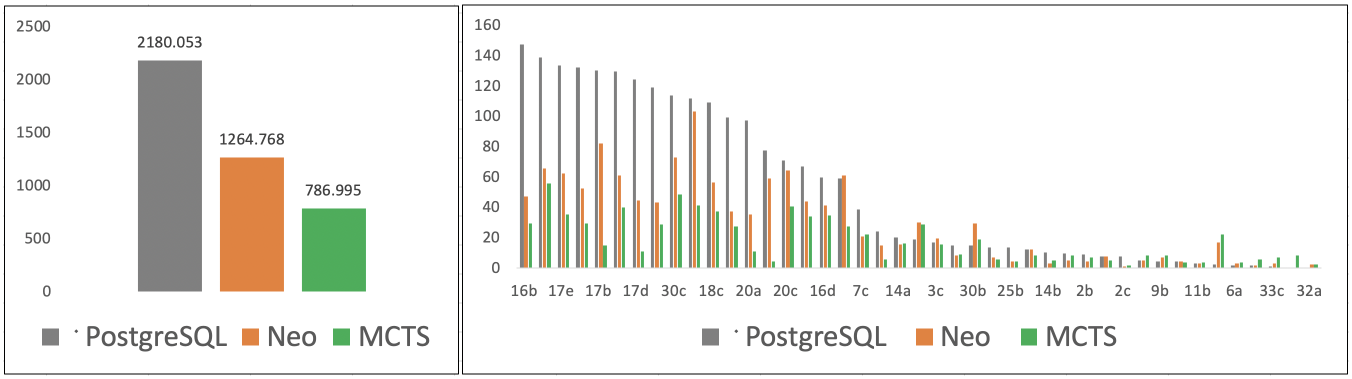

Figure 2: Performance comparison between the optimizer of PostgreSQL, Neo and MCTS. The MCTS method achieved the best overall performance.

gains a remarkable improvement for slow queries (queries with large execution time).

Our method still performs worse than the optimizer in PostgreSQL on a set of fast queries, which we need to investigate to further improve the overall performance. Additionally, we aim to further reduce the MCTS search time.

REFERENCES

[1] Viktor Leis, Andrey Gubichev, Atanas Mirchev, Peter Boncz, Alfons Kemper, and Thomas Neumann. How good are query optimizers, really? *Proc. VLDB Endow.*, 9(3):204–215, 2015.
[2] Ryan Marcus and Olga Papaemmanouil. Deep reinforcement learning for join order enumeration. aiDM'18. Association for Computing Machinery, 2018.
[3] Rémi Coulom. Efficient selectivity and backup operators in monte-carlo tree search. In H. Jaap van den Herik, Paolo Ciancarini, and H. H. L. M. (Jeroen) Donkers, editors, *Computers and Games*, pages 72–83, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
[4] David Silver, Aja Huang, and et. al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.
[5] Jennifer Ortiz, Magdalena Balazinska, Johannes Gehrke, and S. Sathiya Keerthi. Learning state representations for query optimization with deep reinforcement learning. In *Proceedings of the Second Workshop on Data Management for End-To-End Machine Learning*, DEEM'18. Association for Computing Machinery, 2018.
[6] Ryan Marcus, Parimarjan Negi, Hongzi Mao, Chi Zhang, Mohammad Alizadeh, Tim Kraska, Olga Papaemmanouil, and Nesime Tatbul. Neo: A learned query optimizer. *Proc. VLDB Endow.*, 12(11):1705–1718, July 2019.