ISSN: 0067-2904

Optimizing Security and Response Time of Relational Database Using Moth-Flame Algorithm

Murtada Mohammed Hasan Alsayyad*, Pedram Salehpour
Department of Computer Engineering, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

Received: 28/1/2025 Accepted: 30/6/2025 Published: 30/10/2025

Abstract

Query optimization in relational databases is a costly process, and the number of different permutations of link operations for a query grows exponentially as the number of tables in the query increases. Current query optimization techniques are unsuitable for use in cases where the number of database tables and the number of rows in each table are large. On the other hand, the key to the success of a database system is the efficiency of its query model. Therefore, this research proposes a method for optimizing query execution by rearranging query structures.
Due to the increasing importance of reducing the execution time of the optimal design in the connection operation with a large number of tables and minimizing the execution time of the algorithm to find the optimal design and not getting stuck in local optimizations, the use of new metaheuristics and efficient algorithms has received much attention. In this research, we used the Genetic algorithm and the Moth Flame Optimization algorithm. Finally, we test our proposed method on the employee database with 200 random queries. In comparison to random and GA runs, our method gets 22.5% and 4.2% better response time, respectively.

Keywords: query optimization, relational database, Moth Flame Optimization, Response time, Genetic Algorithm.

تحسين الأمان ووقت الاستجابة لقواعد البيانات العلائقية باستخدام خوارزمية عثة اللهب

مرتضى محمد حسن عبد الصياد *, پدرام صالح پور
قسم هندسة الحاسوب، كلية هندسة الكهرباء والحاسوب، جامعة تبريز، تبريز، ايران

الخلاصة

يعد تحسين الاستعلامات في قواعد البيانات العلائقية عملية مكلفة، ويتزايد عدد التباديل المختلفة لعمليات الارتباط للاستعلام بشكل كبير مع زيادة عدد جداول مشاركة الاستعلام. إن تقنيات تحسين الاستعلامات الحالية غير مناسبة للدعم في الحالات التي يكون فيها عدد جداول قاعدة البيانات وعدد الصفوف في كل جدول كبيرًا. من ناحية أخرى، فإن مفتاح نجاح نظام قاعدة البيانات هو كفاءة نموذج الاستعلام الخاص به. لذلك، في هذا البحث، تم اقتراح طريقة لتحسين تنفيذ الاستعلامات من خلال إعادة ترتيب بنية الاستعلامات.
نظرًا للأهمية المتزايدة لتقليل وقت تنفيذ التصميم الأمثل في عملية الاتصال مع عدد كبير من الجداول وتقليل وقت تنفيذ الخوارزمية من أجل العثور على التصميم الأمثل وعدم التعثر في التحسينات المحلية، فقد حظي استخدام خوارزميات ميتاهيوريستية الجديدة والفعّالة باهتمام كبير. في هذا البحث، استخدمنا الخوارزمية الجينية

*Email: murtada.alsayyad@gmail.com

وخوارزمية تحسين عثة اللهب. أخيرًا، اختبرنا طريقتنا المقترحة على قاعدة بيانات الموظفين باستخدام 200 استعلام عشوائي. بالمقارنة مع التشغيلات العشوائية والخوارزميات الجينية، تحصل طريقتنا على وقت استجابة أفضل بنسبة 22.5٪ و4.2٪ على التوالي.

1. Introduction

Optimizing queries in the general case and the problem of ordering links in the specific case are one of the most important criteria for the success of the database management system, whose task is to select an optimal order to link the tables with the lowest execution cost [1]. A query such as q entered by the user is considered input to this system. Assuming that S is a set of all possible strategies for responding to the query q and each member of S, like x, has a $cost(x)$, its cost function can have different forms and purposes, such as shortening response time, input, and output consumption, CPU consumption, memory consumption, resource consumption, such as the battery in mobile-based environments, etc., or a combination of the above. The goal of an optimization algorithm is to find a member s_0 from S such that:

$$Cost(S) = min\, Cost(s_0), s_0 \in S \tag{1}$$

The link is a commutativity and associativity operator. Therefore, the number of strategies for responding to a query increases exponentially with the increase in the number of links. For example, although the following two statements are equivalent and have the same output, their time cost may not be equal.

$$(R_1\, join\, R_2)\, join\, R_3 = R_1\, join\, (R_2\, join\, R_3) \tag{2}$$

For example, for four relationships, and given that in the practice of bonding, the property of commutativity and associativity is established, there are 12 possible arrangements. In the general case for n relations, the number of possible sequences is equal to:

$$\frac{(2(n-1))!}{(n-1)!} \tag{3}$$

Finding an optimal order for links with n relations can be examined in different ways. Prove that this problem is an NP-hard problem. So far, different methods have been proposed [2].

Definitive Algorithms. All algorithms in this classification construct a step-by-step solution definitively, either by taking the initiative or by completing a search.

Random Algorithms. Algorithms in this classification look for a completely different method. A set of movements is defined. These movements form the edges between the different solutions of the solution space. The two solutions are connected by an edge if they can be converted to another with exactly one movement.

Metaheuristic algorithms. These algorithms are usually inspired by nature and the life of creatures; they try to approach the optimal answer by simulating the life of creatures.

1.1 Query execution

In Figure 1 the query execution steps are shown. First, syntax analysis and semantic confirmation are done. The next query optimizer is executed. Query optimization is the process of selecting the most efficient means of executing a SQL statement [3].

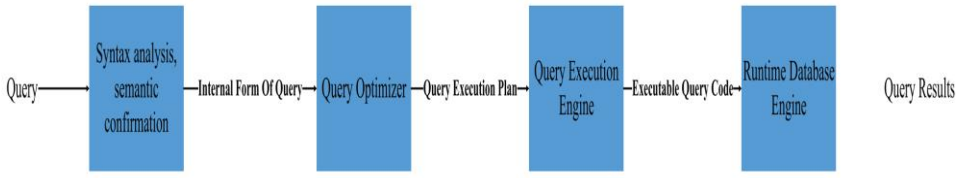


Figure 1: Query execution

1.2 Database

In order to test our proposed method, the Employees database has been used. This database consists of 6 tables and about 3000000 rows. Queries are designed as a left-deep tree [4]. In Figure 2 and 3, a sample of the Employees database is shown.

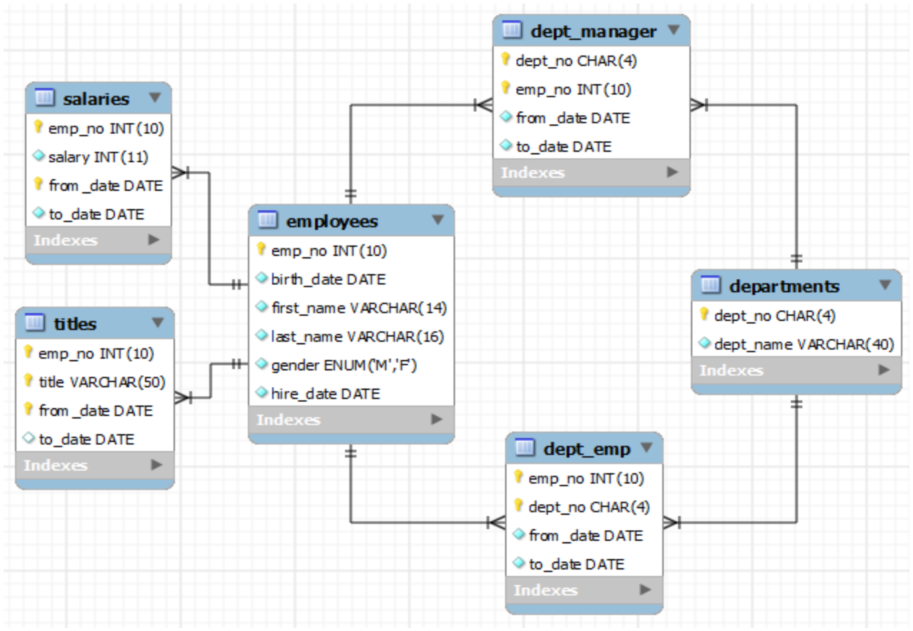


Figure 2 : Example of employee database

Queries that are performed on relational databases are based on relational algebra. In relational algebra, there are eight commonly used operators whose result is defined as a relation as follows. Note that for union, intersection, and difference operators, the important thing about relations used as operands is that these relations must have the same set of properties.
Union operator. This operator is binary (binary or two operands) and will include all the tuples in its operands. In the resulting relation, duplicate rows are removed.
Intersection operator. The result of this binary operator will include a set of tuples that are common to both relations.
Difference operator. The result of this binary operator will include tuples from the first relation that do not exist in the second relation.
Cartesian Product. Each tuple resulting from this binary operator is formed by concatenating a tuple from the first relation and a tuple from the second relation.
Selection operator. An operator that returns only tuples from its relation that meet a certain condition or criterion.
Projection operator. It is the only operator that extracts only the specified features from the available tuples.
Join operator. The most common type of this operator is known as Natural Join. The relation resulting from this binary operator includes all the properties in the relations of its two operands. It is formed by concatenating tuples of two relations with the same values on their common properties.
Division operator. The result of this binary operator will only include features from the first relation that are not present in the second relation. The tuples of the resulting relationship will also include the values of these attributes in the tuples of the first relation, whose combination exists with all the tuples of the second relation in the first relation.

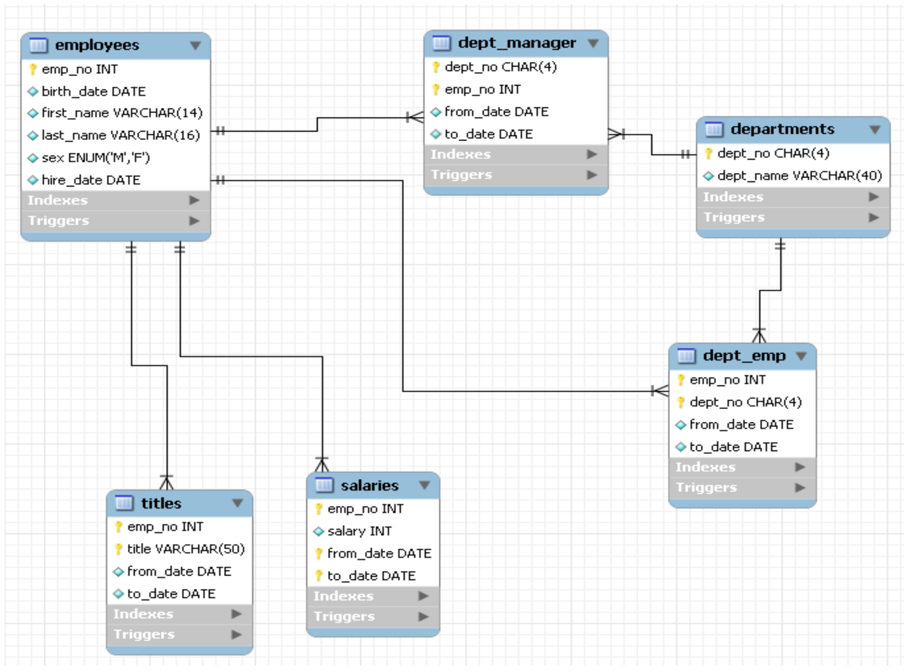


Figure 3: Sample of employee database

1.3 Access control mechanisms

To achieve security goals such as privacy and preventing unauthorized access, accessibility, and comprehensiveness, security policies must be clear and transparent, and the part of the information that needs protection and who can access the data must be defined. can have access should be clear [5]. Today, one of the most important issues in integrated systems is the management and control of access to resources. Access control is the clearest symbol of security and the most important component in establishing information security because access control mechanisms form the first line of defence against unauthorized access to protected information assets. NoSQL databases are classified into four groups: columnar databases, document-oriented databases, graph databases, and key-value databases. MongoDB is a document-oriented database that manages a collection of documents. It supports complex types of data and has high access speed for very large data. Flexibility and speed are features of this database [6]. All the data and documents in MongoDB are stored in plain text format, and there is no encryption mechanism for its data files, which provides the opportunity to access information for fraudulent people. MongoDB uses the SSL protocol to authenticate and establish a secure connection between the user and MongoDB. However, it does not support authentication and authorization when running in Shared Mode. Passwords are encrypted with an MD5 hash and MD5 algorithms, which are not very secure. Since MongoDB uses JavaScript as an internal scripting language, it is very vulnerable to injection attacks.

CouchDB is a document-oriented database in which flexibility and fault tolerance are its features. it is an open-source project and runs on Hadoop. CouchDB does not support data encryption, but it supports authentication based on cookies and passwords. Passwords are encrypted with the PBKDF2 hash algorithm and sent over the network with the SSL protocol. CouchDB is also vulnerable to injection and denial of service attacks. Cassandra is a distributed and open-source storage for managing large data, which is the type of key-value database used in Facebook. Its features include high flexibility and high scalability. All passwords in Cassandra are encrypted with the MD5 hash algorithm and are very weak. Suppose any fraudulent user wants to steal information, there is no authentication mechanism between the nodes where data is transferred. Cassandra is vulnerable to denial-of-service attacks. The query language used in Cassandra (CQL) is similar to the SQL query language, so it is subject to injection attacks like SQL.

1.4 Database Security

In today's world, database management systems (DBMS) are used as an integral part of various organizations and companies. For this reason, database security is one of the most important issues that businesses need to pay special attention to. In the following, the topics related to the security of databases are stated. Database security refers to the various measures that organizations use to ensure that their databases are protected against internal and external threats. Database security refers to the protection of the database itself, the data in it, the related database management system, and various applications that have access to the database. Organizations must secure databases against various intentional attacks, such as network security threats and misuse of data and databases. Over the past few years, the number of information breaches and law-breaking in this field has increased significantly. In addition to the significant damage that these threats cause to a company's reputation and credibility, there are various regulations and penalties for data breaches, and organizations must deal with the challenge of data breaches. One of these is the General Data Protection Regulation (GDPR), which is often very costly. Considering the mentioned points, we can consider effective database security as a key to consistency, protecting the reputation of organizations, and keeping their customers.

Data in transport and access control in database security. In general, (Access Control) of data in transportation refers to a special security system, with the help of which the necessary assurance of the transfer process will be achieved. Simply put, with this type of database security control, no one can read or interpret data as it moves between different servers or network configurations. The main goal of this type of database security is to limit any potential node related to intrusion or unauthorized access to server systems at any time. Therefore, these data settings are also known as access control. Every single node of data entering and leaving the secure server system is completely encrypted and unreadable unless it is securely deposited in a secure system database or displayed to a user requesting that data.

Authentication in database security controls. Authentication is the next type of database security, and after completing the data, it is necessary to apply this issue in the transport protocol. This security protocol has different layers within it. In general, authentication is a way to verify whether a user is who they say they are. In simpler terms, Authentication means authenticating the request or query sent by authorized personnel or a dedicated user. In order to implement authentication, different methods can be used. For example, using the multi-factor authentication method (multi-factor), where different security layers are added in combination. This process leads to the authentication of a particular user and their successful access. If the authentication process is not implemented as a practical practice in database configuration and security, anyone, even illegal hackers, can easily access database servers and cause crashes, and compromise database security. To grant access and effective user authentication, things like two-factor authentication and authentication through username and password can be used.

Licensing in database security controls. The next step in this process and the third type of maintaining database security is authorization. By applying this security layer, it is clear what elements the dedicated user has access to. If necessary, restrictions can be applied to a specific user, and their access is limited to only an overview of the systems. For example, a user may have access to general website content, but sensitive confidential information such as personal or financial information of other users may be restricted to him or any other Guest or regular user. This phase of database security is the most important of them all. Due to this, the necessary confidence to establish the security of the database is obtained. In fact, by using

Authorization, no one can wander into unknown areas or explore areas that are not meant to be noticed. The permission level assigned to a specific user can be configured or customized for a specific organization or application.

2. Related Works

Much research is done on query optimization in distributed database systems. In [7], a method based on an artificial bee colony algorithm and genetic operators is proposed. They have benefited from genetic algorithm operators to improve their work while using artificial bee colony algorithms. In this way, they have prevented the bee algorithm from getting stuck in the local optimum. Database query optimization with comprehensive search techniques, such as dynamic programming, is suitable for queries with a small number of relationships, but with the increase in the number of relationships in the query, due to the need for high memory and processing, the use of these methods will not be appropriate.

In [8], Trumme et al. present SkinnerDB, a novel database management system that is designed from the ground up for reliable optimization and robust performance. SkinnerDB implements several adaptive query processing strategies based on reinforcement learning. We divide the execution of a query into small periods in which different join orders are executed. Thereby, we converge to optimal join orders with regret bounds, meaning that the expected difference between actual execution time and time for an optimal join order is bounded. This method features a mature, cost-based query optimizer. It is therefore representative of the strengths and weaknesses of traditional optimizers and cannot make up for the effects of badly chosen join orders.

In [9], Mancini et. al. explained a method for join order optimization for queries with a large number of joins. their query optimization method can implement parallelism while significantly pruning the search space, and can be efficiently implemented on GPUs too. Their results, in the case of a defined scenario, demonstrate that their techniques are significantly better than other state-of-the-art techniques in terms of query optimization time. their heuristic method is capable of efficiently exploring a larger search space for very large join queries (e.g., 1000 rels), thereby allowing them to find plans with much better costs compared to state-of-the-art heuristic techniques, and handles data skew and large join trees effectively. The Disadvantages of this method include high optimization overhead for very large query plans and may not perform well in highly dynamic environments.

In [10], Swidan et al. present a survey among the most important research projects in the field of query processing concerning both conventional and preference queries in Crowd-sourcing database systems (CSDBs). Crowdsourcing is a powerful solution for finding true solutions to costly and unanswered queries in databases, including those with unknown and imperfect data. Using crowd-sourcing to exploit human abilities to process these costly queries, human workers have helped to provide accurate results by utilizing the available data in the crowd. CSDBs merge the knowledge of the crowd with a relational database by using some variant of a relational database with minor changes. The concentration of this study is on highlighting the advantages and disadvantages of different approaches in the field of crowd-sourcing. One of the problems facing this method is that human input is slow and costly compared to automated processing.

In [11], Bachav et al. offered a materialized intermediate view query optimizer that is effective. There are two processes involved in optimizing database queries: creating a search space and choosing the best plan from the available search space. By getting rid of frequently used sub-expressions, queries can run faster. By cutting down on query evaluation and response times, they create an intelligent query optimizer that enhances resource usage in cloud and distributed contexts. To reuse them for the execution of subsequent queries, the suggested query optimizer materializes intermediate views during query processing. Therefore, it reduced I/O operations, communication costs, and execution time. The method significantly improved query response time, but requires extra storage for materialized views, and may not benefit queries with low reuse potential. Their query optimizer architecture is shown in Figure 4.

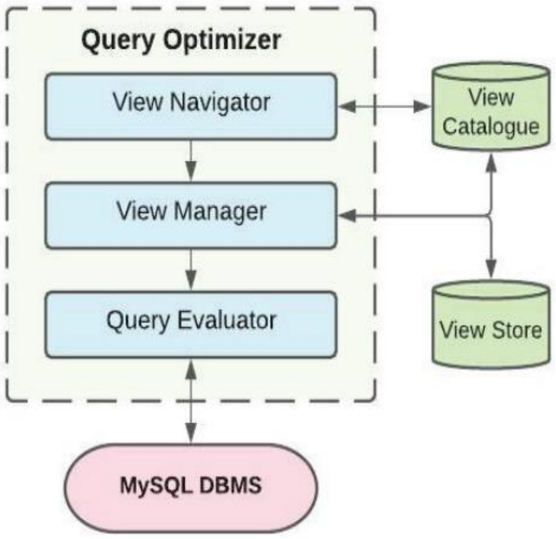


Figure 4 : The architecture of the query optimizer in [12]

In [13], V. Kumar et al. proposed the combination of modified Ant Colony Optimization with a Genetic Algorithm to get better results for multi-join query optimization and experimentally depicted that their proposed technique on the randomly created database had better performance over GA and ACO in terms of execution time. The structure of query processing is shown in Fig. 5. ACO is used to explore optimal join orders by simulating pheromone trails, while GA enhances solution quality through selection, crossover, and mutation for faster convergence. Advantages of this method include producing near-optimal join sequences, balancing exploration and exploitation efficiently, and scaling well with complex query trees. Disadvantages (i) high computational cost in large search spaces, (ii) requires careful parameter tuning.

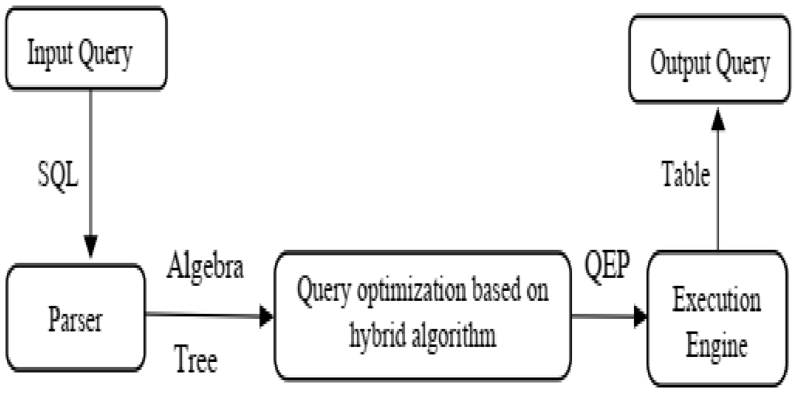


Figure 5: Query processing [14]

In [15], the authors proposed Diagnostic Multi-Query Graphs (DMG), whose goal is to reach query optimization and graph optimization techniques. The Fault Diagnostic Queries (FDQs) that are a component of every DMG node undergo query optimization. For FDQ optimization, the optimum query execution strategies and access techniques are chosen. Our DMG is also subjected to graph optimization without compromising the FDQ semantics. The suggested methods are evaluated using a variety of parameters, such as (i) FDQs and (ii). Network topologies include (iii) Bushy Tree (BT) and Left Deep Tree (LDT). This method is predicated on graph node merging and pruning. Scheduling and fault detection within the specified timeframe are more likely for an efficient DMG with fewer processing nodes and an optimized QEP. Their steps in the proposed method are illustrated in Fig. 6. Some challenges facing this method are high complexity for large query graphs, Static scheduling may lack flexibility, and be sensitive to changes in task execution times.

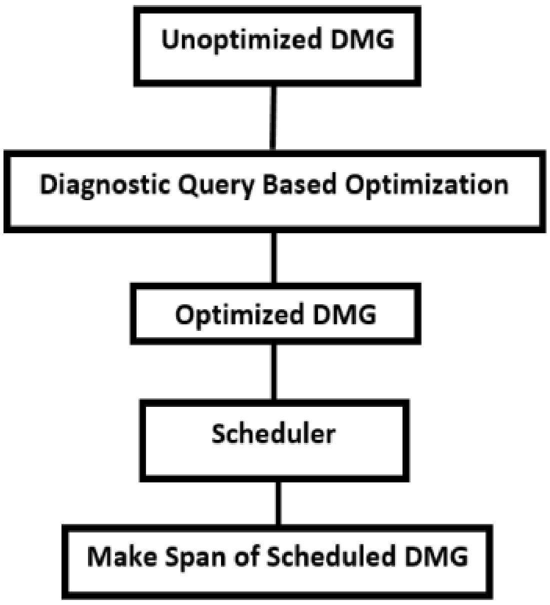

Unoptimized DMG
Diagnostic Query Based Optimization
Optimized DMG
Scheduler
Make Span of Scheduled DMG

Figure 6: DMG process

In [16], P. Patidar et al. 3 examine the three main approaches to SQL query optimization: Genetic Algorithms (GA), Ant Colony Optimization (ACO), and Heuristic Greedy Optimization. By using transformation rules including early selection and projection, predicate pushdown, and sub-query nesting, heuristic optimization enhances query performance. In ACO, artificial agents follow and reinforce pheromone trails to find optimal or nearly optimal join orders during query execution, simulating the natural behaviour of ants. To identify effective answers, genetic algorithms use processes like crossover, mutation, and selection to evolve populations of candidate query plans. This approach is biologically inspired. Every approach has advantages and disadvantages.

Although the heuristic technique is quick and easy, it might not always identify the optimal execution strategy. Although ACO has a greater processing overhead and necessitates careful adjustment of parameters like pheromone intensity and iteration limitations, it is effective at exploring huge search spaces and handling complex query formats. Similar to this, genetic algorithms can be resource-intensive and may require exact control over population size, crossover, and mutation rates, but they are useful for resolving complicated, non-linear optimization issues. All approaches contribute significantly to query optimization, but in some contexts, they also pose flexibility and scalability issues. [17]. To improve join query performance in remote databases, the paper suggests a hybrid optimization technique dubbed MOGABAT (Multi-Objective Genetic Algorithm with BAT). This method eliminates impractical solutions and enhances the quality of query plans by combining the crossover and mutation operations from genetic algorithms with the exploration capability of the BAT algorithm. Join queries are divided into four categories- chain, cycle, clique, and

star- based on their join graph architectures to more effectively target optimization tactics. This method's primary benefits include its capacity to generate excellent, workable execution plans, balance local refinement with global exploration, and efficiently adjust to different query formats. Its high computational cost, susceptibility to parameter adjustment, and potential for performance variance based on query complexity are its drawbacks.
In [18], Z. He et al. study uses Graph Neural Networks (GNNs) to present a novel approach for query execution time prediction in graph databases, specifically focusing on systems such as Neo4j. Three main phases make up the suggested framework's operation. First, a variety of queries are created and run to gather execution plans and the performance statistics that go along with them. To properly represent their structure, these execution plans are then converted into graph-based models. To identify patterns and generate precise forecasts regarding the execution timings of future queries, a GNN model is subsequently trained on these graph representations. The strategy has several benefits. It promotes better system resource allocation by offering accurate performance forecasts, adapts well to various query forms, and increases estimation accuracy by modeling complicated query dependencies. It does have certain limits, though. GNN training is a computationally demanding process that significantly depends on the caliber and scope of the training material. Furthermore, because of the intricacy of implementation and the requirement for knowledge of machine learning and graph-based data structures, integrating such a system into already-existing databases can be difficult.

In [19] L. Woltmann et al., to improve the efficiency of learning query optimizer hints in PostgreSQL, a novel approach is presented in the paper FASTgres. Particularly for complex queries, traditional query optimizers can have trouble selecting the best plans. Better plans can be suggested by current learning-based techniques, but it has proven challenging to smoothly integrate them into optimizers that are already in place. By employing machine learning to produce useful clues that direct the optimizer toward more optimal execution plans, FASTgres overcomes this difficulty. By recommending changes like join order hints based on a model developed from prior query workloads, it acts in unison with the optimizer rather than taking its place. To progressively improve recommendations, these hints are then assessed using PostgreSQL's explain functionality. Improved query performance without changing the core optimizer, quicker convergence to optimal plans, and compatibility with actual database systems are the key benefits of this approach. When compared to native optimizers, it also drastically cuts down on the execution time of complicated queries. Limitations, however, include the difficulty of generalizing to unknown query structures, the potential overhead from repeated hint evaluations, and the reliance on past query data for training. Additionally, the accuracy of PostgreSQL's cost estimation, which could still add bias, determines how effective the strategy is. Table 1 shows a summary of related work.

Table 1: Comparison between related work in terms of method, approach, advantages, and disadvantages

Rf.	Method name	Approach	Advantages	Disadvantages
[7]	ABC with Genetic Operators	Combines Artificial Bee Colony with GA operators to avoid local optima	Enhances global search efficiency	Not scalable for queries with many relationships
[8]	SkinnerDB (Reinforcement Learning)	Adaptive execution using join order switching with regret bounds	Learns optimal plans over time	Dependent on past execution, can't fix bad initial join order choices
[9]	Parallel Heuristic on GPU	Heuristic pruning + parallel implementation for large queries	Efficient with 1000+ joins; handles skew	High optimization overhead; less effective in dynamic environments
[10]	Crowdsourced Query Processing	Uses human workers to solve complex/incomplete queries	High accuracy in tough queries	Slow, costly, limited scalability
[11]	Materialized Intermediate Views	Stores reusable sub-queries for faster future execution	Reduces I/O, execution time, and resource consumption	Extra storage needed; limited reuse = limited gain
[13]	Modified ACO + GA	ACO explores join orders; GA optimizes further with crossover/mutation	Near-optimal join sequences; balanced search	High computational cost; needs fine-tuned parameters
[15]	Diagnostic Multi-Query Graph (DMG)	Optimizes FDQs via graph node merging/pruning	Effective for fault detection and resource-efficient execution plans	Complex for large graphs; inflexible to runtime changes
[16]	GA, ACO, Heuristic Comparison	Compares GA, ACO, and heuristic techniques for query optimization	Highlights contextual strengths of each method	Each has limits—no universal best; GA/ACO need tuning, heuristics miss the global optimum
[17]	MOGABAT (GA + BAT)	Hybrid algorithm using GA for refinement and BAT for exploration	Adapts to query structures (chain, cycle, etc.); high-quality plans	High computation cost; sensitive to parameter tuning
[18]	GNN-based Query Execution Time Estimation	Uses Graph Neural Networks (GNNs) to model query execution plans as graphs and predict execution time	- Accurate performance forecasting - Adapts to different query types - Models complex dependencies	- High computational cost for training - Depends on data quality and volume - Complex to integrate with existing systems
[19]	FASTgres	Machine learning-based hint generation for PostgreSQL optimizers; works in tandem with existing optimizer	- Improves query performance - Requires no changes to the core optimizer - Efficient for complex queries	- May not generalize well to new queries - Repeated hint evaluations cause overhead - Relies on PostgreSQL cost estimates and historical data

3. Discussion and Results

3.1 Proposed Optimization Algorithm

The moth flame optimization algorithm (MFO) evolved from the lateral positioning and navigation mechanism of moths in nature [20]. Moths utilize the far-off moon as a reference when they fly at night, and the moonlight can be thought of as parallel light. The moth's distance from the flame varies repeatedly while maintaining a constant angle to it, creating a

flight path that spirals closer to the flame. The MFO method offers good overall qualities and a significant capacity for parallel optimization. For non-convex functions, since these functions have a large number of local optimal points, the MFO algorithm can widely explore the search space and find that there is a higher probability of global optimal points.
Even though the transverse orientation works well, butterflies typically fly in a spiral pattern around the lights. In actuality, artificial lighting can deceive butterflies into exhibiting these behaviours. This is because lateral orientation is inefficient and only works well for traveling in a straight line when the light source is far away. Butterflies attempt to fly straight when exposed to artificial light by maintaining a similar angle to the light. Because such light is so close compared to the moon, maintaining a similar angle to the light source makes a spiralling flight path useless or fatal for butterflies. A conceptual model of this behaviour is shown in Fig. 7. It may be seen in Fig. 7 that the propeller eventually converges toward the light. This behaviour is mathematically modelled to create an optimizer called the flame optimization (MFO) algorithm or the plug-and-propeller algorithm.

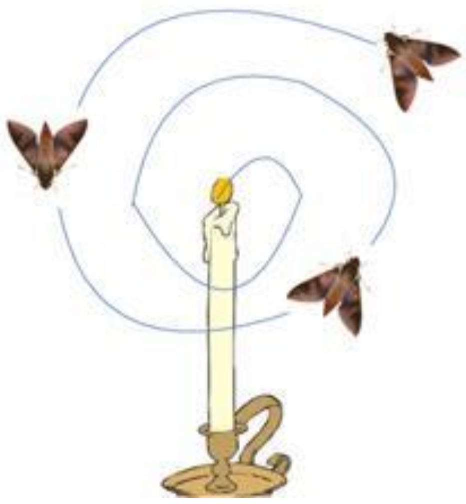


Figure 7: Moth flames

The moth is represented in the moth flame algorithm by m. Assuming that m is a potential remedy for the issue at hand, within the realm of feasibility, and the vector location of m is employed to symbolize the variable that has to be resolved, then m may be possible. The flight route is the range of the solution, and its position coordinates are the potential solutions, when flying in the domain in one or even more dimensions. Since the MFO algorithm essentially belongs to the category of swarm intelligence optimization algorithms, the moth population can be represented by the following matrix (4):

$$M = \begin{bmatrix} m_{1,1} & \cdots & m_{1,d} \\ \vdots & \ddots & \vdots \\ m_{n,1} & \cdots & m_{n,d} \end{bmatrix} \tag{4}$$

Where n is the number of moths and d is the dimension size, that is, the number of control variables to be found. The fitness value of the moth is stored in the relevant array, and the fitness value corresponding to the $i-th$ Moth is OM_i, which is expressed as follows in the array *OM*:

$$OM = \begin{bmatrix} OM_1 \\ \vdots \\ OM_n \end{bmatrix} \tag{5}$$

The position of each Moth after flight is substituted into its corresponding fitness function, and the return value of the function becomes the fitness value, that is, each parameter of the matrix M is passed into the fitness function, and the fitness value of each moth is returned, which are collectively represented as a matrix OM. There are several moths in the population, and each one has a matching flame that it flies along to update its position according to the MFO algorithm. This sole matching technique won't work when several moths are made to fly around a flame simultaneously, and the moths can fully search the global exploration space, stay out of the local optimal scenario, and improve their optimization skills. It follows from this that the flame and the moth have the same dimensions. If F is used to represent the flame, the variable matrix that describes the flame's location within the search space is as follows:

$$F = \begin{bmatrix} F_{1,1} & \cdots & F_{1,d} \\ \vdots & \ddots & \vdots \\ F_{n,1} & \cdots & F_{n,d} \end{bmatrix} \tag{6}$$

where n is the number of flames and d is the dimension size, that is, the number of control variables to be determined. The M and F matrices are the same size. The fitness value of the flame is given in the following formula:

$$OF = \begin{bmatrix} OF_1 \\ \vdots \\ OF_n \end{bmatrix} \tag{7}$$

In the MFO algorithm, flames and moths have similar functions and can be used as candidate solutions to solve the problem, but they are distinguished by their difference. The main difference is that the flame is updated, and the fitness function of the substitution is different. Essentially, moths represent individuals who are constantly searching in the feasible domain, and the local optimal positions searched by moths are replaced by flames, each time the position of the flame is updated according to the fitness value, the best flame position is saved for participation in the next iteration, so the process of updating the iteration does not result in the loss of the optimal solution from the previous iteration. As a new type of swarm intelligence optimization algorithm, the moth flame optimization algorithm can be described by the general triple model of the optimization algorithm, namely.

$$MFO = (I, P, T) \tag{8}$$

In the above description, we represent the function that needs to be initialized. When the MFO starts to run the solution, the position of the moth needs to be randomly generated in the global search space as the initial solution, and the fitness value corresponding to each moth is generated at the same time, as follows:

$$I = \Phi \to \{M, OM\} \tag{9}$$

P represents the position update function of the MFO algorithm. This function has an important role; it is the search centre and decision centre of moths and flames in the whole algorithm. When the moths fly paths and flight patterns in the feasible domain, the processing method to find the optimal position, etc., whether the P function is reasonably selected, is related to the performance of the entire algorithm. In the MFO algorithm, the P function mainly updates and iterates the position matrix M, which plays the role of connecting two iterations, which can be described as follows:

$$P: M \to M \tag{10}$$

T represents the threshold condition when the MFO algorithm satisfies the stopping operation. T is usually not an exact value. More often, the value of T is a range. Within this range, the MFO algorithm operation result reaches this range, and the condition is satisfied to stop the operation of the algorithm, and the T function returns True; if not satisfied, the return value is False. It is expressed as follows:

$$T: M \rightarrow \{true, false\} \tag{11}$$

By coupling the I function, the P function, and the T function, the structure of the resulting MFO algorithm is as follows:

```
M=I();
While T(M) is equal to false
    M=P(M);
End
```

As an initialization function, the I function must generate an initial solution, including the position of the moth and its corresponding fitness value. The initialization process is as follows:

I function:

```
for i=1:n
  for j=1:d
    M(i,j)=(ub(i)-lb(j))*rand()+lb(i);
  end
end
OM=FitnessFunction(M);
```

Among them, ub represents the upper bound vector in the search domain. Since each Moth and each flame need to be initialized to a value smaller than the upper bound vector at the beginning of the MFO algorithm, the dimension of ub is the same as that of the moth and the flame, which can be expressed as:

$$ub = [ub_1, ub_2, \dots, ub_n] \tag{12}$$

In the formula, ub_i represents the ith upper bound vector.

lb represents the lower bound vector in the search domain. Since each Moth and each flame need to be initialized to a value greater than the lower bound vector at the beginning of the MFO algorithm, the dimension of lb is the same as that of the moth and the flame, which can be expressed in the form of:

$$lb = [lb_1, lb_2, \dots lb_n] \tag{13}$$

lb_i represents the ith lower bound vector.

Moths use their special navigation mechanism for lateral positioning. In the MFO algorithm, a related mathematical model needs to be established to describe the flight state of each Moth and the position state of the flame. Each time the algorithm iterates, the moth needs to be in the P function. The mechanism is used to update the flight status and position, and after the update, the next iteration can be continued. The update mechanism is described as follows:

$$M_i = S(M_i, F_j) \tag{14}$$

Among them, S represents the helical function that the moth needs to construct when flying, and the S function should meet the following conditions:

1. It is necessary to provide the vector position of the S function's beginning point before the MFO method may execute the matching computation.
2. The optimal solution position discovered in each iteration of the MFO method should be saved by the S function prior to its conclusion.
3. The magnitude of the function is between the upper bound vector ub and the lower bound vector lb.

In MFO, the spiral function S of the moth is represented by the P function; its specific mathematical model can be expressed by the following equation:

$$S(M_i, F_j) = D_i e^{bt} \cos(2\pi t) + F_j \tag{15}$$

where Mi is the $i-th$ moth and F_j is the $j-th$ flame and D_i is the distance from the $i-th$ moth to the $j-th$ flame, $D_i = |F_j - M_i|$. b is the logarithmic spiral shape, constant t is the value range [-1,1].
Equation (15) clearly describes the flight path of each moth and its corresponding flame. After iteration, the position of the Moth at the next moment can also be calculated. In this formula, t represents the parameter for distance, the size of t is positively related to the distance between the flame and the moth, when the value of t is 1, the distance of the moth relative to the flame is far, when the value of t is -1, the moth is close to the flame. At the same time, this spiral function also shows that the flying domain of the moth is the whole worldwide space, including not only the space between the flame and the flame, but also the space occupied by the flame. This method can better ensure that the MFO algorithm can operate globally, while maintaining Space and local search capabilities. The logarithmic spiral path of the moth in flight and the exploration space around the flame, the vector position in the range of t value [-1, 1] is shown in Figure 8:

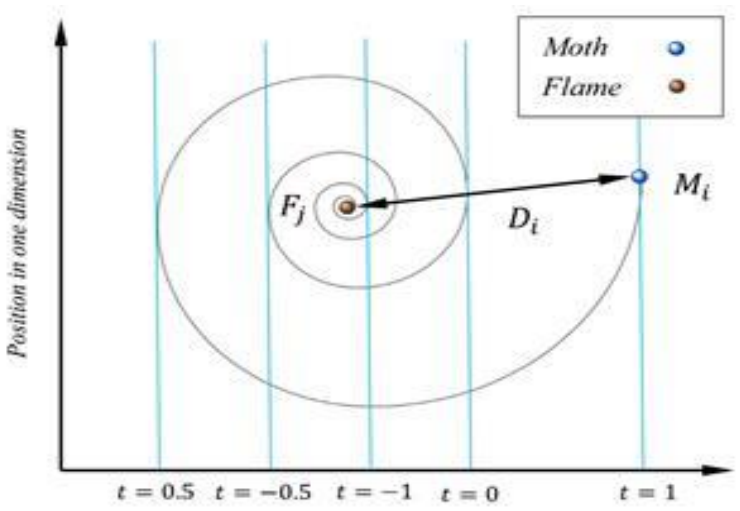


Figure 8: Logarithmic spiral and space around the flame

As shown in Fig. 8, when the MFO algorithm is iteratively updated, the position that the moth M_i may reach when it spirals around the flame, F_j, and the distance D_i of the moth from the flame is described. In Fig. 9, only a one-dimensional spiral path is shown, indicating that the problem model to be optimized has only one parameter that needs to be optimized. The blue line in Fig. 9 represents the flight path of the moth, and the green line represents the position path of the flame. The moth flies around the flame along the blue line. After the MFO algorithm updates the position of the moth, its fitness value will usually become better, as shown by the black dotted line. shown, then the position of the moth at this time will be recorded and used as the position of the flame in the next iteration, which is convenient to fully improve the local exploration performance of the algorithm. Similarly, the MFO can be deduced to two-dimensional or even high-dimensional aspects. The model has the following characteristics:
(1) If the parameter *t* is chosen randomly, the moth can converge somewhere near the flame;
(2) The size of the *t* value is positively correlated with the distance of the moth from the flame;
(3) The distance of the moth from the flame is negatively correlated with the frequency of the flame position update.

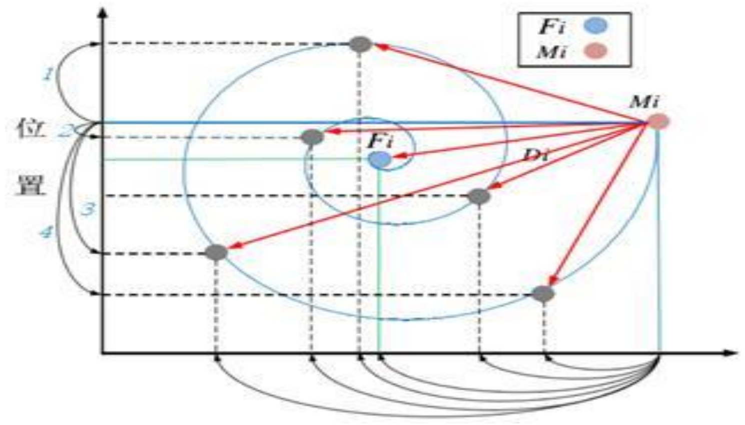


Figure 9: MFO iteratively updated

An adaptive mechanism to solve several flames during each iteration is mathematically defined as:

$$Num\ OfFlames = round\left(N - l * \frac{N-1}{T}\right) \quad (16)$$

where N is the maximum number of flames, l is the current iteration's number, and T is the maximum number of iterations.

One of the most important things in metaheuristic algorithms is the definition of the fitness function. In this research, considering that the goal is to reduce the execution time of the queries, the fitness function is defined as the sum of the execution time of each relationship.

$$fitness = \sum_{i=1}^{N} E_i \quad (17)$$

Where E_i represents the Execution time of the query i.

There are multiple control features that come with database security. so that by following the security protocols, it is possible to prevent the disclosure of the confidential information of the database. In the following, some control strategies used in database security are discussed.

Consolidation and continuous monitoring. Basic structural design in database security is proposed as a complementary method for database management systems or databases. This issue facilitates constant and stable updating in the system, keeping the database systems reliable. So that these systems work well with safety measures and precautions as part of security measures. This is achieved by keeping the system transparent and not having a point of penetration into it, and constantly monitoring the system's performance.

DBMS System Structure. An important part of providing successful performance in a database system is the configuration set used for the database management system. This configuration should cover administration activities and privileges, including access management activities. Hence, any mismanagement in the configuration settings can lead to great harm in protecting the security of the database and themselves. These actions are built in when the database security system is set up on an application, and they simplify the configuration process.

Robustness. These methods include several authentication features. That is, approaches that, together with the relevant authorization parameters, such as username and password, end up confirming the personnel and their access to the system. With the help of this approach, the security of the database remains untampered and away from danger, and the protection of the system is facilitated.

Acceptance criteria. One of the most effective ways to use database security features is to create a valuable constraint

3.2 Experiment

In many cases, there is a need to optimize the SELECT command while using it. Suppose the best execution plan is to be selected from among 100 different execution plan modes in the shortest possible time. Now, if the number of available modes for the Execution Plan and the execution of your order reaches 100,000, it will definitely be more difficult to find the optimal way to execute your orders in the same short time. With the above in mind, imagine that you have connected several tables A, B, C, D, and E using one of the JOIN types. Depending on how you did the JOIN, your query will generally have one of the following two structures:

1. Left-Deep Tree in which first table A is linked to table B, and then their result is linked to table C, and the JOIN combination of tables A, B, C is linked to D, and... Fig. 10

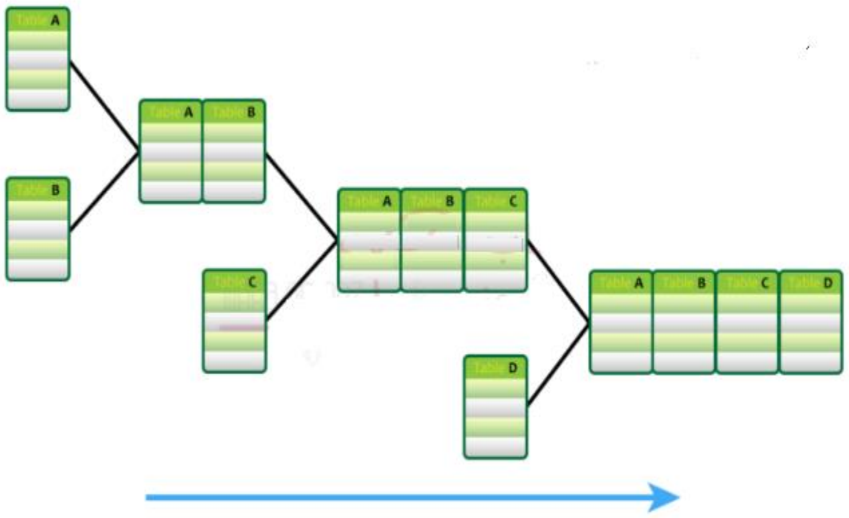


Figure 10: Left deep tree

2. Bushy Tree, in which table A is linked to B on one side, table D to C on the other side, and the result of these two JOINs is JOIN to each other and... Fig. 11

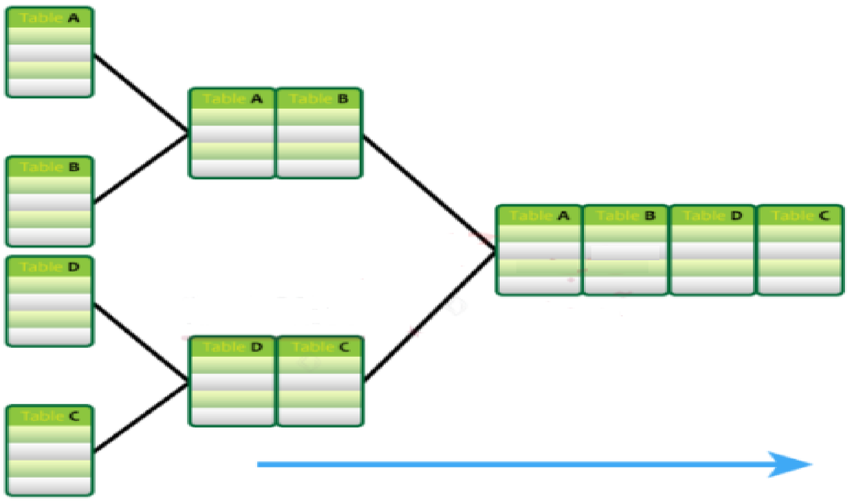


Figure 11: Bushy tree

In the case where the tables are JOIN with each other as a Left-Deep Tree, assuming there are n tables, the number of possible states will be $n!$ For example, if we have joined three tables with this method, the possible states will be 3, which is equal to 6 states. If n tables are joined with each other as a bushy tree, the number of states will be $\frac{(2n-2)!}{(n-1)!}$. This number is equal to

12 for 3 tables. Now, if the number of tables reaches 6 or 7, what numbers will your calculation display for the Left-Deep tree and Bushy tree states? You have arrived at very different numbers. Now, imagine that the query optimizer wants to find the best way to execute the query and optimize the SELECT statement among these many modes.
The higher the number of available states, the higher the probability of selecting an inappropriate Execution Plan for query execution by the query optimizer. Therefore, as much as possible, avoid creating a Bushy Tree structure during JOIN tables. In this way, by using the Left-Deep Tree method, you can allow the query optimizer to choose the optimal execution method for your query, which will lead to the optimization of the SELECT command in SQL Server.

The purpose of this research is to find the best execution time for query optimization using the MFO algorithm. Relations can be written in different ways, and by using the MFO algorithm, the best way with the shortest execution time is found. Each relation is processed and then evaluated using the fitness function. The fitness function in this research is the execution time of all relations to be evaluated. Our goal is to minimize this time.

In the MFO algorithm, several important parameters govern its performance and convergence behaviour. The population size N, which represents the number of moths in the search space, is typically set to 30. This parameter controls the diversity of the candidate solutions and influences the algorithm's ability to explore the solution space effectively. The number of iterations T, or generations, is set to 600 in most of the experimental setups. This determines how long the algorithm runs and how many opportunities it has to refine the solution. The dimensionality d of the problem, or the number of decision variables, is problem-specific. For example, in an engineering application such as the propeller design case mentioned in the paper, this value was set to 20. To guide the search process, MFO dynamically adjusts the number of flames (i.e., the best solutions) using a formula that decreases the flame count linearly with the number of iterations. This is given by equation 17.
The flowchart of the MFO algorithm is described in Fig. 12.

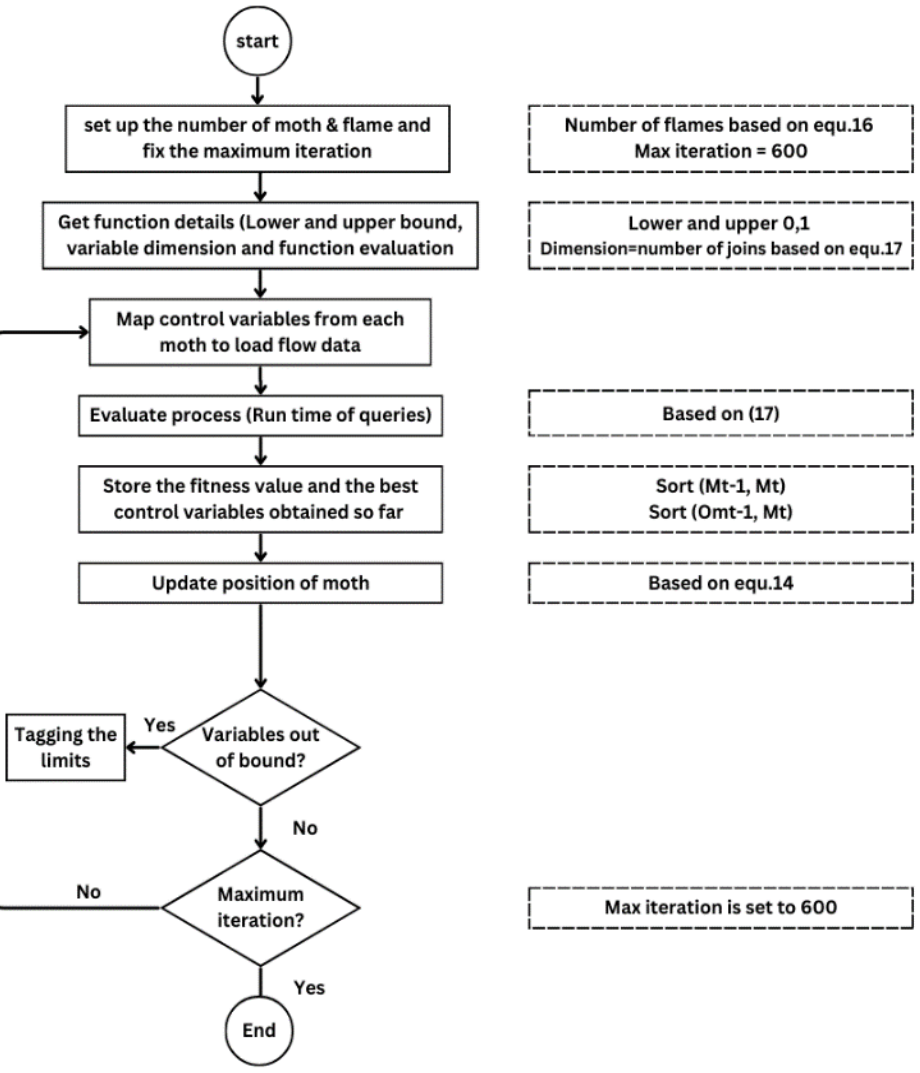

start
set up the number of moth & flame and fix the maximum iteration
Number of flames based on equ.16
Max iteration = 600
Get function details (Lower and upper bound, variable dimension and function evaluation
Lower and upper 0,1
Dimension=number of joins based on equ.17
Map control variables from each moth to load flow data
Evaluate process (Run time of queries)
Based on (17)
Store the fitness value and the best control variables obtained so far
Sort (Mt-1, Mt)
Sort (Omt-1, Mt)
Update position of moth
Based on equ.14
Variables out of bound?
Yes
Tagging the limits
No
Maximum iteration?
No
Max iteration is set to 600
Yes
End

Figure 12: Flowchart of the proposed method

The initial population generates an initial solution. Each solution is represented as below:

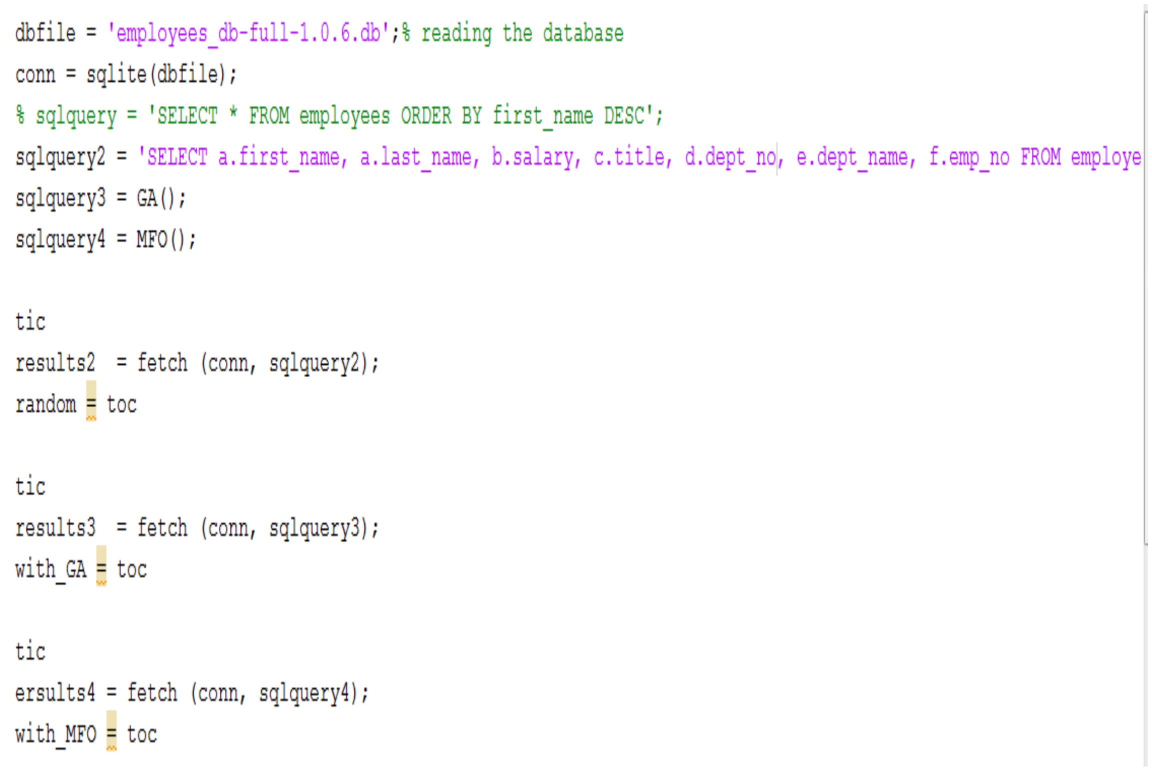
$E_1Q_i(t_it_{i+1})$	$E_2Q_i(t_{i+1}t_i)$	….

Which
E: is the query execution time per relation
Q: query relation
t: table in the database (example: table 1, table 2, etc.)
i: relation number (1, 2, 3..., etc.)
We test our proposed method on the employee database. First, generate 200 queries and then run them in 3 different scenarios. The first scenario is a random run, the second uses GA, and finally, MFO is employed. One sample of queries is:

```
SELECT a.first_name, a.last_name, b.salary, c.title, d.dept_no, e.dept_name, f.emp_no

FROM salaries b, employees a, titles c, dept_emp d, dept_manager f, departments e
WHERE …. AND
….  AND
…. AND
```

The simulation is implemented and run in MATLAB. The sample code's execution is as follows:



```
dbfile = 'employees_db-full-1.0.6.db';% reading the database
conn = sqlite(dbfile);
% sqlquery = 'SELECT * FROM employees ORDER BY first_name DESC';
sqlquery2 = 'SELECT a.first_name, a.last_name, b.salary, c.title, d.dept_no, e.dept_name, f.emp_no FROM employe
sqlquery3 = GA();
sqlquery4 = MFO();

tic
results2  = fetch (conn, sqlquery2);
random = toc

tic
results3  = fetch (conn, sqlquery3);
with_GA = toc

tic
ersults4 = fetch (conn, sqlquery4);
with_MFO = toc
```

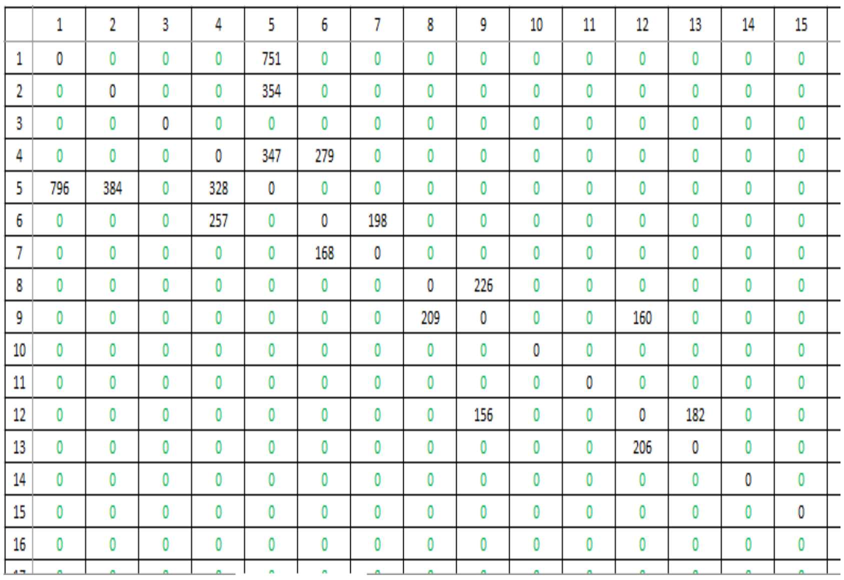
3.3 Results

Three different methods were tested to improve response time (random execution, GA, and MFO); their results were compared based on the total execution time. The random method showed the worst relative performance, taking up the entire execution time, making it a benchmark for comparing the other methods. The genetic algorithm GA achieved a significant improvement, taking only 77.5% of the time taken by the random method, indicating its effectiveness in significantly reducing execution time. The MFO algorithm, meanwhile, performed the best among the three methods, taking 73.7% of the time compared to the random method, demonstrating its greater effectiveness in reducing execution time and improving query execution than the GA.
The comparison of methods is described in Table 2:

Table 2: Comparison of the proposed method

Method	Random	GA	MFO
Response time (s)	3212	2492	2357
Response time (%)	100	77.5	73.3

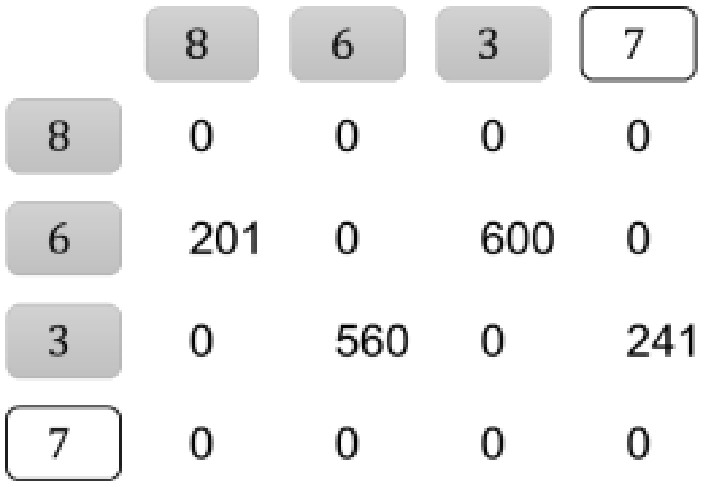
As we can see, our method has a better response time. To do this project, the dataset of a store "AdventureWorks1" has been used. The database is mounted on the SQL engine, and this database has 71 tables, 65 of which can be joined together. In total, six types of tables that contain the necessary information for this project have been extracted from this database, which includes the numbering table of tables, the table of relations, the table related to the number of related records in the tables, the table related to the number of related records in graphs, the table about time Connection between different tables and table related to connection time in graph mode. The matrix Fig. 13 has been produced using the relationship table and the joining time obtained from the experiment between the two tables. At the intersection of two tables, the joining time of those two tables is written. Of course, this value is zero when there are no connections between the two tables.



	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	0	0	0	0	751	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	354	0	0	0	0	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	347	279	0	0	0	0	0	0	0	0	0
5	796	384	0	328	0	0	0	0	0	0	0	0	0	0	0
6	0	0	0	257	0	0	198	0	0	0	0	0	0	0	0
7	0	0	0	0	0	168	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0	0	226	0	0	0	0	0	0
9	0	0	0	0	0	0	0	209	0	0	0	160	0	0	0
10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	0	0	0	0	0	0	0	0	156	0	0	0	182	0	0
13	0	0	0	0	0	0	0	0	0	0	0	206	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Figure 13 A part of the matrix of joining times:

According to the size of the joining time matrix between the tables, which has been obtained as a two-by-two timing between the tables, a part of it is shown in Fig. 13. To solve the query optimization problem, it is necessary to form an n-by-n matrix n is the number of tables that are related by records or keys, in which the cost of the related database tables is determined. The main diameter of this matrix is zero because it is meaningless to join a table with itself, and other values indicate the query time between that table and the opposite table. In the initial stages of the implementation, first, the time required to join the tables is extracted two by two, and then, based on these times, a matrix of dimensions $n * n$ is created, which is the number of tables that can be joined. Naturally, the main diameter of the matrix will be zero. Then, the input tables are received from the user in stages, and the most suitable combination for the tables is extracted with the MFO algorithm so that the joining time of the tables is the least. Joining time means the time it takes for two tables to join together. Here, there is no discussion about the execution time of the MFO algorithm or any other algorithm, and in fact, the discussion is about the time of joining the tables. According to the nature of the query problem, it is clear that this problem is one of the types of problems in which the answer changes in a permutative manner. For example, if we have only 3 tables and these tables are named with the numbers 2, 1, and 3, the possible answer for the query between these tables is n! (if all the tables are related two by two). we can get from one answer to another only by permuting. The cost function for the proposed algorithm is a combination of joining time and penalty. For example, if the relationships and joining times are like Figure 14 between the four tables 3, 6, 8, and 7, the value of the fitting function for each answer is calculated as follows.



	8	6	3	7
8	0	0	0	0
6	201	0	600	0
3	0	560	0	241
7	0	0	0	0

Figure 14: An example of a time matrix

If we have an answer like 6 -> 8 -> 3, because there is a connection between table numbers 6 and 8, then the value of joining time is added to the value of the fitting function, which is zero, then for table 8 and 3, which do not have links between them, a big penalty is considered, which in our implementation is the largest joining time in the table. We add as a penalty to the current value of the fitting function, which in the above example will be equal to

$$201 + 600 = 801$$

This increase in joining time is a penalty for the lack of relationship between two tables, which causes the cost to be very high in situations where there is no relationship between two consecutive tables, and practically, this solution is not one of the answers. To separate the possible answer from the impossible one, it is enough to check the existence of a consecutive relationship between the tables at the end of the program, and after obtaining the best answer, the cost of its calculation is $O(n)$.

The cost function for the genetic algorithm is obtained as follows:

$$Fitness = \sum_{Total\ number} Join\ time\ if\ exist\ else\ penaly \tag{18}$$

The results of the experiments are shown in Table 3:

Table 3: Comparison of MFO and GA

	Best fitness MFO	Best fitness GA	improvement
100 iterations	0.08	0.27	337%
300 iterations	0.18	0.49	272%

In order to check the security, a table similar to the timetable can be created and assigned a number to each combination of the table in terms of security, so that the joining of tables can also be checked in terms of security.

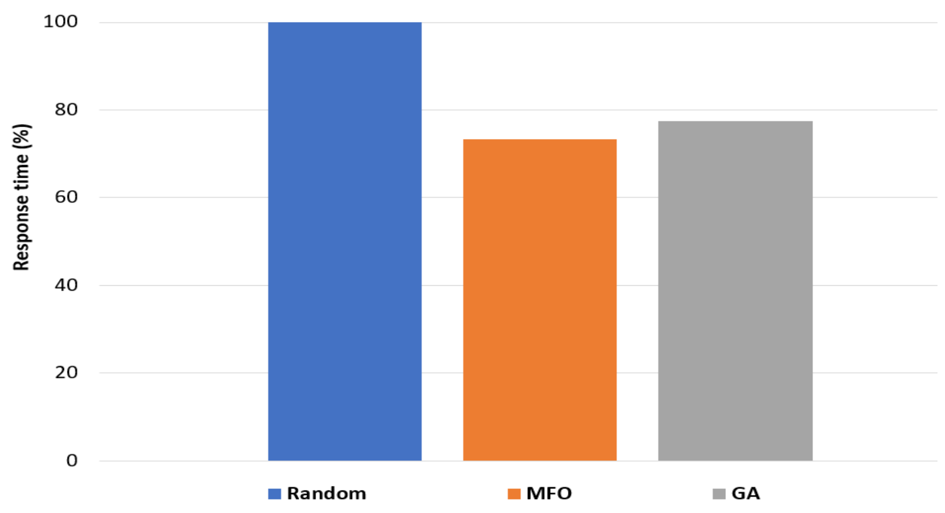


Figure 15: Comparison of response time

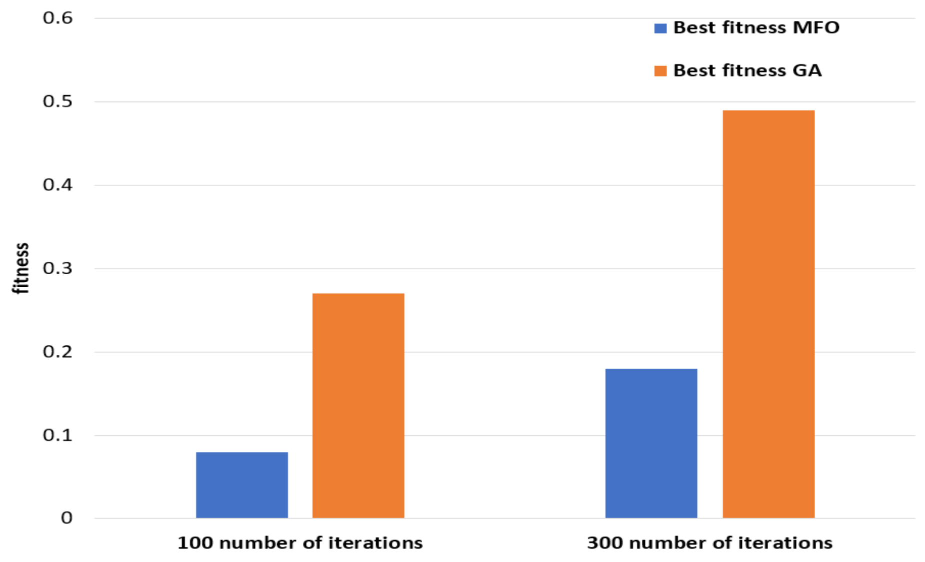


Figure 16: Best fitness:

The simulation results are displayed in Fig. 15, and out of all the investigated approaches, the Moth-Flame Optimization (MFO) algorithm offers the quickest query response time. Genetic Algorithm (GA) reduced response time by 22.5%, improving it to 2492 seconds from the random baseline's 3212 seconds (100%). With a reaction time of 2357 seconds, or just 73.3% of the initial time, the MFO algorithm further enhanced performance, showing a 26.7% improvement over the baseline and more effective performance than GA. In Fig. 16, the Moth-Flame Optimization method (MFO) consistently outperforms the Genetic method GA in terms of best fitness value when compared over varying iteration counts. MFO improved by 337% to reach a best fitness of 0.08 at 100 iterations, while GAs was 0.27. Likewise, in 300 cycles, GA achieved 0.49 and MFO reached 0.18, for a 272% improvement. These outcomes show that MFO outperforms GA in this optimization problem in terms of efficiency and convergence capabilities.

4. Conclusions

In this research, using the MFO algorithm, a method for better query performance for relational databases is proposed. A relational database consists of a series of two-dimensional

tables or logically related files that are used to store information in the form of a database. The word related is used to describe the relationship of two-dimensional tables or relational model files. A relational database has two important components. 1) The information itself is stored in a series of tables, files, or two-dimensional relationships (some use each of these terms interchangeably). 2) The logical structure of that information.
One of the important issues in a relational database is query execution time. Relations can be written in different ways. In this research, by using the MFO algorithm, the best way with the shortest execution time is found. Each relation is processed and then evaluated using the fitness function. Finally, the proposed method is compared with random execution and the genetic algorithm. Results showed that the proposed method has a better execution time than random and GA runs at 22.5% and 4.2%, respectively.

Acknowledgments First, I thank God Almighty for all his blessings, and I would like to thank my parents for their continuous support and also respect my professor Dr. Pedram Salehpour for all his unreserved support and effort. I sincerely appreciate my beloved wife and sons who had such happy days together and helped us achieve my goals. I would like to thank all my good brothers and wish them success. In the end, I dedicate the reward of my effort to my master and mawla, the owner of the era and the time of the awaited Imam Mahdi (may God hasten his honorable reappearance), and we last claim is that praise be to God, Lord of the worlds.

References

[1] P. K. Yadavr and S. R. Sam, "Query Optimization: Issues and Challenges in Mining of Distributed Data," *Big Data Analytics,* pp. 693-698, 2018.

[2] E. Azhir, N. J. Navimipour and M. Hosseinzadeh, "Deterministic and non-deterministic query optimization techniques in the cloud computing," *Concurrency and Computation: Practice and Experience,* pp. 1-17, 2019.

[3] W. Zheng, X. Jin, F. Deng, S. Mo, Y. Qu and Y. Yang, "Database query optimization based on parallel ant colony algorithm," *International Conference on Image, Vision and Computing (ICIVC),* pp. 653-656, 2018.

[4] "Employees (MySQL database)," Dataedo, August 2022. [Online]. Available: https://dataedo.com/samples/html/Employees_MySQL/doc/Employees_(MySQL_database)_7/home.html.

[5] J. Cannady, "Security Models for Object-Oriented Databases," in *Handbook of Data Management 1999 Edition*, Auerbach Publications, 2021, pp. 283-291.

[6] Rodrigues and B. J. Siqueira, "Security of a NoSQL database: authentication, authorization and transport layer security," 2019.

[7] Y. Du, Z. Cai and Z. Ding, "Query Optimization in Distributed Database Based on Improved Artificial Bee Colony Algorithm," *Applied Sciences,* 2024.

[8] I. Trumme, J. Wang, Z. Wei, D. Maram, S. Moseley, S. Jo, J. Antonakakis and A. Rayabhari, "Skinnerdb: Regret-bounded query evaluation via reinforcement learning," *ACM Transactions on Database Systems (TODS),* vol. 46, no. 3, pp. 1 - 45, 2021.

[9] R. Mancini, S. Karthik, B. Chandra, V. Mageirakos, V. Mageirakos and A. Ailamaki, "Efficient massively parallel join optimization for large queries," *Proceedings of the 2022 International Conference on Management of Data,* pp. 122 - 135, 2022 .

[10] M. B. Swidan, A. A. Alwan, Y. Gulzar and A. Z. Abualkishik, "An overview of query processing on crowdsourced databases," *arXiv preprint arXiv,* pp. 5-12, 2022.

[11] A. Bachhav, V. Kharat and M. Shelar, "An Efficient Query Optimizer with Materialized Intermediate Views in Distributed and Cloud Environment," *Tehnički glasnik,* vol. 15, no. 11 , pp. 105-111, 2021.

[12] A. Bachhav, V. Kharat and M. Shelar, "An Efficient Query Optimizer with Materialized Intermediate Views in Distribut-ed and Cloud Environment," *Tehnički glasnik ,* no. 1, pp. 105-111, 2021.

[13] V. Kumar and M. Biswas, "Multi-join Query Optimization Using Modified ACO with GA," *International Conference on Artificial Intelligence and Data Engineering,* p. 937–945, 2020.

[14] V. Kumar and M. Biswas, "Multi-join Query Optimization Using Modified ACO with GA," *In International Conference on Artificial Intelligence and Data Engineering,* pp. 937-945, 2020.

[15] T. Nadra, "Minimizing the makespan of diagnostic multi-query graphs in embedded real time systems," 2020.

[16] P. Patidar and P. Dashore, "Survey on Query Optimization using Heuristic and Ant Colony Algorithms," *International Journal for Scientific Research & Development,* vol. 8, no. 7, pp. 847-849, 2020.

[17] E. Azhir, N. J. Navimipour, M. Hosseinzadeh, A. Sharifi, M. Unal, M. Unal and A. Darwesh, "Join queries optimization in the distributed databases using a hybrid multi-objective algorithm; Arash Sharifi," *Cluster Computing,* pp. 1-16, 2022.

[18] Z. He, J. Yu, T. Gu and D. Yang, "Query execution time estimation in graph databases based on graph neural networks," *Journal of King Saud University-Computer and Information Sciences,* 2024.

[19] L. Woltmann, J. Thiessat, C. Hartmann, D. Habich and W. Lehner, "Fastgres: Making learned query optimizer hinting effective," *Proceedings of the VLDB Endowment,* pp. 3310-3322, 2023.

[20] M. Seyedali, "Moth-flame optimization algorithm: A novel nature-inspired heuristic para-digm," *Knowledge-based systems,* pp. 228-249, 2015.