[1]Ms. Nisha Jain,

[2]Dr. Preeti Tiwari

Implementing Hybrid Tournament Selection Strategy for Join Query Optimization in HiveQL

Journal of Electrical Systems

Abstract: Genetic Algorithm (GA) use different selection methods to determine which individuals are chosen for reproduction. These selection strategies directly influence the algorithm's convergence speed, diversity, and overall efficiency. This research paper focuses on enhancing the performance of GA in the Hadoop-Hive environment by optimizing query join orders, a crucial aspect of query execution. Join order optimization is an NP-hard problem, meaning that traditional deterministic methods often fail to find optimal solutions efficiently. To address this challenge, propose an algorithm which is a hybrid of selection methods that combines Tournament and Ranking based Selection to improve the balance between exploration (searching a broad solution space) and exploitation (refining the best solutions). Tournament Selection ensures diversity by allowing a mix of high and low-fitness solutions, while Ranking Selection prevents premature convergence by ensuring fairness in selection probability distribution. The join order problem is modelled using the Traveling Salesman Problem (TSP) approach, where different join sequences are treated as paths that need optimization. By mapping the problem to TSP, the TRGAHH (Tournament-Ranking Genetic Algorithm Hadoop Hive) algorithm can effectively explore and evaluate multiple join sequences, identifying the most efficient query execution plan. To validate the effectiveness of our proposed hybrid selection method, we conducted extensive experiments using the TPC-DS dataset, a widely accepted benchmark for Hive query performance evaluation in large-scale data environments. Our experiments were carried out in a Hadoop-Hive framework, with multiple queries involving complex joins executed under different GA selection strategies.

Keywords: Genetic Algorithm (GA), Hadoop, Hive, Join Order Optimization, Tournament Selection, Rank-Based Selection, TPC-DS Dataset

Introduction

Big Data has transformed data management, and one of the key solutions to this challenge is Hadoop. Developed by the Apache Software Foundation [1], Hadoop is an open-source framework that uses a distributed computing model [2], enabling organizations to handle large-scale data processing tasks by leveraging distributed computing on cost-effective hardware clusters, capable of managing both structured and unstructured data sets. This allows for cost-effective and scalable data processing, which is a significant improvement over traditional relational database systems, which often struggle with scalability and performance issues. Hadoop [3] operates using a master-slave architecture and it consists of two key parts: HDFS (storage) [11] and MapReduce (processing). HDFS safely stores data across many machines by breaking it into blocks and copying them, ensuring data is always available even if hardware fails. The MapReduce programming model [10], on the other hand, divides computation tasks into smaller parallel tasks, which improves resource utilization and enables efficient data processing. Hadoop's advantages include scalability, fault tolerance, flexibility, cost-effectiveness, and parallel processing [4]. It can scale horizontally by adding more nodes, handle a range of data formats, lower infrastructure costs, and support parallel task execution. However, challenges persist, particularly with query optimization in Hadoop-based tools like Apache Hive [5]. Hive, a data warehousing solution built on top of Hadoop, enables users to query large datasets in HDFS using a language similar to SQL (HiveQL) [6], eliminating the need for complex MapReduce programming. Hive serves as a bridge between relational database management systems (RDBMS) and Hadoop, simplifying big data analysis for users familiar with SQL. It supports various file formats, including TextFile, SequenceFile, ORC, and RCFile [7], offering flexibility for diverse data storage needs. Hive translates HiveQL queries into MapReduce jobs that execute in a directed acyclic graph (DAG) format, optimizing query performance [8]. The processed results are then stored back in

[1] Assistant Professor, S.S. Jain Subodh P.G. Mahila Mahavidyalaya, Jaipur, Rajasthan, jain.nisha28@gmail.com

[2]Associate Professor, International School of Informatics &Managment, Jaipur, Rajasthan, Preeti.tiwari@icfia.org

HDFS, benefiting from its scalability and fault tolerance. Hive's ability to integrate SQL-like syntax with big data processing makes it crucial in modern database systems.

Query optimization is essential for improving the efficiency of data processing in big data systems [9]. Among the various optimization challenges, join order optimization is particularly complex. Determining the optimal sequence for joining multiple tables in a query is an NP-hard problem [12], meaning that finding the best solution requires exhaustive computations, which is infeasible for large datasets. Traditional query optimizers, such as rule-based [10] or cost-based approaches [13], often struggle to handle these complex queries efficiently in high-dimensional data environments. To address these challenges, Genetic Algorithm (GA) have gained attention as an effective heuristic method for join order optimization. GA simulates natural evolution, using selection, crossover, and mutation techniques to iteratively improve execution plans, enhancing query performance. The benefits of GA include scalability, adaptability, and the ability to process tasks in parallel, making them a promising solution for optimizing query execution in big data systems.

1.2 Hive Query Processing Using Genetic Algorithm (GA) for Join Order Optimization

Genetic Algorithm (GA), introduced by J.H. Holland in 1992 [15], is a probabilistic optimization method inspired by natural selection, widely used for solving complex computational challenges. It iteratively refines potential solutions using evolutionary mechanisms such as selection, crossover, and mutation [17]. In the context of Hive, GA plays a crucial role in join order optimization. The process starts with a randomly generated set of query execution plans (QEPs), which evolve over generations to minimize execution costs. The selection operator favors fitter QEPs, improving the overall quality of the population [16]. Meanwhile, the crossover operator combines genetic material from two parent QEPs [19], and mutation introduces random changes to maintain diversity and avoid premature convergence. This evolutionary process iteratively refines the join order, leading to improved query performance in distributed systems like Hadoop [21]. GA is particularly effective for solving NP-hard optimization problems [21] like join order optimization in databases, especially within the context of distributed environment-Hive. It uses a population-based approach, maintaining multiple candidate solutions to explore a wide solution space. This helps to avoid local optima, a common challenge in optimization problems. Unlike traditional optimization techniques, such as dynamic programming [14] or greedy algorithms [13], GA can simultaneously evaluate multiple solutions, allowing for the exploration of diverse areas within the solution space while refining the best options. GA's ability to handle complex, multi-objective optimization problems makes it ideal for Hive join order optimization, which is often modelled as a Traveling Salesman Problem (TSP) [22]. GA's strength lies in its flexibility and ability to navigate discontinuous and multi-modal search spaces. By balancing conflicting goals, such as reducing query execution time and optimizing resource utilization, GA enhances performance and efficiency in distributed environments like Hadoop, making it an indispensable tool for modern query optimization.

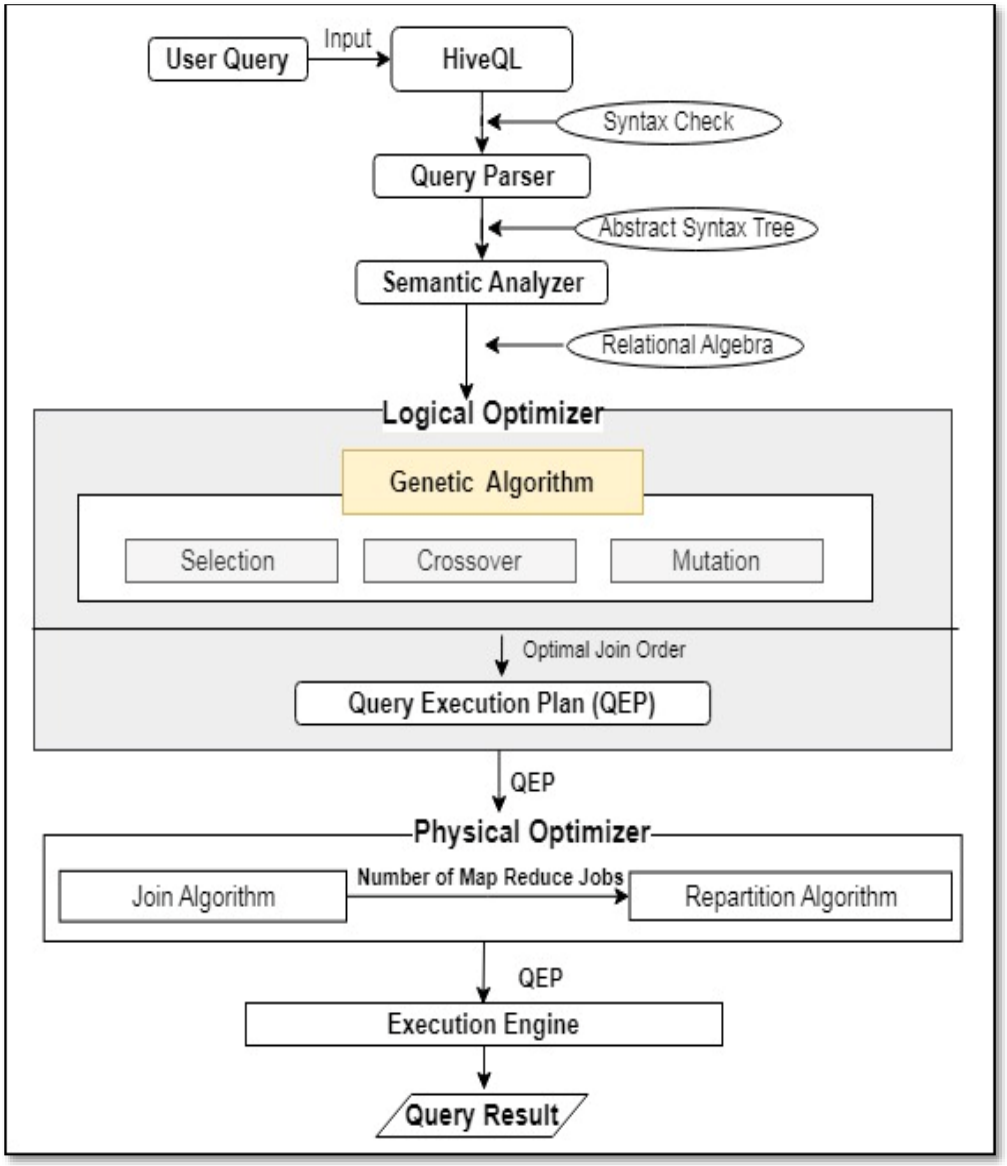


Fig -1 Logical Optimizer of HiveQL: Genetic Algorithm(GA) as Search Space

In Hive query processing (Fig-1), a Genetic Algorithm (GA) is used to find the best execution plan. The process starts when a user's query is translated into a relational algebra after analyzed parsed, validation checking by semantic analyzer. The GA then generates many possible execution plans, which are evaluated based on their performance. The best plans are selected and used to create even better ones, repeating the process until the optimal plan is found. This optimal plan is served to physical optimizer (for join algorithm like reparation algorithm) then used to execute the query, resulting in faster response times. To test the effectiveness of this approach, the GA is applied to a standardized benchmark TCP-DS dataset, providing a fair comparison with other query optimization methods in big data environments.

1.3 Working Principles of Genetic Algorithm (GA): Pseudocode & Flowchart

The Genetic Algorithm follows a structured process (Fig-2) consisting of initialization, evaluation, selection, crossover, mutation, and termination [23]. The process begins by creating an initial population of candidate solutions, where each individual represents a possible solution to the problem. The fitness of each individual is then assessed using a specified fitness function. The algorithm iterates until a termination criterion is met. In each iteration, parents are selected based on their fitness scores. These parents undergo crossover [24], where their genetic material is combined to produce offspring, creating new solutions. Mutation is then applied to introduce small random changes, maintaining genetic diversity and preventing premature convergence [25].

After mutation, the fitness of the new population is evaluated. The old population is then replaced by the new one, ensuring only the best solutions are carried forward. This process continues until the algorithm identifies the best possible solution to the optimization problem.

Pseudo code

Input: query representing the number of relations with joins
Initialize, Chromosomes (representing QEP)
Begin Create Search Space by representing each chromosome as QEP
Calculate the Fitness of each QEP based on a cost function
Repeat Until the Termination Criterion (Convergence) is not Complete

Selection
Crossover
Mutation
End Repeat
Output: The Optimal Chromosome (QEP) with minimum cost.

Flow Chart

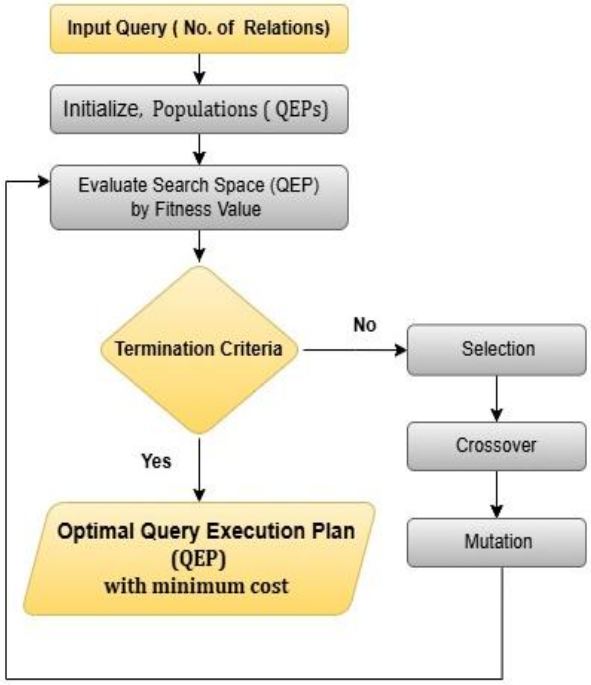


Fig -2 Genetic Algorithm (GA): Pseudocode & Flowchart

1.4 Literature Review based on selection methods for Genetic Algorithm (GA)

Genetic Algorithm (GA) is a popular class of optimization techniques inspired by natural selection, widely employed for solving complex problems [26], including those related to query optimization in databases. In the context of query optimization, GA is particularly useful for solving the join order problem, where the goal is to minimize the query execution time by determining the optimal join order [27]. A necessary component of GA is the selection process, which directly influences the algorithm's convergence rate, solution quality, and diversity. This review explores various selection methods [28] used in GA, highlighting their strengths and weaknesses, in query optimization, especially in the Hive framework in Table-1.

Selection Method	Description	Strengths	Weaknesses	References
Roulette Wheel Selection	Allocates selection probability based on fitness values mapped to a wheel.	Favors high-fitness individuals but risks premature convergence.	May lead to premature convergence and loss of diversity.	[16]
Rank Selection	Assigns selection probability based on rank, not raw fitness.	Reduces selection bias and helps maintain diversity.	Slower convergence than fitness-proportional methods.	[29]
Tournament Selection	Selects individuals via tournaments among random subsets.	Effective for maintaining diversity and handling complex landscapes.	Can slow convergence and increase complexity for larger	[28]

			populations.	
Stochastic Universal Sampling (SUS)	Selects individuals at uniform intervals based on a random starting point.	More uniform selection improves population diversity.	More complex than roulette wheel.	[20]
Boltzmann Selection	Selection pressure increases as temperature drops, focusing on optimal solutions while maintaining diversity, with an adaptive fitness function.	Balances exploration and exploitation for better convergence.	Risk of information loss; may need elitism for best solutions.	[30]
Elitism Selection	Retains the best solution from each generation for the next.	Helps preserve the best solutions, improving overall performance.	May reduce diversity if overused.	[22]

Table-1: Highlights the strength and weakness for various selection methods

According to above mentioned Table-1, Genetic Algorithms (GA) applied to the Join Order Problem (JOP) based on the Traveling Salesman Problem (TSP). Tournament Selection and Ranking Selection [31] are often favored for their ability to balance exploration and exploitation, effectively preventing premature convergence and yielding high-quality solutions. Roulette Wheel and Tournament Selection are particularly important in determining which join orders advance to recombination. While Roulette Wheel Selection [27] prioritizes high-quality solutions, it can lead to premature convergence, limiting diversity. Elitism-based Selection also plays a crucial role in retaining the best candidates through generations, significantly enhancing both convergence speed and solution accuracy. Stochastic Universal Sampling (improved version of Roulette Wheel) is effective for improving diversity, allowing for a more thorough exploration of the solution space. Julstrom [32] found that Tournament Selection is preferred over Ranking Selection, primarily because repeated tournaments are faster than arranging the population to assign ranking probabilities. Goldberg [33] compared Proportional, Ranking, Tournament, and Genitor (steady-state) selection. They concluded that Ranking and Tournament Selections are superior to Proportional Selection in maintaining steady pressure toward convergence. They also recommended Tournament Selection over Rank Selection due to its more efficient time complexity. A comparative study by Zhong [34] considered Roulette Wheel and Tournament Selection, using seven test functions. Results showed that Tournament Selection outperformed Roulette Wheel Selection, with the former yielding better performance in the SGA, while the latter resulted in average performance. Shukla [27] observed that Tournament Selection outperformed other techniques in terms of convergence rate and time complexity. In the context of JOP, adaptive and elitist tournament-based selection methods have proven to be particularly effective in navigating complex problem landscapes. These methods typically outperform others regarding both speed and the quality of the final solution, especially in combinatorial optimization problems such as JOP. The overall conclusion is that a hybrid approach, combining Elitism with Tournament or Ranking Selection, offers the most promising results for optimizing join order problem. This combination ensures a balance between exploration of the solution space and exploitation of the best solutions, leading to enhanced GA performance in these challenging optimization problems.

1.5 Proposed TRGAHH Algorithm to improves the Hive queries performance

The proposed pseudo code & flowchart (1.5.1) offers a clear, step-by-step process for optimizing Hive queries. By combining various techniques, it enhances query performance through a structured sequence of instructions. With predefined inputs and expected outputs, the TRGAHH ensures efficient execution by selecting the optimal query plans within the Hadoop-Hive framework.

1.5.1 TRGAHH (Tournament-Ranking Genetic Algorithm Hadoop Hive) Algorithm

Pseudocode &Flowchart

1. **Start Timer → Evaluate Population (fitness f(c))**
2. **Initialize Generation = 0**
3. **For each generation:**
- Calculate fitness f(c)
- Check Stagnation Limit =10
- Apply Elitism: Preserve top 2 individuals
4. **While New Population < Population Size:**
- Apply Selection (**Hybrid Tournament-Ranking)**
- Apply Crossover: Order Crossover (OX)
- Apply Mutation: Swap Mutation
- Add offspring to New Population
5. **Update Population**
6. **Get Best Chromosome**
7. **Calculate Fitness**
8. **Stop Timer**
9. **Display Results:** Execution Time or Query Response Time (QRT)
10. **End**

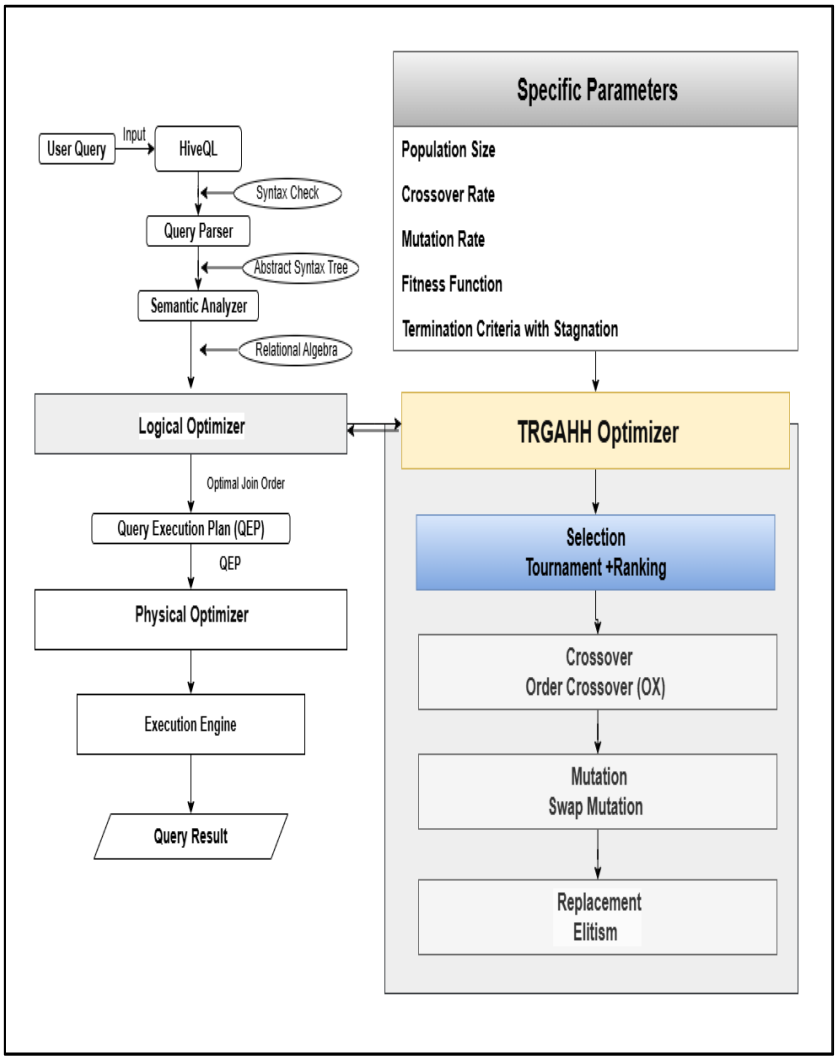


Fig-3 Flowchart (1.5.1) of the TRGAHH Algorithm

To enhance the Genetic Algorithm (GA), several key components are integrated, including Random Initialization for chromosome generation, a Fitness Function to assess solution quality based on query execution time and resource utilization, and a Hybrid Parent Selection combining Ranking and Tournament methods to balance selection pressure and maintain diversity. Additionally, the Order Crossover operator preserves gene order, the Swap Mutation operator introduces controlled randomness, and an Elitism Replacement Strategy ensures high-quality solutions are retained. These methodologies collectively form the backbone of the proposed pseudocode (1.5.1), enabling efficient optimization of Hadoop-Hive queries by reducing execution time and enhancing system performance.

1.6 Experiment Setup

- **Hardware and Software Specifications:** The experiment was conducted in a Hadoop-Hive environment using the following specifications:

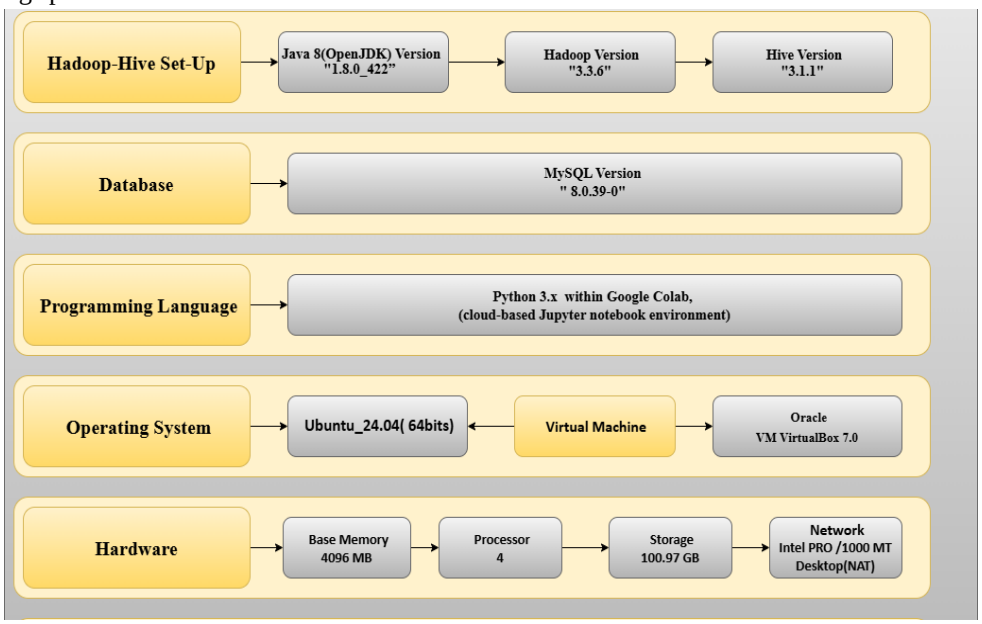


Fig-4 Hardware & Software Specification of the TRGAHH

- **Experimental Dataset:** The proposed TRGAHH algorithm using the TPC-DS dataset with 24 tables (Relations).
- **TRGAHH Parameters Metrics:**

Parameter	Value
Population Size	125
Termination Criteria (Generation)	50
Crossover Rate	0.8
Mutation Rate	0.02
Tournament Size	3
Stagnation limit	10
Queries Tested	Query 1 to Query 8 (15 to 23 Joins)

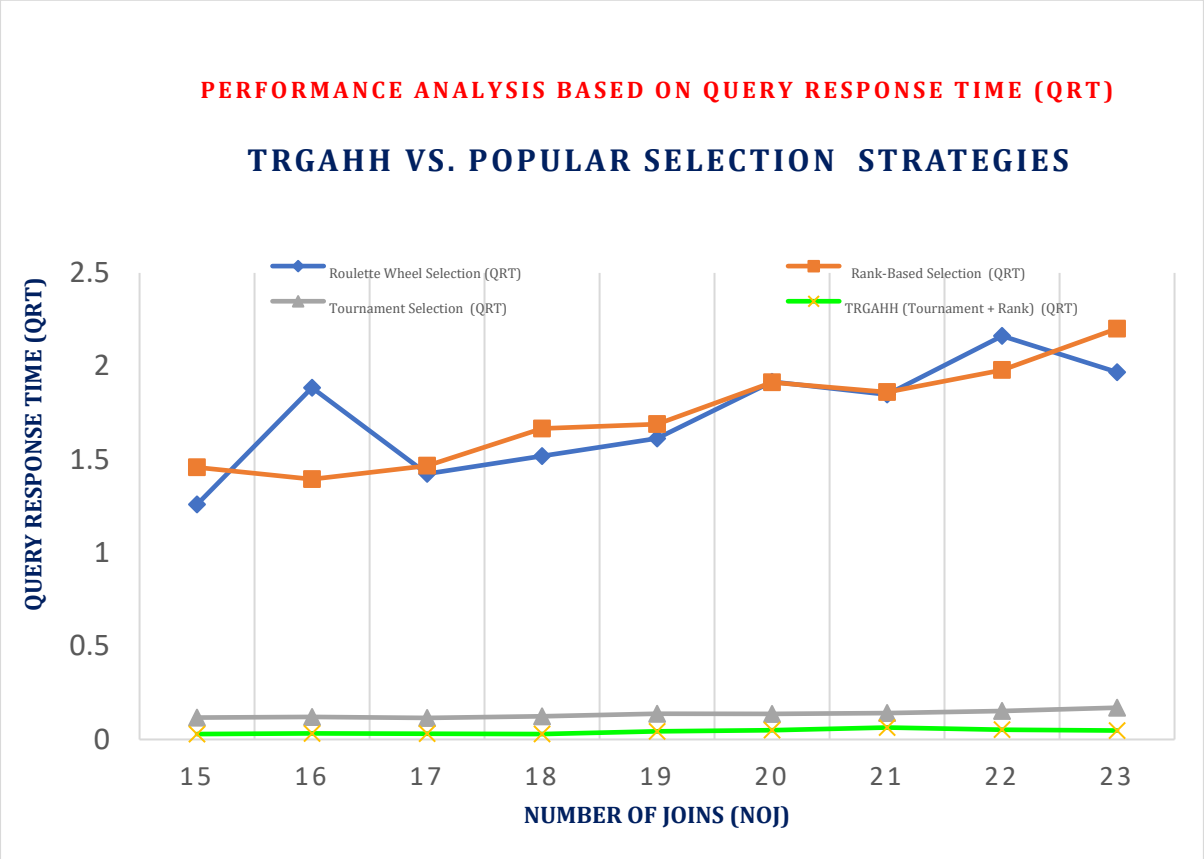
Table-2 Experimental Parameters of TRGAHH Algorithm for optimal QEP (Query Execution Plan)

1.7 Performance Comparison of Selection Strategies: Query Response Time (QRT)

The table-3 below compare the various selection methods, including Rank Selection, Roulette Wheel, Tournament Selection with TRGAHH Algorithm

TRGAHH Vs. Other Selection Strategies Average Query Response Time (QRT) Vs. Number of Joins (NOJ) (15 to 23)				
Number of Join (NOJ)	**Roulette Wheel Selection (QRT)**	**Rank-Based Selection (QRT)**	**Tournament Selection (QRT)**	**TRGAHH (QRT)**
15	**1.259606**	**1.458125**	**0.117262**	**0.029004**
16	**1.88497**	**1.394583**	**0.120666**	**0.032124**
17	**1.423566**	**1.466584**	**0.115505**	**0.030957**
18	**1.518533**	**1.666206**	**0.123837**	**0.029385**
19	**1.612921**	**1.689668**	**0.13797**	**0.043596**
20	**1.916347**	**1.913445**	**0.1369**	**0.049741**
21	**1.848451**	**1.861281**	**0.141708**	**0.06407**
22	**2.162205**	**1.97952**	**0.152753**	**0.052403**
23	**1.968148**	**2.20148**	**0.170277**	**0.047843**

Table-3: TRGAHH vs. Other Methods: QRT Performance Analysis



Graph-1: TRGAHH vs. Other Methods: QRT Performance Analysis

The performance analysis of the TRGAHH algorithm (Table-3) compared to other selection strategies (Roulette Wheel, Rank-Based, and Tournament Selection) reveals that TRGAHH consistently outperforms the others in terms of Query Response Time (QRT) across varying numbers of joins (NOJ) from 15 to 23. At 15 joins, TRGAHH has a remarkably low QRT of 0.029004, significantly better than the other methods—Roulette Wheel (1.259606), Rank-Based (1.458125), and Tournament Selection (0.117262). As the number of joins increases,

TRGAHH maintains its efficiency, with QRTs ranging from 0.029385 at 18 joins to 0.047843 at 23 joins, showing minimal variation even as query complexity grows. In contrast, the other methods experience a noticeable increase in QRT as NOJ rises. Roulette Wheel and Rank-Based Selection exhibit a steady increase in QRT, with values reaching 2.162205 and 2.20148, respectively, at 23 joins. Tournament Selection, while better than the other two, still shows a rise in QRT, reaching 0.170277 at 23 joins, which is still significantly higher than TRGAHH's performance. Overall, TRGAHH (Graph-1) stands out for its stable and lower QRT, making it the most efficient selection strategy for handling complex queries involving multiple joins.

1.8 Conclusion and Future work

In conclusion, the TRGAHH algorithm significantly outperforms traditional selection strategies—Roulette Wheel, Rank-Based, and Tournament Selection—in terms of Query Response Time (QRT) for join order optimization in big data environments. TRGAHH consistently delivers lower QRT, even as query complexity increases, with a minimal rise in response time from 15 to 23 joins. In contrast, other methods show a significant increase in QRT, making TRGAHH the most efficient choice for complex queries.

For future work, aim to integrate adaptive methods to further enhance the performance of TRGAHH. Additionally, we plan to explore its application to other big data optimization tasks, such as index selection and query workload optimization. Incorporating deep learning-assisted heuristics for parent selection could also improve the algorithm's adaptability in large-scale systems. Lastly, conducting real-world case studies with industry-scale datasets will help validate the scalability and robustness of TRGAHH in diverse big data applications.

1.9 References

1. Ketu, S., Mishra, P. K., & Agarwal, S. (2020). Performance analysis of distributed computing frameworks for big data analytics: hadoop vs spark. Computación y Sistemas, 24(2), 669-686,

2. Singh, A., Khamparia, A., &Luhach, A. K. (2019, June). Performance comparison of apachehadoop and apache spark. In Proceedings of the Third International Conference on Advanced Informatics for Computing Research (pp. 1-5).

3. Veiga, J., Expósito, R. R., Pardo, X. C., Taboada, G. L., &Tourifio, J. (2016, December). Performance evaluation of big data frameworks for large-scale data analytics. In 2016 IEEE International Conference on Big Data (Big Data) (pp. 424-431). IEEE.

4. Pal, S. (2016). SQL on Big Data: Technology, Architecture, and Innovation. Apress.

5. Nerić, V., & Sarajlić, N. (2021). Big Data Optimization Using Hive. ElektrotehniskiVestnik, 88(5), 290-298.

6. Bansal, K., Chawla, P., & Kurle, P. (2019). Analyzing performance of Apache Pig and Apache hive with hadoop. In Engineering Vibration, Communication and Information Processing (pp. 41-51). Springer, Singapore.

7. Pandey, P., & Satsangi, C. S. (2019). Comparative performance evaluation using hadoop ecosystem–pig and hive through rendering of duplicates. In International Conference on Advanced Computing Networking and Informatics: ICANI-2018 (pp. 89-95). Springer Singapore.

8. Patel, N. (2019) Evaluating the Performance of Apache Hive and Apache Pig using Hadoop environment.

9. Choudhary, A., & Satsangi, C. S. (2015). Query Execution Performance Analysis of Big Data Using Hive and Pig of Hadoop. International Journal of Computer Sciences and Engineering, 3(9), 91-97.

10. Thusoo, A., Sarma, J.S., Jain, N., Shao, Z., Chakka, P., Anthony, S. and Murthy, R. (2009) 'Hive: a warehousing solution over a map-reduce framework', Proceedings of the VLDB Endowment, Vol. 2, No. 2, pp.1626–1629.

11. Thusoo, A., Sarma, J. S., Jain, N., Shao, Z., Chakka, P., Zhang, N., ... & Murthy, R. (2010, March). Hive-a petabyte scale data warehouse using hadoop. In *2010 IEEE 26th international conference on data engineering (ICDE 2010)* (pp. 996-1005). IEEE.

12. Shanoda, M. S., Senbel, S. A., &Khafagy, M. H. (2014, December). Jomr: Multi-join optimizer technique to enhance map-reduce job. In 2014 9th International Conference on Informatics and Systems (pp. PDC-80). IEEE.

13. Pang, Z.,Wu, S., Huang, H., Hong, Z., & Xie, Y. (2021). AQUA+: Query Optimization for Hybrid Database-MapReduce System. Knowledge and Information Systems, 63(4), 905-938.

14. Wu, S., Li, F., Mehrotra, S. and Ooi, B.C. (2011) 'Query optimization for massively parallel data processing', Proceedings of the 2nd ACM Symposium on Cloud Computing, Cascais, Portugal, p.12.

15. Rabaaoui, S., Aloui, K., Naceur, M. S., &Barkaoui, K. (2024, April). Genetic Algorithm-Based Approach for Optimizing Query Performance in Big Data Environments. In 2024 IEEE International Conference on Advanced Systems and Emergent Technologies (IC_ASET) (pp. 1-6). IEEE.

16. Blickle, T., & Thiele, L. (1995). A comparison of selection schemes used in genetic algorithms.

17. Katoch, S., Chauhan, S. S., & Kumar, V. (2021). A review on genetic algorithm: past, present, and future. Multimedia tools and applications, 80, 8091-8126.

18. Kaya, Y., & Uyar, M. (2011). A novel crossover operator for genetic algorithms: ring crossover. *arXiv preprint arXiv:1105.0355*.

19. Lim, S. M., Sultan, A. B. M., Sulaiman, M. N., Mustapha, A., & Leong, K. Y. (2017). Crossover and mutation operators of genetic algorithms. International journal of machine learning and computing, 7(1), 9-12.

20. Lambora, A., Gupta, K., & Chopra, K. (2019, February). Genetic algorithm-A literature review. In 2019 international conference on machine learning, big data, cloud and parallel computing (COMITCon) (pp. 380-384). IEEE.

21. Luo, X., & Fu, X. (2019). Configuration optimization method of Hadoop system performance based on genetic simulated annealing algorithm. Cluster Computing, 22, 8965-8973.

22. Sun, L. (2015, March). Genetic Algorithm for TSP Problem. In 2015 International Industrial Informatics and Computer Engineering Conference (pp. 1436-1439). Atlantis Press.

23. Alam, T., Qamar, S., Dixit, A., &Benaida, M. (2020). Genetic algorithm: Reviews, implementations, and applications. arXiv preprint arXiv:2007.12673.

24. Pachuau, J. L., Roy, A., & Kumar Saha, A. (2021). An overview of crossover techniques in genetic algorithm. Modeling, Simulation and Optimization: Proceedings of CoMSO 2020, 581-598.

25. Hong, T. P., Wang, H. S., Lin, W. Y., & Lee, W. Y. (2002). Evolution of appropriate crossover and mutation operators in a genetic process. Applied intelligence, 16, 7-17.

26. Sivanandam, S. N., Deepa, S. N., Sivanandam, S. N., & Deepa, S. N. (2008). Genetic algorithm optimization problems. *Introduction to genetic algorithms*, 165-209.

27. Shukla, A., Pandey, H. M., & Mehrotra, D. (2015, February). Comparative review of selection techniques in genetic algorithm. In 2015 international conference on futuristic trends on computational analysis and knowledge management (ABLAZE) (pp. 515-519). IEEE.

28. Saidi, R., Bouaguel, W., &Essoussi, N. (2019). Hybrid feature selection method based on the genetic algorithm and pearson correlation coefficient. Machine learning paradigms: theory and application, 3-24.

29. Tiwari, P., & Chande, S. V. (2019). Join Query Optimization Using Genetic Ant Colony Optimization Algorithm for Distributed Databases. In Emerging Technologies in Computer Engineering: Microservices in Big Data Analytics: Second International Conference, ICETCE 2019, Jaipur, India, February 1–2, 2019, Revised Selected Papers 2 (pp. 224-239). Springer Singapore.

30. Bajaj, A., & Sangwan, O. P. (2019). A systematic literature review of test case prioritization using genetic algorithms. Ieee Access, 7, 126355-126375.

31. Chande, S. V., & Sinha, M. (2009). Genetic algorithm: a versatile optimization tool. BIJIT-BVICAM's International Journal of Information Technology, 1(1), 7-12.

32. Julstrom, Bryant A. "It's all the same to me: Revisiting rank-based probabilities and tournaments." Evolutionary Computation, 1999. CEC 99. Proceedings of the 1999 Congress on. Vol. 2. IEEE, 1999

33. Goldberg, D. E., & Holland, J. H. (1988). Genetic algorithms and machine learning. Machine learning, 3(2), 95-99.

34. Zhong, J., Hu, X., Zhang, J., & Gu, M. (2005, November). Comparison of performance between different selection strategies on simple genetic algorithms. In *International conference on computational intelligence for modelling, control and automation and international conference on intelligent agents, web technologies and internet commerce (CIMCA-IAWTIC'06)* (Vol. 2, pp. 1115-1121). IEEE.