Nanotechnology Perceptions
ISSN 1660-6795
www.nano-ntp.com

Analysis of Query Optimization Using Deep Reinforcement Learning Using Particle Swarm Optimization Algorithms

Karthikeyan M P[1], Dr. Krishnaveni K[2]

[1]Research Scholar, Department of Computer Science, Sri S.Ramasamy Naidu Memorial College, (Affiliated to Madurai Kamaraj University, Madurai), Sattur,Tamilnadu, India, karthi.karthis@gmail.com
[2]Associate Professor & Head, Department of Computer Science, Sri S. Ramasamy Naidu Memorial College, (Affiliated to Madurai Kamaraj University, Madurai), Sattur, Tamilnadu, India, kkrishnaveni@srnmcollege.ac.in

Query optimization is a well-studied problem in the database industry, with numerous solutions proposed over the last several decades. The success of deep reinforcement learning (DRL) has generated new opportunities in query optimization. One of the most difficult tasks in query optimization and query plan generation is determining the order in which join operations between tables are done (i.e. relations). Even if the final results of a query remain identical regardless of join order, the order in which the tables of a query are joined can have a significant impact on query execution time. Deep reinforcement learning, in particular a data-driven method to reasoning about enumeration heuristics, provides a novel algorithmic viewpoint on join enumeration. We must now control what training data the model views and how that data is featured, rather than the standard tunable parameters of a query optimizer. The algorithm makes few assertions about the cost model's structure or the search space's topology. We demonstrate that Q learning optimizes plans well across many different cost models for a small set of training queries. On the TPC-H database, the Query Optimization Algorithms Q-Learning and PSO (Particle Swarm Optimization) are assessed. During the evaluation, the optimizer failed to complete one query within the maximum time permitted, whereas the deep reinforcement learning-based models (Q-Learning) and heuristics model (PSO) managed. Of course, the standard join ordering problem is NP-hard, and practical algorithms use heuristics to make the search for a good plan efficient. A novel method for Query Optimization using Particle Swarm Optimization (QOPSO) and Deep Q Learning (DQL) for parameter tuning of Join Operation Cost and Processing Cost is suggested in this paper.

Keywords: Query Optimization, Deep Reinforcement Learning, Particle Swarm Optimization, Machine Learning, Join Query, Neural Networks.

1. Introduction

To achieve high performance in database systems, a query optimizer that can considerably

reduce query execution time is required. On the one hand, writing a decent optimizer today takes a lot of man-hours, so only a small group of experts have complete control over their craftsmanship. Worse, search optimization methods require repair work, — particularly as the system's completion as well as combustion engine capacity grow and develop. As a result, all completely accessible public information request optimization methods aspire to match its production. Yes, overseas business machines, Intel, and Microsoft's advertising take various forms. So, while query processing appears to be heuristic-based, there have been intensive efforts in recent generations to improve search optimization techniques by requiring them to learn. To find the best plan for sequential DBMSs, query planner optimization can be done and developed using a variety of planner creation and choosing algorithms. Because jobs take longer to complete than relational searches, query optimization algorithms and more complex algorithms are needed. To optimize a query rapidly, make sure it runs quickly and employs the best optimization strategies available. To reduce the plan search area, most databases evaluate only left-depth plans. The execution of queries, on the other hand, is more essential for efficiency, so there is no pipeline between the original data and the operator [1][2].

Query optimization is in charge of selecting the best alternative strategy for executing a query while taking into account a variety of parameters. Among the variables are Computational power, I/O operations, selection, projection, join ordering, division, and communication cost. The query engine in DBMS handles this task. The query processor is in charge of locating tables that are needed in a distributed query, which may be distributed and/or duplicated, incurring communication costs. Query response time is increased since data access from multiple sites [3].

The method of improving a database query's processing technique is known as query optimization. As a consequence, it is a critical stage in query processing. The word "query handling" refers to the operations that occur when information is obtained from a database [4]. Because these queries are given to the DBMS in an elevated code, these activities include query translation into expressions that can be applied at the file system level, query optimization steps, transformations, and query evaluation [5]. Subscribe several times Query optimization is a difficult problem in any relational database system, not just SQL server [6] [7]. The query is converted into standard format, i.e. a query graph, after the parser first checks for syntax crimes When you type in (5)the search engine analyzes your query and creates a plan based on what it thinks looks best. The query prosecution machine is also urged to use this formal query plan; it is evaluated and the query result is returned.

The process of selecting the best query plan from a collection of alternatives by taking into account Query optimization is the process of determining which queries, based on the data and the user's specific requirements, can be run quickly on the database. Even when the best query plans are used, the results can be inaccurate or take longer than required to process. This is due to the fact that the underlying models used to generate the plans are sometimes inaccurate, and the plans themselves aren't always tailored to the requirements of the user.

We want a scheduler that can use data from past queries to improve future ones more effectively. We've come up with a new query optimization technique called Deep Q Learning (DQL) and Particle Swarm Optimization (QOPSO). These methods use deep reinforcement learning to give better query plans faster.

2. Problem Formation

The objective of the optimization technique is to start creating a query processing strategy that cuts down on overall query processing time. Conventional query optimizers use cost-based and logically sound optimization. When applied to action plans, The principles of heuristic processing are to do things in a certain order, to use the operations that are the most constrained, and to start with the operations that are the easiest. This almost always leads to a shorter processing time.The same optimizer analysts evaluate and analyse During cost-based optimization, the expenses of query execution are estimated using statistics and cardinality. After that, the strategy with the lowest projected cost is selected. By doing so, the query is run as few times as feasible to achieve the lowest cost.

Example 1: Consider the subsequent TPC-H [8] [9] Database example: utilizing the retail database for the We look at data from TPC-H [10] to figure out what might be a good plan for our work. This includes figuring out what products we might sell, what kinds of orders we might receive, and where the parts for those products might come from, access methods, unary selections, and binary combine operators. As a running example, we'll use the database below, which contains three relations that each important current about a student and a join relation:

In this dataset focus four tables are Customer (C_Custkey, C_Name, C_Address , C_Nationkey) , Order (O_Orderkey, O_Custkey,O_OrderDate, O_Totalprice) , Nation (N_Nationkey, N_Name, N_Regionkey, N_Comment) and Region (R_Regionkey,R_Name, R_Comment).

Consider the following join query:

```
SELECT * FROM Customer as C, Oder as O,
Nation as N, Region as R WHERE C.C_Custkey
= O.O_Custkey AND C. C_Nationkey
=N.N_Nationkey AND N.N_Regionkey =
R.R_Regionkey
```

To begin running this query, join the orders and client relations. The above produces the connect graph shown in Figure 1. It includes, in particular, freshly constructed orders and customer relationships. Performing a nested loop join on purchases, customers, and nations is one option from here. In order to determine the q-value of carrying out this action in the current state, the action and state must be encoded as a fixed size vector that the trained neural network model expects as input. In this case, the move and query would be encoded as:

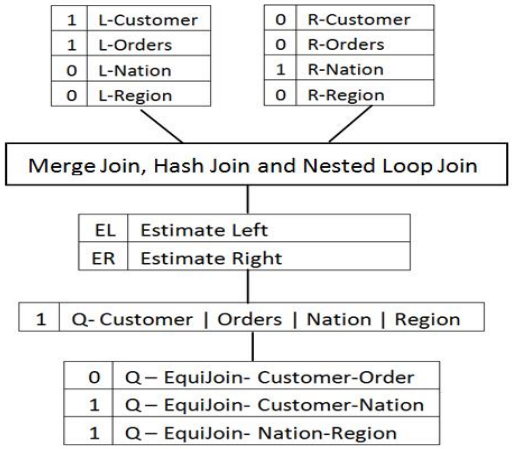

Figure 1: Join Graph

Before conducting the join action, the 1-hot encrypts the tables from the left and right input relations. Its first 4 connections are on the left, and the next four are on the right. Despite the fact that customers and purchases are connected on the left, the first two rows are each one and the third and fourth are both zero. The actual join operator, which may be a mergejoin, hashjoin, or nested loop join, is then 1-hot encoded for the join operation. Next, encode the left and right input links predicted by the PostgreSQL [11][12] optimizer. Encode the relationships that are displayed in the end query return.

Encode whether the equijoin predicate exists for each of the three groups of tables. Three of these items are non-zero, corresponding to the query's three equijoin predicates.

3. Deep Reinforcement Learning

Using neural networks to estimate Q-values or a policy given a state, Deep Reinforcement Learning (DRL) is a hybrid of Reinforcement Learning and Deep Learning that forms the basis for action selection. Many challenging problems and real-world issues have been successfully solved in this field of research. For multi-dimensional problems, a family of stochastic optimization techniques called reinforcement learning (RL) [13] is used (MDPs). An RL algorithm uses random sampling to create a model that predicts how decisions will affect the overall reward. This model can then be used to help make better decisions in the future. When deep neural networks are used for reinforcement learning, The final result is deep reinforcement learning. Deep neural networks are the agents that learn to link state-action combinations to rewards. The neural network is either encouraged or discouraged by the outcome of the action done on this input in the future, depending on the outcome. Numerous domains have made significant use of deep reinforcement learning [14]. It is especially utilised in fields where an agent may receive a reward or a reprimand for any behavior.

Our main finding is that join ordering and reinforcement learning have a close algorithmic relationship (RL). The issue formation that underlies RL and the sequential structure of join ordering are the same. This algorithm allows us to integrate RL models into traditional query

optimization systems, making it much simpler to use RL models [15]. This realisation enables us to rapidly integrate RL models into other optimization systems by leveraging System R's architecture. When compared to earlier suggestions for a "learning optimizer," we secondly use the problem's layered structure to drastically lower training costs.

To better understand how rl relates to join ordering, think about the conventional "bottom-up" dynamic programming approach. A plan is created by breaking it down into smaller parts, and then following an algorithm that will always result in the best possible solution. In order to create a succession of 1-step optimal decisions, enumerated subplans are stored in a lookup table. Then, because of its powerful expressiveness, rl uses deep rl techniques to recall the various states (DRL). Deep neural networks are also capable of predicting future states based on the past states. The environment, or database management system, is how the user, or planner, interacts with it. The environment currently generates a set of valid actions on which two tables can be joined and records the present state, which is the immediate join plan. Deep neural networks are used to assess the value of actions and states, and The optimizer selects an activity from the activity basis of a set on its likelihood of producing the intended result. The chosen action is then rewarded by the DBMS by switching to the following state with the selected combo. When a full join plan is created and all the tables are joined, this MDP process halts. An episode is the word for the entire process from the initial state to the forming limit, which is followed by another episode that repeats the process until the model coincide.

SELECT * FROM Customer as T_C, Oder as T_O, Nation as T_N, Region as T_R WHERE T_C.C_Custkey = T_O.O_Custkey AND T_C.C_Nationkey =T_N.N_Nationkey AND T_N.N_Regionkey = T_R.R_Regionkey

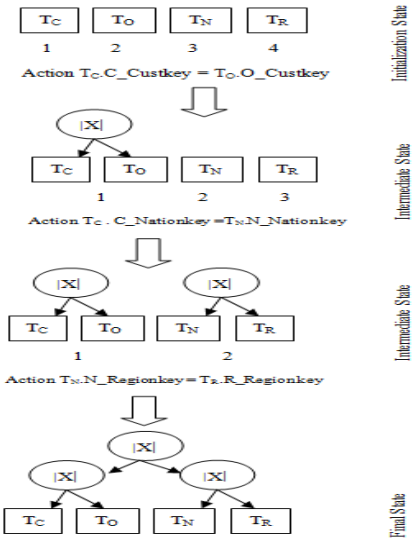

Figure 2: DRL-based join order selection procedure.

Figure 2 depicts a possible scenario for a query involving four relations: Customer as TC, Order as TO, Nation as TN, and Region as TR. s1 = TC, TO, TN, TR is the starting state. The action set A1 includes for each ordered pair of relations, each element, such as (1, 2) A1 for joining TC with TO and (2, 3) A1 for joining TN with TR. The agent selects the action (1, 3), which represents the decision to combine TC and TN. s2 = TC |X| TO, TN, TR is the next province. The agent then selects the action (2, 3) that represents the decision to combine TN and TR. The next state is s3 = { T_C |X| T_O, T_N |X| T_R}. The agent has only two options left at this point, A3 = {(1, 2), (2, 1)}. If the entity chooses the option (1, 2), the subsequent state s4 = { (T_C |X| T_O)|X| T_N |X| T_R} represents a terminal state.

A linear function of each condition is required to join data. This representation contains details about the join tree topology as well as the join/selection predicates. Then, to show that reinforcement learning strategies can work well even with little input data, we outline a straightforward vectorization approach for capturing this information. Trunk's Composition data about the tree topology, Since there are n total links in the database, We create a row vector of size n for each binary subtree x sj. If the ith relation is in x, then the value vi is equal to 1 h(i,x); if it is not in x, then the value vi is equal to zero. h(i, x) is the height of the relation ri in the subtree x. (the distance from the root). the second-to-last state's first entry in the tree vector, TC |X| TO, TN |X| TR, equates to (TC |X| TN) in the example in Figure 2.The subtree C has a height of 2 according to the number 12 in the third column of the first row. The second column of the first block is empty because relation B is not contained in the subtree, which results in a number of zero. We produce $n \times n$ binary symmetric matrix m for each show to hold important join predicates data. The valuemi, j is one when a join condition connects the ith and jth relations; otherwise, it is zero. All feasible equi-joins procedures are covered by this simple representation. Figure 3 shows a sample of this array. Because of the condition TC, the valuem2,1 = m1,2 = 1. TO.O Custkey = C Custkey. Because there is no join predicate linking B and C, the valuem 2,3 = m3,2 = 0. A k-dimensional vector, where k is the amount of characteristics in the database, makes up the selection predicate vector (the actual number of attributes shared by all relations). If the provided query has a selection predicate for the ith attribute, the ith number is one; otherwise, it is zero. This shows the traits that are used or are not used to sort tuples.

RL is a kind of learning where you try to figure out how to do something by figuring out what rewards (like points or money) you get when you do it. However, RL doesn't work with data that's already labelled, like a dataset of past actions. You need to have this data available before you can train your learning model. And once your model is trained, you can use it to predict future actions.

$$Q(S_t, a_t) \leftarrow Q(S_t, a_t) + \alpha(R_t + \gamma Q(S_{t+1}, a_{t+1}) - Q(S_t, a_t). \quad (1)$$

In Q-Learning [16][17], all potential state-action combinations (St, at) and their corresponding Q-values are kept in a table (referred to as a Q-table). In Equation (1), Q (St, at) is an evaluated value (also known as the Q-value) that tells us what action to take when we see the present state at St. This value is used to determine which action to take based on the current conditions. In our case, there are numerous containers on which a single inquiry can be performed. As a result, the Q-table could hold a large number of state-action pairs.Iterating over a large Q-table

adds time to query processing. The value-based reinforcement learning technique Q-Learning is well-known [10]. In Q-Learning, a table (referred to as a Q-table) is used to record all possible state-action pairs (St, at) as well as the evaluated Q-value linked with each pair. After consulting the Q-table, the agent finds the Q-value for each possible action, and then chooses the action with the highest value. This is important for reinforcement learning, as it helps keep the value of the action through proper evaluation. This number can be between 0 and 1, depending on how often the agent wants to learn from the experience.

Algorithm 1: Query Processing with with Deep Q Learning

Data: SQL query, Weight Profile wp, Reward Function R (), Learning rate, Discount rate
Result: The query result set of the input query
t=0;
Result = Q_O
Qt= 0;
QEP = Query Optimizer (query)
While QEP != Q_O
State St = convert QEP to a state vector;
Actiont=RunLearningModel (St, wp);
Result=Result [execute (Op, Actiont);
QEP=QEP-Op;
Update Rt=R (wp, Actiont.time, Actiont.money));
Obtain Q-value of next state Q_{t+1} from the neural network;
Update Q-value of current state Q_t = Bellman(Q_t;Q_{t+1};Rt; α,γ);
Update Weights in the neural network;
t=t+1;
end
return Result;

When the optimizer gets a query, it converts it into a QEP (logical plan). The QEP is then transformed into a vector representation. As a result, the current QEP accurately represents the current condition and can be fed into a deep net. To show this vector, we use a one-hot vector, a technique adapted from recent work. This vector is sent to the RL model, which is a neural network. The RL model will assess the Q-values for all feasible steps to execute the following query operator. Each of these operations has two parts: the best physical operator and the best receptacle for the next query operator to be performed. The DBMS will then select and perform the move with the highest Q-value. The completed function is then removed from the QEP, and the time and resources spent on its execution are used to calculate the reward for this activity. In the Bellman Equation, the reward is updated to account for the time and expense of executing the operator, and the updated reward also affects the expected Q-value (3).The neural network weights are properly updated using the back- propagation method. This process is repeated for each operator in the QEP and is completed when all of the operators in the QEP have been executed. The query's results are then sent to the recipient. The reward function is critical to the entire programme. According to the Bellman equation, if the reward for completing a prior action At-1 is high on state St-1, the Q-value will be high as well. This

means that, given the same state, the action with the best prior performance will be chosen more frequently.

4. QUERY OPTIMIZATION USING PARTICLE SWARM OPTIMIZATION (QOPOS)

The optimization of join queries is an NP-hard [18][19] issue. The number of query execution plans (QEP) corresponding to a query grows exponentially as the number of joins increases, resulting in a very large computational complexity of the multi-join query optimization issue. Solving problems with heuristic algorithms has recently become popular. Selfish Algorithm [20], Genetic Algorithm [21], Ant Colony optimization Algorithm [22], and others are examples. PSO represents the most effective efficient algorithm influenced by intelligent swarm movement. PSO is a tried-and-true technique for resolving social interaction problems. the quantity acting as agents that move around in the solution domain in search of the best solution. Each particle changes depending on personal experience as well as the experiences of other particles. Individually and globally, each particle is accelerated towards the finest particle for them [23] [24].

PSO does not require too many parameters to be adjusted and is gradient information, making it straightforward and easy to implement. Additionally, the early rapid convergence rate is significant. Although the traditional particle swarm algorithm has some impact on query optimization [25] [26], it has limited local search capability and is susceptible to local optimal flaws. The user can search with new inquiries to users of real information need, and reduce the results of the inquiry that has nothing to do with the user's desires, so the accuracy of the returned results is improved. The weight of the inquiries vector is continuously adjusted. Documents and queries are represented as vectors, and the initial population is built using the findings of the first query optimization deployment of the evolutionary algorithm and particle swarm mixed algorithm.

PSO for Query Optimization: Particle Swarm optimization for finding best v
Input: data set consists quires and their associated Join Query values, methodology for tuning parameters.
Output: optimized values of Join Query.

Methodology:
Step 1: The n queries are initialized particles with position and velocity taken randomly of tuning parameters along with the range of velocity between. The Initial positions of each particle with query are taken Personally Best.
Step 2: Let w=0.5, cognitive learning parameter c1=2.0, and social parameter c2=2.0.
Step 3: Repeat the following steps 4 to 9 until particles exhaust or number of repetitions.
Step 4: For each particle evaluate fitness function (Join Query) for each particle, position, and tuning parameters. The objective is to minimize the Join Query by taking values quantified in step 1.
Step 5: find the Pbest particle by calculating and comparing Join Query of the current values and previous values. If current one is better than the previous then update the values otherwise ignore it.
Step 6: Select the least Pbest values as Gbest.
Step 7: The weights are updated using the following equation

$$W_{new} = [(T_{mi} - T_{ci}) * (W_{iv} - W_{fv})]/T_{mi} + W_{fv}$$

Step 8: The C1,C2 factors are modified with
C1(t) = 2.5 – 2 * (Tci / Tmi)
C2 (t) = 0.5 + 2* (Tci / Tmi).
Step 9: each query values updated using the following equations
for j = 1 to m do
begin ( )

$$V_{ji}^{n+1} = w * V_{ji}^{n} + c_1 * rand()_1 * (Pbest - S_{ji}^{n}) + c_2 * rand()_2 * (Gbest - S_{ji}^{n})$$

$$S_{ji}^{n+1} = S_{ji}^{n} + V_{ji}^{n+1}$$

end;
Step 10: Join Query values Gbest are optimal solution.
Step 11: Stop

The fitness function structure and the starting population generation this paper's information retrieval paradigm is based on the vector space model. In a multidimensional space, In order to describe records and queries, a vector is used, with one vector for each dimension having the weight of representative keywords. Documents and the vector's query structure Documents and queries the Angle vector in the vector space model to show the distance between two vectors. The greater the cosine of two vectors, the more comparable they are to one another. By comparing the query with the similarity of each record size, the document is then sorted in decreasing order. An n-dimensional vector representing any text dj D has the following expressions: dj = (w1j, w2j,..., wnj), where the vector components wij are the first I feature words ki in text dj, and n is the overall number of key words in the system. Wij's value scope is in the continuous real number range [0, 1] due to the vector space model's "partial matching" strategy. The user's queries are also expressed as an n-dimensional vector q = (w1q, w2q,... wnq), where wiq denotes the first I keyword ki's weight value inquiry.

Initial Population: This initialization generation starts with both the initial query retrieval as well as employing In order to calculate how similar two papers are, angle cosine is used. This enables us to better reflect user requirements, leading to improved search results.

Fitness Function: The degree of similarity between documents and queries can be seen; the more similar, the better it will serve the requirements of users; the more top, the smaller the angle; and the more similar, the better it will convert for the most amount of objective function. The angle of the orientation in the aforementioned formula illustrates how comparable the vector representation and the query vector function cosine (di, q) are for values [0, 1].

Let us choose the fitness function to determine the best query strategy. The cost of creating a left deep tree is the fitness function for the query optimization issue. The selectivity and cardinality of the data are estimated to determine this expense. The number of triples that fit a specific pattern is known as its cardinality. The number of triples meeting both TC.C Custkey = TO.O Custkey is the measure of the selectivity of a join between two triples, TC.C and TC.O. Ri should represent cardinality, and fi,j should represent choice. Selectivity can be defined as in the following if pi,j is the join condition between Ri and Rj (3):

$$f(i,j) = Ri\ |X|_{ij}\ Rj \,/\, Ri\ |X|\ Rj \quad (3)$$

For a given join tree T, the resultant cardinality |T| can be recursively computed as in (4) and (5):

$$|T| + |Ri| \text{ if T is a leaf Ri} \quad (4)$$

$$|T| + |Ri| + (\Pi Ri \in T_{C.C\_Custkey},\ Ri \in T1, fi,j)\ |\ T_{C.C\_Custkey} \,||\, T_{O.O\_Custkey} \,|\, if\ T = T_{C.C\_Custkey} \infty T_{O.O\_Custkey} \quad (5)$$

, For a given join tree T, the cost function C_{out} is defined as

$$C_{out}(T) = 0 \text{ if T is a leaf Ri} \quad (6)$$

$$C_{out}(T) = |T| + C_{out}\,|\,T_{C.C\_Custkey}| + C_{out}\,|\,T_{O.O\_Custkey} \text{ if } T = T_{C.C\_Custkey} \infty T_{O.O\_Custkey} \quad (7)$$

The PSO has the advantage of requiring fewer parameters to adjust the requirements. The effectiveness and accuracy of the algorithm are significantly impacted by these variables in any situation. In our optimization issue, every particle represents a possible allocation

optimization solution. The optimal allocation can be expressed as an N-dimensional vector, Xi = Xi1, Xi2,..., Xin, due to the fact that there are N jobs in grid labour. the j-th job has been assigned to resource Xij for execution, as indicated by each part of Xij. As an illustration, four tasks have been given to three resources. Table 1 shows that Xi1 = 2, Xi2 = 1, Xi3 = 3, and Xi4 = 2 have been assigned by the i-th particle, Xi1 = 2. This means that job 1 has been assigned to resource 2 for processing.

5. Performance Evaluation

PostgreSQL 8.4 modified query optimizer and query engine were used in the experiments. The data is spread among these VPSs. The TPC-H database benchmark is used to create the searches and database tables. TPC-H Database is based on A wealth of data on clients, orders, line items, sections, part distributors, distributors, countries, and regions can be found in the retail database from TPC-H benchmark. TPC-H is still the most popular relational system benchmark, and most join searches involve three or more tables. [13]. To illustrate the join ordering challenge, the following four tables are used (customer, orders, nation and region). Our The basic row size is the scale factor of 1 that is used in TPCH experiments (equals 1GB raw data). We run 50,000 queries, with query templates selected at random from the benchmark's 22 query templates.

Query latency:Figure 3 depicts the effectiveness of the query plan that the PostgreSQL optimizer chose versus the latency of the performed query plans produced by QOPSO and DQL. Throughout this instance, each test query is executed 15 times with a chilled cache. The graph shows the minimum, maximum, and median delay increases. The plans generated by DQL join ordering always perform better than or are on par with the plans generated by PostgreSQL. Consequently, DQL can produce plans that are easier to implement (not just a lower cost according to the cost model). Once again, QOPSO's performance is considerably worse than DQL's, demonstrating that DQL is not simply speculating on a random join sequence.

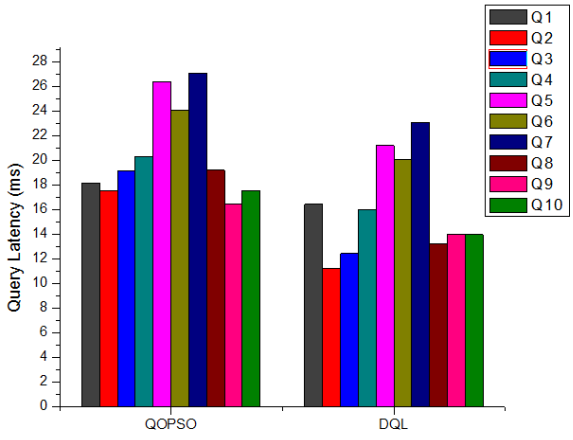

Figure 3: Query Latency

Optimization Latency: The time it takes to create a query strategy, or optimization latency, for QOPSO and DQL is compared in Figure 4. Each point represents one of the TPC-H queries, and spots above The queries represented by the black curve were those for which DQL had a quicker optimization time than QOPSO. The amount of relations joined in the query is colour coded on the points. Even though QOPSO avoids Because of the overhead of calling the neural network, Deep Plan is slightly slower than DQL for small join queries. The exponential search cost outweighs the neural network overhead as more relatives join, resulting in a relatively faster DQL optimization latency. This occurs when 11 or more links are connected.

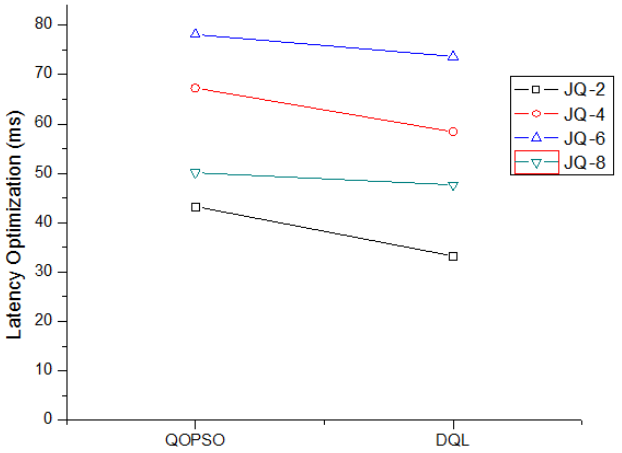

Figure 4: Optimization Latency

Average Query Execution Cost: Given the index configuration selected, the cost is calculated by adding the execution times of each individual query. In this instance, we employ the previously described execution time, which corresponds to the engine's cost. We will be allowed to use correct action for our training and evaluations in this work, as well as the impact of the recommended indexes on actual running time. With minor differences in execution time, the suggested 92% of the benchmark queries, the model found execution plans that outperformed QOPSO, and they failed in 8% of them. In any case, both provide the same implementation methods. As shown in Figure 6, the mean processing time of the QOPSO model (represented by the X mark in graphs) is greater than that of the DQL model. The average execution time of queries in the benchmark using the suggested DQL model is 79.11 ms, while the average implementation time for QOPSO is 89.14 ms. Furthermore, in a subset of searches, the DQL model outperforms QOPSO.

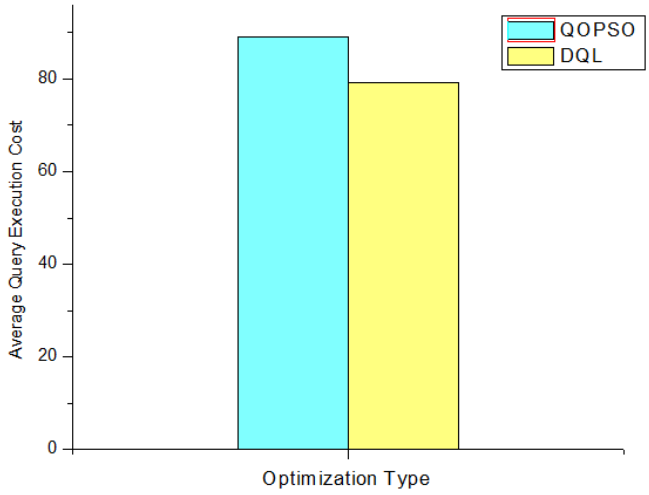

Figure 4: Average Query Execution Cost

Running Time and Convergence Speed:The difference between QOPSO and DQL is whether curriculum learning methods are employed. In comparison to QOPSO and DQL, it reduces training time and gets to the optimal value quicker. Undoubtedly, searches with fewer tables are trained first because they require fewer iterations to obtain representations and, as a result, take less time to train. The overall training time of QOPSO is 478.32 ms, which represents a 95% relative improvement over DQL. This demonstrates that QOPSO can provide the mode with quicker and more stable training.

6. Conclusion

The learning model underpins query optimization, and evolutionary algorithms produce better results. Despite the fact that the best way of learning data is available, a methodical technique that can yield much better results is required.. For Join Query optimization, we proposed a novel model structure combination of Query Optimization using Particle Swarm Optimization (QOPSO) and Deep Q Learning (DQL) in this paper. The criteria for determining top query plans are running time and convergence speed, average query execution cost, optimization latency, and query latency.It can be challenging to determine the best join sequence because As there are more tables, there are increasingly more join pairs that the optimizer must take into account. This means the optimizer can't calculate costs for any of the tables. every combination that might be used. As a To achieve a balance between optimization time and plan quality, the majority of optimizers use heuristic rules to prune the search area. Furthermore, the optimizer estimates the cost of a plan using pre-calculated statistics, which may cause the optimizer to choose an inefficient plan. Traditional models require more planning time as the number of join conditions grows.The performance evaluation results indicate that DQL outperforms QOPSO in selecting show how learning-based models can speed up query planning by identifying the optimal join orders for numerous queries. The deep reinforcement learning model has an advantage over the PSO model in that it is better suited

to scenarios with a large number of states and actions because it employs a neural network to approximate the query execution time. The experiments also show that the QOPSO model can generalise to previously unseen training-phase query conditions, which is another significant advantage of deep reinforcement learning models.

References

1. J. Wang, R. Elmasri and M. Eltabakh, "Query Optimization Techniques for NoSQL Databases," 2021 IEEE International Conference on Big Data (Big Data), 2021, pp. 4965-4970.
2. J. Zhou et al., "Automatic Query Optimization: A Survey," IEEE Transactions on Knowledge and Data Engineering, vol. 33, no. 2, pp. 432-450, Feb. 2021.
3. R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, 2018.
4. A. K. Tripathi, A. Shukla and R. Kumar, "Performance Evaluation of DBMS Using Different Benchmarks," 2021 International Conference on Advances in Computing, Communication and Control (ICAC3), 2021, pp. 1-6.
5. L. Zeng, J. Wang and X. Liu, "DBMS Performance Optimization Based on Machine Learning," 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2021, pp. 2045-2048.
6. B. V. Joshi and M. A. Rajendran, "Performance Analysis of SQL Server Using Index Optimization Techniques," 2021 International Conference on Advances in Computing and Data Sciences (ICACDS), 2021, pp. 1-6.
7. V. K. Singh, A. K. Singh and D. K. Upadhyay, "SQL Server Performance Tuning: A Comprehensive Approach," 2021 IEEE International Conference on Computing, Communication and Automation (ICCCA), 2021, pp. 1-5.
8. A. Sharma, R. Ghosh and R. Ranjan, "Benchmarking TPC-H on Cloud Platforms: A Comparative Study," 2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom), 2022, pp. 1-8.
9. Y. Zhao, L. Zheng and J. Wu, "TPC-H Benchmark Testing for Distributed Database Systems," 2021 IEEE 37th International Conference on Data Engineering (ICDE), 2021, pp. 1-8.
10. TPC. 2013. TPC-H Benchmark. http://www.tpc.org/tpch/
11. M. K. Srivastava and A. K. Sharma, "Performance Tuning and Optimization in PostgreSQL: A Case Study," 2021 International Conference on Computational Intelligence and Communication Technology (CICT), 2021, pp. 1-6.
12. S. Gupta and A. Kumar, "PostgreSQL Optimization Techniques for Big Data Workloads," 2021 IEEE International Conference on Big Data (Big Data), 2021, pp. 1-8.
13. Z. Xu, W. Sun and Y. Liu, "Reinforcement Learning Based Intelligent Control Systems: A Survey," IEEE Transactions on Neural Networks and Learning Systems, vol. 33, no. 4, pp. 1386-1406, Apr. 2022.
14. J. He, Y. Li and T. Zhang, "Reinforcement Learning for Autonomous Driving: A Review," IEEE Transactions on Intelligent Vehicles, vol. 7, no. 2, pp. 175-195, June 2022.
15. R. Marcus and O. Papaemmanouil. Deep reinforcement learning for join order enumeration. In Proceedings of the First International Workshop on Exploiting Artificial Intelligence Techniques for Data Management, page 3. ACM, 2018.
16. H. Zou, L. He, Z. Zhang and S. Liu, "Q-Learning Based Task Offloading Optimization in Vehicular Edge Computing," 2022 IEEE International Conference on Communications Workshops (ICC Workshops), 2022, pp. 1-6.

17. Y. Liu, Z. Yang, Z. Zhang and Y. Xu, "Q-Learning Based Congestion Control for Wireless Multimedia Sensor Networks," IEEE Transactions on Wireless Communications, vol. 21, no. 5, pp. 3285-3296, May 2022.
18. C. Li, X. Yao, Z. Zhang and J. Chen, "Approximability of NP-Hard Optimization Problems," IEEE Transactions on Computers, vol. 70, no. 12, pp. 2195-2206, Dec. 2021.
19. P. K. R. Madduri, R. K. S. Gorthi and S. N. Sivanandam, "Efficient Heuristic Solutions for NP-Hard Problems Using Genetic Algorithms," 2021 IEEE Congress on Evolutionary Computation (CEC), 2021, pp. 1-8.
20. R. T. Mahajan, S. N. Shah and M. A. Joshi, "Selfish Algorithm Based Approach for Distributed System Optimization," 2021 IEEE 10th International Conference on Communication Systems and Network Technologies (CSNT), 2021, pp. 1-5.
21. T. M. Vo and P. R. Prakash, "A New Hybrid Genetic Algorithm for Solving Multi-objective Optimization Problems," 2022 IEEE Congress on Evolutionary Computation (CEC), 2022, pp. 1-6.
22. X. Li, J. He, and M. L. Wong, "Ant Colony Optimization for Dynamic Optimization: A Survey," IEEE Transactions on Evolutionary Computation, vol. 26, no. 4, pp. 710-723, Aug. 2022.
23. M. Ghosh, S. Dutta and S. N. Saha, "Enhanced Particle Swarm Optimization Algorithm for Multi-objective Problems," 2022 IEEE Congress on Evolutionary Computation (CEC), 2022, pp. 1-8.
24. L. Zhang et al., "Adaptive Particle Swarm Optimization Algorithm for Global Optimization," IEEE Transactions on Cybernetics, vol. 52, no. 1, pp. 450-462, Jan. 2022.
25. M. Li, H. Zhang, Z. Zheng and Y. Zhao, "Deep Q-Learning Based Resource Allocation for Network Slicing in 5G Wireless Networks," 2022 IEEE Global Communications Conference (GLOBECOM), 2022, pp. 1-6.
26. K. Zhang, H. Ma and Z. Wang, "Adaptive Deep Q-Learning for Resource Management in Dynamic Network Slicing," IEEE Transactions on Mobile Computing, vol. 21, no. 1, pp. 158-170, Jan. 2022.