Article

Dynamic Cost Ant Colony Algorithm to Optimize Query for Distributed Database Based on Quantum-Inspired Approach

Sayed A. Mohsin ^{1,*}, Ahmed Younes ² and Saad M. Darwish ¹

¹ Department of Information Technology, Institute of Graduate Studies and Research, Alexandria University, Alexandria 21526, Egypt; saad.darwish@alexu.edu.eg

² Department of Mathematics & Computer Science, Faculty of Science, Alexandria University, Alexandria 21568, Egypt; ayounes@alexu.edu.eg

* Correspondence: sayed.abdelmohsin@MTS-EGY.com

Abstract: A distributed database model can be effectively optimized through using query optimization. In such a model, the optimizer attempts to identify the most efficient join order, which minimizes the overall cost of the query plan. Successful query processing largely relies on the methodology implemented by the query optimizer. Many researches are concerned with the fact that query processing is considered an NP-hard problem especially when the query becomes bigger. Regarding large queries, it has been found that heuristic methods cannot cover all search spaces and may lead to falling in a local minimum. This paper examines how quantum-inspired ant colony algorithm, a hybrid strategy of probabilistic algorithms, can be devised to improve the cost of query joins in distributed databases. Quantum computing has the ability to diversify and expand, and thus covering large query search spaces. This enables the selection of the best trails, which speeds up convergence and helps avoid falling into a local optimum. With such a strategy, the algorithm aims to identify an optimal join order to reduce the total execution time. Experimental results show that the proposed quantum-inspired ant colony offers a faster convergence with better outcome when compared with the classic model.

Citation: Mohsin, S.A.; Younes, A.; Darwish, S.M. Dynamic Cost Ant Colony Algorithm to Optimize Query for Distributed Database Based on Quantum-Inspired Approach. *Symmetry* **2021**, *13*, 70. <https://doi.org/10.3390/sym13010070>

Received: 22 November 2020

Accepted: 26 December 2020

Published: 2 January 2021

Publisher's Note: MDPI stays neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Copyright: © 2021 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

Keywords: distributed database system; quantum computing; query optimization; ant colony optimization

1. Introduction

A distributed database is a group of interrelated entities that are physically distributed over the network to improve the computer performance, reliability, availability, and modularity of the distributed systems [1]. Optimizing query in databases, centralized or distributed, continues to be an important issue and main problem in commercial and academic fields for quite a long period of time [2]. Many approaches have been discussed earlier on query optimization that uses different technologies, but suffer from the problem of dimension and accuracy [3,4]. The importance for optimization arises from the flexibility provided by modern user interfaces to databases that help the users to easily specify queries effectively. The purpose of the optimizer, in this case, is to determine the best query execution plan (QEP) from many equivalent QEPs that will reduce the execution cost with less time complexity and utilize the minimum resources [5]. With large number of entities (large queries), the number of equivalent QEPs increase exponentially and the optimizer cannot explore all the query plans in such a huge search space. In this case, the selection of the best QEP, by applying a search strategy, is classified as an NP-hard optimization problem [6,7].

The search strategy typically falls into one of three categories—exhaustive search, heuristic-based, or randomized [2,4]. *Exhaustive Search* algorithms have exponential worst-case running time and exponential space complexity, which can lead to an algorithm requiring an infeasible amount of time to optimize large user queries [5]. Since exhaustive algorithms enumerate over the entire search space, the algorithm will always find the

optimal plan based upon the given cost model. The traditional dynamic programming (DP) enumeration algorithm is a popular exhaustive search algorithm, which has been used in a number of commercial database management systems [8].

Heuristic-based algorithms were proposed with the intention of addressing the exponential running time problem of exhaustive enumeration algorithms. Heuristic-based algorithms follow a particular heuristic or rule in order to guide the search into a subset of the entire search space [4]. Typically, these algorithms have polynomial worst-case running time and space complexity but the quality of the plans obtained can be orders of magnitude worse than the best possible plan. Iterative dynamic programming (IDP) is an example for a heuristic-based algorithm [9].

Randomized algorithms consider the search space as a set of points each of which correspond to a unique QEP [2]. A set of moves M is defined as a means to transform one point in space into another i.e., a move allows the algorithm to jump from a point in space to another. If a point p can be reached from a point q using a move $m \in M$, then we say that an edge exists between p and q . Randomized-based models and algorithms are applied with success to several optimization issues. Simulated annealing (SA), iterative improvement (II), and genetic algorithm (GA) have been suggested to optimize large scale recursive queries [10–12].

The ant colony optimization (ACO) algorithm is an apt and effective solution for optimizing query in a distributed database because of its features and characteristics, including its robustness, global optimization, parallelism obtained from the ability to act concurrently and independently, and capability to integrate with other methods [3]. To utilize the ACO algorithm for addressing the issue associated with query optimization, the issue will be described as a graph. In this case, the graph symbolized as $G = (N, E)$. The parameters N and E formulate the number of entities (tables) and the relations (edges) between these entities. The edges that link the nodes together on the graph G represent the join relations among entities. In such a case, the purpose of the query optimizer would be to seek out the best Hamiltonian path for G .

The most significant advantage of quantum computing is its ability to potentially resolve specific issues faster and more efficiently than classical computing, such as problems with a high computational cost [13]. Quantum-inspired exploration procedures employ the ability of parallel processing by adopting the superposition principle to overcome the limitation of the classical mechanism and to fulfil a higher performance [14]. It is noteworthy that superposition is the aptitude of a quantum system to be in numerous positions (states) simultaneously while waiting for measuring. It is often customary to employ such ability of carrying out parallel processing to solve issues that require the exploration of huge solutions spaces.

This paper is a substantial extension of our conference paper [15]. Compared with this small version, further details of the suggested method are presented, and more extensive performance evaluation is conducted. In addition, this paper gives a more comprehensive literature review to introduce the background of the offered method and make the paper more self-contained. Therefore, this version of the paper provides a more comprehensive and systematic report of the previous work. This paper investigates how the quantum-inspired ant colony optimization (QIACO) algorithm can be used to overcome the problem of join query optimization in distributed databases when it comes to search spaces where entities (tables) are not replicated and depends on total time for explaining the cost model. Because processing of the queries is considered an NP-hard problem, current traditional approaches, especially when the size of the database increases, suffer from large computational cost, non-convergence to a global optimum and premature convergence. To solve the problems in traditional methods, quantum-inspired ant colony (QIACO) paradigm is used in try to reach the optimum query optimization. Here, quantum-inspired is employed to change the seeking procedure used by the classical ant colony algorithm to move from one node to another. Instead of using the probabilistic mechanism while building the ant solution, our algorithm will use the quantum partial negation gate, controlled by

pheromone values, to control the ant movement. Our model was tested using a synthetic data set and modified TPC-H benchmark queries. This paradigm is able to improve the slow convergence speed and avoid falling into the local optimum. The result shows that our model behaves better than the classical model, especially for the queries that contained many entities.

The rest of this paper is divided into four sections. Section 2 formulates the ACO algorithm and describe the basics of quantum computing used in the proposed algorithm. Section 3 reviews the previous related works. Section 4 presents the four algorithms for query optimization. Section 5 describes the experimental result that evaluates the algorithm. The paper is concluded with Future Works and Conclusions sections.

2. Preliminaries

2.1. Ant Colony Optimization

Ant colony optimization (ACO) is one of many various approaches of swarm intelligence, a field wherein specialists consider the collective behavioral of insects as an inspiration to mimic approaches. At first, ACO was utilized to find a solution for the traveling salesman and quadratic assignment problems [16]. Ants solve their issues by traversing the graph that represents the problem and leaving behind pheromone to lead the remaining ants. Pheromone trails are used to give the ants a chance to cooperate and benefit from the experiences of other ants by providing positive feedback. On the contrary, negative feedback, which is represented by pheromone evaporation, will need to avoid doldrums. The first ACO algorithm, called the ant system, is utilized to solve TSP [16]. Many different ACO techniques and algorithms have been created and suggested since then. One of the main features of ACO is that, at each iteration, the pheromone values raised by every ant are modified by the ants at the same site that provide a solution. The pheromone τ_{ij} , attached to the edge joining nodes i and j , is updated as follows [16]:

$$\tau_{ij} = (1 - \rho) \cdot \tau_{ij} + \sum_{k=1}^m \Delta \tau_{ij}^k, \quad (1)$$

where ρ is the evaporation rate for pheromone quantity $\Delta \tau_{ij}^k$ placed on edge (i, j) by ant k from m ants:

$$\Delta \tau_{ij}^k = \begin{cases} \frac{Q}{L_k} & \text{if ant } k \text{ used edge } (i, j) \text{ in its tour,} \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

where Q is a constant and L_k is the length of the round created by ant k . When building the solution, the ants choose the next node that should be visited according to a randomized mechanism. When ant k is in node i and has so far constructed the partial solution s^p , the probability of going to node j is given by:

$$p_{ij}^k = \begin{cases} \frac{\tau_{ij}^\alpha \cdot \eta_{ij}^\beta}{\sum_{c_{ij} \in N(s^p)} \tau_{ij}^\alpha \cdot \eta_{ij}^\beta} & \text{if } c_{ij} \in N(s^p), \\ 0 & \text{otherwise} \end{cases} \quad (3)$$

where $N(s^p)$ is the set of feasible nodes. The relative significance of the pheromone in contrast to the heuristic information η_{ij} , controlled by the parameters α and β and obtained using distance d_{ij} by:

$$\eta_{ij} = \frac{1}{d_{ij}}, \quad (4)$$

where d_{ij} formulates a distance or cost from nodes i to the connected node j .

2.2. Quantum-Inspired Evolutionary Algorithms

Quantum-inspired evolutionary algorithms (QIEA) are population based meta-heuristics that draw inspiration from quantum mechanical principles to improve the efficiency

and the search for evolutionary optimization algorithms. The possibility feature of parallelism given by quantum computing and simultaneous evaluation of all possible represented states, drive to the improvement of models which integrate some feature of quantum computing with evolutionary computation [17,18]. These models are prepared to execute on classical computers, not on quantum computers, and categorized as “quantum inspired”. One of the early attempts was made by Han and Kim [18] who prepared a general model of QIEA. Rather than binary, numeric, and symbolic representations that exist in a classical computer, QIEA uses Q-bit to represent the smallest unit of data. A qubit may be in the “1” state, in the “0” state, or in any superposition between “1” and “0”. The qubit’s state is described mathematically as [19]:

$$|\psi\rangle = a|0\rangle + b|1\rangle, \quad (5)$$

where a and b are complex numbers that give the probability amplitudes for each corresponding state [19]. The probability that the qubit will exist in the state $|0\rangle$ obtained by $|a|^2$ and the probability that the qubit will exist in the state $|1\rangle$ obtained by $|b|^2$ [19]. Normalization of the state to unity insure:

$$|a|^2 + |b|^2 = 1. \quad (6)$$

The qubit’s state can be modified by an operation called a quantum gate. A quantum gate is a reversible gate and can be represented as a unitary operator U acting on the qubit basis states satisfying $U^\dagger U = U U^\dagger$, where $U^\dagger$ is the complex conjugate transpose of U [19]. There are many quantum gates, like the NOT gate, rotation gate, Hadamard gate, etc. [20]. Figure 1 shows the overall structure of QIEA where $Q(t)$ is the Q-bit representation for the individuals in the search population at time t , $P(t)$ is the solution acquired by measuring the states of $Q(t)$, and $B(t)$ is the best solution at time t . More details regarding the complete steps of QIEA can be found in [18].

Figure 1. Overall Structure of Quantum-inspired evolutionary algorithms [18].

2.3. Problem Definition

A distribution allocation scheme is used in distributed databases to dispense data, which may be propagated at different locations. The objective of the query optimizer, in this case, is to provide an execution plan (from different equivalent plans) that helps in reducing

the cost of query execution that is based on either response time or total time, to a minimal. The solution space for query that contains many entities and, in the same time, many database locations will grow exponentially. Searching for the best query execution plan, in this case, becomes computationally difficult and classified as an NP-hard optimization problem. Here, finding the best execution plan depends on the search strategy that will be used to explore the solution space.

The query search space can be represented as a graph $G = (N, E)$ with a set of vertices N , and represents the set of entities in the query, and set of edges E , represents the join between entities, such that each edge $e \in E$ is assigned a cost C_e . Let H be the set of all Hamiltonian cycles, a cycle that visits each vertex exactly once, in G . The optimizer problem is to find the path $h \in H$ in G such that the sum of costs C_e is minimized. Given a set of entities n enumerated as $0, 1, 2, \dots, n-1$ to be joined with the join cost between entity i and j given by C_{ij} . We introduce a decision variable y for each (i, j) such that:

$$y_{ij} = \begin{cases} 1 & \text{if entity } j \text{ joined to entity } i, \\ 0 & \text{otherwise} \end{cases}.$$

The objective function in this case is:

$$\min \sum_i^n \sum_j^n C_{ij} y_{ij}.$$

Here, this objective function will be guided by two parameters, the number of entities n and the join cost C which is affected by the number of database locations because the entities will be transferred between different locations.

So, as to establish a solution for the optimization of the query problem, three important parts should be studied: search space, search strategy, and cost model. Search space refers to the generation of sets of alternative and equivalent QEPs that differ in the execution order of the operators. Search strategy refers to the algorithms applied to explore the search space and determine the best QEP based on join selectivity and join sites so as to reduce the cost of query optimization. Cost model refers to the model used to predict the cost for every QEP. In this paper, a quantum-inspired ant colony algorithm will be used as a search strategy, depending on a dynamic cost technique [15], to identify the best QEP. Here, the ant colony algorithm will be used to identify the routing path and the quantum computing will be used to enrich the search process for identifying the join entity order.

3. Related Work

The key component in the optimizer of query is the employed search methodology. There is extensive and rich literature describing the process of optimization and studying the utilized search technique, indicating its significance. In a distributed database system (DDBS), failures in the midst of a transaction processing (such as failure of a site where a sub-transaction is being processed) directly affect the query optimizer and may lead to an inconsistent database query result [21]. As such, a recovery subsystem is an essential component of a DDBS. To ensure correctness, recovery mechanisms must be in place to ensure transaction atomicity and durability even in the midst of failures.

There are mainly three main search approaches utilized to determine the best QEP, exhaustive, heuristic-based, and randomized strategies. Dynamic programming (DP) is one of the most recognized exhaustive search strategies, and it is used as a search technique in most commercial databases. The basic algorithm of DP used for optimizing query is introduced in [9]. The optimizer process depends on the drill down approach by frequently creating composite search plans using divided smaller search sub-planes till the overall plan is complete. Here, by pruning process, the high-cost search plan is neglected early in the case an alternative equivalent search plan exists with a low cost. Although this technique gives a better performance than a randomized strategy when it comes to queries

with a small number of entities, randomized strategies are a much better fit for queries with a large number of entities.

Iterative improvement (II) is one of the most known techniques categorized as randomized algorithms [22]. II initially chose an irregular starting point. Then, the solution improved by repeated acceptance of random subsidence moves until reaching the local minimum. This procedure is repeated until a predetermined halting condition is met. By then, the algorithm reaches the point that has a minimal cost. One of the primary disadvantages of II is that, sometimes, the conclusive outcome is unsuitable, in any event, when an enormous number of beginning stages are utilized. At the point when the set of solutions include an enormous number of significant cost nearby minima, II gets handily caught in one of them.

Genetic algorithm (GA), another randomized algorithm, is introduced in [23]. Here, GA is presented as a solution technique to optimize query issues, and it is tested in comparison with other methodologies. However, this methodology does not think about the modified crossover and mutation process. In [24], the author consolidated GA and the min-max ant system to create an optimization methodology for a query in order to enhance the efficiency of the query. The superiority of parallelism exceptionally appeared by corresponding GA and the max-min ant system in the event of an enormous number of relations. In comparison with different algorithms, this execution plan has less inquiry time, and furthermore, time of query execution is diminished in the optimal plan created. This has shortcomings where the computation time and the cost of computation are expanded due to parallel processing of two algorithms.

As one of the stochastic-based algorithms, the ACO algorithm was used in this investigation as a search methodology for optimizing queries in both centralized and distributed database environments [25]. In [26], the author proposed a multi-colony ant algorithm to improve join inquiries in distributed systems in which tables can be copied, however, they cannot be partitioned or fragmented. In this planned algorithm, 4 sorts of ants coordinate to create an execution plan. In this way, every one of the emphasis has four subterranean insect settlements. So, as to locate the optimal plan, every ant performs dynamic decision-making. Two kinds of cost models fixated on total time as well as response time are used for the assessment of the quality of the generated plan. In this algorithm, despite the fact that the total time is diminished and the convergence speed is increased, it is getting a worse performance for a small query and can fall in a local optimum.

Quantum-inspired evolutionary algorithms are one of the key areas of research linking quantum computing with evolutionary algorithms. The theoretical applications of quantum-inspired evolutionary algorithms in various fields are presented for the first time in [27]. In [28], the author applies QIEA for locating minimum costs of the assignment in the quadratic assignment problem (mathematical model for the assignment of a collection of economic activities to a collection of locations). The main contribution behind this paper is to present how the algorithm is tailored to the problem, containing crossover and mutation operators, furthermore setting the overall framework for the utilization of quantum ideas in varied applications. In addition, QIEA is applied along with genetic programming to improve prediction accuracy of toxicity degree of chemical compounds [29]. In this work, the accuracy of the linear equation that was used to calculate the degree of toxicity increased as a result of using genetic programming. Moreover, quantum computing helps in improving the selection of the best of run individuals and handling stinginess pressure to decrease the complexity of solutions. Additionally, in [30], the author creates a new technique to find optimal threshold values at different levels of thresholding for color images and uses a minimum cross entropy-based thresholding method as an objective function. In this technique, the results are described in terms of the best threshold value, fitness measure, and the computational time for each technique at various levels. Here, the convergence curves prove that the use of quantum-inspired concepts along with the ACO technique outperforms the results obtained by the classical ACO technique.

Our proposed algorithm represents an extension to the work submitted in [15]. This work employs the total query time calculated for the distributed query optimization model that is utilized for entities which non-replicated as the model's cost. The processing and the cost of communication for the query plan are calculated dynamically and depend on the path used by ants and the entities site's location. No fixed cost exists over the edge of the problem graph, but it is calculated dynamically as an intermediate outcome while applying the query's joins. Quantum gate will be used in the suggested model as a replacement of ACO stochastic search to enhance the total cost and accelerate the search convergence.

4. The Proposed Technique

Our developed methodology for optimizing query in the environment of a distributed database will be presented based on the implementation of search space, the method used to obtain the cost, and the search methodology utilized to find the best QEP. The search space implementation and cost calculation method used the same concept as [15] but the search methodology will employ the concept of quantum computing to get the best QEP. Figure 2 shows the main components of the suggested optimization model and the way these components are linked together.

Figure 2. Quantum-inspired ant colony model for distributed database query optimization.

In this figure, the SQL statement was analyzed to identify the involved entities (tables). Then, the database statistics are associated with the identified table extracted from the database category tables. These statistics include the field's length and type, the entity tuple length and number of tuples in every entity, and number of pages required to store the entity data. In addition, the entity site locations and relations between entities are also obtained from the database category tables. The next step is using the information obtained from the database category tables to create the search space. Finally, our search method will be applied to the search space to obtain the best join order. The major component of the model is described in the following sections.

4.1. Cost Model

As it exists in [15], the cost calculation method is obtained based on total time (the total sum of all components cost) dependent on the same calculation method as in [26]. Here, the total cost is obtained as a sum of I/O cost, calculated for all join process, and data transfer (communication) cost that is calculated when transferring entities between sites is necessary:

$$Total\ Cost = IO_{join} + COM_{R_i}, \quad (7)$$

where IO_{join} is the cost of the join process and COM_{R_i} represents the cost of transferring entity R_i among the location of sites. The IO_{join} cost is computed as:

$$IO_{join} = (P_{join} + P_{write}) * IO(S_k), \quad (8)$$

where $IO(S_k)$ is the I/O time for the disk in location S_k , P_{write} is the page count that is required to save the join outcome, and P_{join} is the page count accessed to perform the join process between R_i and R_j . P_{join} is computed as:

$$P_{join} = P_{R_i} * P_{R_j}, \quad (9)$$

where P_{R_i} is the page count in entities R_i and R_j . P_{write} is computed as:

$$P_{writ} = \frac{card(R_i\ join\ R_j) * len(R_i\ join\ R_j)}{ps}, \quad (10)$$

where $card(R_i)$ is the tuple count in R_i , $len(R_i)$ is the average length for tuple in R_i , and ps is the size of page. The cost of required to transfer relation R_i from location S_k to location S_p is computed by:

$$COM_{R_i} = card(R_i) * len(R_i) * COM(S_k, S_p), \quad (11)$$

where $COM(S_k, S_p)$ is the time needed to move one byte from location S_k to location S_p .

4.2. Build Search Space

The catalog in the database is used to primarily store the schema, which contains information regarding the tables, indices, and views [1]. The information about tables includes the table name, the field names, the field length, the field types, and the integrity constraints between tables. Various statistics are also stored in the catalog such as the number of distinct keys in a given attribute and the cardinality of a particular relation. In addition, the catalog includes information about the resident site for tables, the number of sites (locations) in the system, along with their identifiers and the replication status. This information extracted from the catalog tables will be employed to build the search space. In our algorithm, the search space will be implemented as a graph $G = (N, E)$ where N represents the collection of vertices (nodes) and E represents the collection of edges (arcs) [4]. Every vertex in the graph denotes an entity (table) in the query specification. Two graph vertices are linked together by an edge if the corresponding tables are joined together in the query. Every vertex in the graph is represented as a class data structure and has a set of attributes like number of tuples, tuple length, keys, and site location. Figure 3 show

a graph representation for a search space that contains a set of entities and the relations among them.

Figure 3. Graph representation for the search space.

4.3. Search Strategy

The search methodology for our proposed model depends on QIACO metaheuristic. The cost of every journey for each ant to locate the minimal spanning graph, which represents the query search space, will be dynamically calculated while building that graph as in [15]. The dynamic calculation of query cost depends on an additional virtual vertex that will be added to the graph to carry the intermediate join process outcome. The cost, in this case, is calculated between the virtual node and the next chosen entity in the join order. In the proposed algorithm, each entity is reformed as one qubit representation as in the form of Equation (5) and quantum partial negation gate will be used to identify the next entity in the join order. The flow of QIACO is described in the following steps:

Step 1. Initialization. In this phase, all the parameters used in the model are initialized, depending on work in [31], experimentally. Minor changes were done to the constant values α , β , and ρ . The value of α used as 3 instead of 1, the value of β used as 2 instead of 5, and the value of ρ used as 0.02 instead of 0.1. These changes in the values increased the dependence of our work on the cost, instead of pheromone, and give a better result, while identifying the next entity in the join order. The number of ants will be determined, and pheromone trails will be initialized. All pheromone values will be initialized by an arbitrary small value equal to $\frac{1}{\sqrt{\text{No. Of Entities}}}$. The query graph that links the tables (entities) is generated such that every table is connected to all other tables. In this phase, each entity will be associated with a qubit and all entities' qubit probability amplitudes will be initialed as $a_i = b_i = \frac{1}{\sqrt{2}}$ which satisfy Equations (5) and (6):

$$|\psi_i\rangle = \frac{1}{\sqrt{2}}|0\rangle + \frac{1}{\sqrt{2}}|1\rangle.$$

All the parameters of the algorithm are initialized in this phase according to Table 1.

Table 1. Parameters for the Ant Colony Optimization algorithm [31].

Parameter	Description	Value
α	Pheromone influence factor	3
β	The influence of heuristic	2
ρ	Pheromone evaporation rate	0.02
Q	Constant used while pheromone update	2
Max-Iteration	Varies with the number of entities	-
Ant-Numbers	Varies with the number of entities	-

Step 2. For each ant, select random entity and uses it as the start-up point for the journey of the ant. This entity transfers to the virtual vertex and waits to choose the following entity to implement the join process.

Step 3. Use a partial negation quantum gate to choose the next entity in join sequence. The selection will be done by applying the negation gate, as operator, on all entities' qubit that have a connection path with the current entity. This process will be applied many times according to the amount of pheromone that is raised on the path between the current entity and the connected entities. Then, the entity with best probability will be selected as the next entity. Let X be the Pauli-X gate, which is the quantum equivalent to the NOT gate and represented as:

$$X = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}. \quad (12)$$

The c^{th} partial negation for operator V is the c^{th} root of the X gate and can be calculated using diagonalization as follows:

$$V = \sqrt[c]{X} = \frac{1}{2} \begin{bmatrix} 1+t & 1-t \\ 1-t & 1+t \end{bmatrix}, \quad (13)$$

where $t = \sqrt[c]{-1}$, and:

$$V^\dagger = \frac{1}{2} \begin{bmatrix} 1-t & 1+t \\ 1+t & 1-t \end{bmatrix}, \quad (14)$$

$$V.V = V^\dagger.V^\dagger = X \text{ and } V.V^\dagger = V^\dagger.V = I. \quad (15)$$

This gate is represented in Figure 4 [32].

Figure 4. Partial negation gate.

Applying the operator V on a qubit d times is equivalent to:

$$V^d = \frac{1}{2} \begin{bmatrix} 1+t^d & 1-t^d \\ 1-t^d & 1+t^d \end{bmatrix}, \quad (16)$$

such that if $d = c$, then $V^d = X$. When $d = c = 2$, this will give $t^2 = -1$ and so:

$$V^2 = V.V = \frac{1}{2} \begin{bmatrix} 1+t^2 & 1-t^2 \\ 1-t^2 & 1+t^2 \end{bmatrix} \cdot \frac{1}{2} \begin{bmatrix} 1+t^2 & 1-t^2 \\ 1-t^2 & 1+t^2 \end{bmatrix},$$

$$V^2 = \frac{1}{4} \begin{bmatrix} 0 & 4 \\ 4 & 0 \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} = X.$$

In our model, the V gate is used as an operator and is conditionally applied for c times on every entity' qubit. The number of times, c , the V gate will be applied on entity' qubit is based on pheromone value and join cost. Here, c will be defined using Equation (3) as:

$$c = \tau_{ij}^\alpha \cdot \eta_{ij}^\beta, \quad (17)$$

where τ_{ij} is the pheromone trail on the arc that connects entity i with entity j and, η_{ij} is the heuristic desirability and computed as the inverse of the intermediate join cost between

entities i and j . Here, the join total cost is identified by Equations (4) and (7). The amplitudes for each entity will be updated at time $t + 1$ as:

$$|\psi_i^{t+1}\rangle = \begin{bmatrix} a^{t+1} \\ b^{t+1} \end{bmatrix} = V \begin{bmatrix} a^t \\ b^t \end{bmatrix}. \quad (18)$$

After that, the suggested system uses the tensor product $\otimes$, a way of putting vector spaces together to obtain a larger vector space, so that for n number of entities:

$$|W\rangle = |\psi_1\rangle \otimes |\psi_1\rangle \otimes \dots \otimes |\psi_n\rangle. \quad (19)$$

The vector obtained in Equation (19) will normalize to the number of elements similar to the number of entities in the model. In this case, each element in the normalized vector will represent the probability of the corresponding entity to be the next entity in the query join order.

Step 4. Instead of selecting the entity with better probability in the normalized vector as the next entity in the join order, the roulette wheel method is used to give the entities with small probability a chance to share in building the query join order.

Step 5. After choose the next entity, the join cost is determined and the outcome of the join process is transferred to the virtual vertex and afterward utilized as the beginning up entity for the following ant's cycle. Equations (7)–(11) are used to determine the join cost.

Step 6. Repeat steps from 3 to 5 till all entities are handled and then the journey cost for the ant is calculated.

Step 7. Select the best journey cost from all the journeys for ants.

Step 8. Pheromone update. At the point when all the ants build their solutions and, in each cycle, the update of pheromone is performed. Every pheromone amount is reduced, to simulate the pheromone evaporation, and raised, to simulate the ant's pheromones deposit on the trail. The modification of pheromone on all graphs' arcs is done using Equation (1). The modifications of the pheromone are also dependent on L_k in Equation (2) that symbolize the total cost of the better tour created by ant k .

Step 9. Repeat steps from 2 to 8 until maximum iterations and when the cycles complete, the best trail for all ants is chosen.

The pseudo-code for the suggested model is represented in Algorithm 1, the partial negation gate is represented in Algorithm 2 and the selection of next entity represented in Algorithm 3.

Algorithm 1. QIACO

- (1) Initialize values for $\alpha, \beta, \rho, Q, \text{Max-Iteration}, \text{Ant-Numbers}$ // (step 1)
 - (2) Initialize pheromone by $\tau_{ij} = \frac{1}{\sqrt{\text{No.Of Entities}}}$. // (step1)
 - (3) Initialize all Qubit by $|\psi_i\rangle = [1/\sqrt{2}; 1/\sqrt{2}]$ // (step 1)
 - (4) $t = 0$
 - (5) **Loop** (cycle < Max-Iteration)
 - (6) $t = t + 1$
 - (7) StartEntity = random beginning entity // (step2)
 - (8) **Loop** (ant < Ant-Numbers)
 - partialStep = NegationGate(ant, StartEntity) // (step 3)
 - NextEntity = ChooseNextEntity (partialStep) // (step 4)
 - CalculateCost(StartEntity, NextEntity) // (step 5)
 - StartEntity = NextEntity
 - (9) **EndLoop** // (step 6)
 - (10) Identify best trail for ants. // (step 7)
 - (11) Modify the pheromones. // (step 8)
 - (12) **EndLoop** // (step 9)
 - (13) Identifybest trail for the solution
-

Convergence of the Model. In [33], the author proves that the convergence of ACO depends on the pheromone value τ_{ij} from Equation (1) and heuristic information η_{ij} , from Equation (4). The suggested model still depends on Equation (1) for updating the pheromone. Additionally, Equation (4) represents the inverse of cost which is used in the suggested model to find the best route obtained by ants. So, the convergence of the suggested model is guaranteed.

Complexity analysis. As in [34], the computational complexity of the most classical ant colony algorithms depends on the number of nodes n , the number of ants m , and the number of iterations T (the colony lifetime). Considering Equation (3) in more detail, we can notice that the computational complexity of the algorithm also depends on the parameters α and β . Here, the computational complexity was explained as:

$$O\left(T m n^2 (\log_2 \alpha + \log_2 \beta)\right).$$

In our proposed QACO algorithm, a single quantum superposed state is prepared to encode each node n in the search space, thus exploration of all nodes in a single iteration is $O(n)$ time. The selection of the next entity in the join order depends on the negation gate which is represented mathematically by vectors product with $O(n)$ time. Here, the vector product will be applied to all n nodes, so the selection process running time is $O(n \times n) = O(n^2)$. So, the overall computational time for the quantum part in our model is $O(n) + O(n^2)$ which can be reduced to $O(n^2)$. Hence, the computational complexity for our QACO algorithm is:

$$O\left(T m n^2 (\log_2 \alpha + \log_2 \beta)\right) + O\left(n^2\right).$$

QACO has a computational complexity greater than the classical ant colony, but gives better results as will be explained in the experimental results section.

Algorithm 2. Partial Negation Gate

Function NegationGate(antX, currentEntity)

(1) $QGate = [0 \ 1; 1 \ 0]$

(2) **For** (entity = 1 to Number-of-Entities)

If entity is visited **then**

 partialStep[entity] = 0.1×10^{-10} ; *very small value*
 continue

End If

 cost = CalcCost(currentEntity, entity)

 ph = GetPheromone(currentEntity, entity)

 tau = $ph^\alpha \times cost^\beta$

$|\psi_i^{t+1}\rangle = QGate^{tau} * |\psi_i^t\rangle$

 partialStep[entity] = $|\psi_i^{t+1}\rangle$

(3) **End For**

(4) **Return** partialStep[entity]

4.4. Limitation

- This algorithm uses the total query time calculated for distributed query optimization as the model's cost.
- The algorithm applies for non-replicated entities.

Algorithm 3. Choose Next Entity

Function ChooseNextEntity(partialStep)

 NRV = NormalizeVictore(partialStep)

 SelectedEntity = RouletteWheel(NRV)

Return SelectedEntity

5. Experimental Results

In this section, many achieved experiments will be demonstrated to study the efficiency for our proposed model. Furthermore, to evaluate the performance, the accuracy of our proposed model will be compared with the traditional optimization techniques. The classical distributed database query optimization in [15] is built by a modified version from the C# code that implements an ant colony optimization (ACO) algorithm to solve the traveling salesman problem (TSP) which was created by Microsoft MSDN Magazine [35]. The quantum version is implemented using C# along with MATLAB.

To test the proposed model, two types of datasets are used, synthetic and benchmark. The first group of tests, experiments 1, 2, and 3, performed on synthetic dataset, which is randomly generated by a problem generator to simulate the existence of the join for different number of entities. The problem generator has two parts; the database generator and the query generator. The first part generates a synthetic database depending on the number of relations. During this part, the cardinality, the length of tuples, and the join attribute of the relations are defined. In addition, the site number is where the entities resident is identified. The relation cardinalities, the tuple sizes, and the number of sites is randomly generated in range [10, 100], [10, 50], and [2, 10], respectively. The second part generates a chain query depending on the schema generated in the first part and uses the number of required joins as an input. For example, the generator can be used to generate the query with four joins (e.g., QJ1, QJ2, QJ3, and QJ4). The queries generated by the generator, in this case, are random and not dependent on a specific application or database. The second group of tests, experiments 4 and 5, are performed on the TPC-H benchmark dataset, as used in [36]. The TPC-H benchmark is a relational data model database that is distributed vertically, as used in our experiments, in a different site location. This database is a decision support benchmark that consists of a suite of business-oriented ad-hoc queries and concurrent data modifications. This benchmark is used to examine large volumes of data and execute queries with a high degree of complexity. Tables 2 and 3 show the tables' size as in [36] and data distributions on different sites.

Table 2. TPC-H tables [36].

Table Name	Number of Tuples	Tuple Size	Table Name	Number of Tuples	Tuple Size
Lineitem	6,000,000	120	orders	1,500,000	100
Partsupp	800,000	140	Part	200,000	160
Customer	150,000	180	supplier	10,000	160
Nation	25	120	region	5	120

Table 3. Data Distribution.

Site	Tables
1	supplier, partsupp
2	orders, lineitem
3	nation, region, customer
4	Part

All tests and experiments are conducted on a PC with Intel Core I5 2.40GHz processor and DDR 8GB main memory running Microsoft Windows 7 Enterprise 64 bit. All experiments are running with fixed parameters $\alpha = 3$, $\beta = 2$, $\rho = 0.02$ as explained in Table 1. The page size, the network transfer rate, and I/O access rate are set to be 1024, 0.98×10^{-3} , and 0.98×10^{-4} , respectively.

In the first experiment, and as in Figures 5–8, the minimum costs obtained by the quantum-inspired model have been compared with the minimum costs obtained by classi-

cal ACO, as in [15], with fixed number of sites equal to 5 and different ants (from one ant to 5 ants) and run over different count of entities 5, 10, 15, and 20.

Figure 5. Comparison between classical ACO and QIACO (5 entities).

Figure 6. Comparison between classical ACO and QIACO (10 entities).

Figure 7. Comparison between classical ACO and QIACO (15 entities).

Figure 8. Comparison between classical ACO and QIACO (20 entities).

In Figure 5, a small search space resulted from the small number of entities (5) and QIACO with one ant which was enough to cover all the search space. So, the optimum solution (cost) was obtained by single ant only, and adding extra ants to the model will not lead to better solution. In case of 10, 15, and 20 entities, the search space becomes larger and adding more ants positively affected the obtained result. The worst, average, and best cost for different number of ants used in this experiment are briefed in Table 4. The performance was calculated as the improvement percentage between the average cost in ACO and the average cost in QIACO. From Table 4, we can conclude that the QIACO algorithm produces better cost than ACO with different number of ants and for different number of join entities. When the number of entities is increased, the search space is exponentially increased and the classic ACO with few iterations cannot cover this search space. So, in case of 15 and 20 entities, the classical ACO with 300 iterations was compared with QIACO with only 100 iterations. In addition, in this case, the QIACO reaches to a better cost than that of ACO. In case of a query with 5 entities, the search space contained few numbers of alternative solutions and the improvement percentage between classic ACO and quantum ACO not more than 13%. But when the number of entities increased, the corresponding search space increased exponentially which contained a huge number of alternative solutions, and in this case, the effect of the quantum search appear. Here, the improvement percentage for QIACO over classic ACO ranged from 77% to 99%.

Table 4. Comparative results between ACO and QIACO.

No. of Entity	ACO				QIACO				Performance %
	No. of Iterations	Worst Cost	Average Cost	Best Cost	No. of Iterations	Worst Cost	Average Cost	Best Cost	
5	100	0.01341	0.01304	0.01239	100	0.01138	0.01138	0.01138	12.73
10	100	0.88999	0.352626	0.10301	100	0.23963	0.11929	0.06968	66.17
15	300	6538.64340	2367.65590	158.82610	100	58.25450	18.66170	2.34710	99.21
20	300	51,574.72700	20,818.50940	1506.29500	100	9689.55970	4627.87840	890.40500	77.77

In the second experiment, to test the efficiency of the proposed algorithm in case of one central database and a different number of join entities, the number of sites is set as one and the number of join entities will start by 3 up to 15. There exists many databases applications that have complicated queries with a massive number of joins that may be reached to 100 or more. Moreover, in some business applications, like banking and retail systems, the application contains queries with smaller number of joins than less than 10. The maximum number of joins used in [26] and [12] was 10 and 15, respectively. We used the maximum of 15 joins in our experiment that are appropriate. Over 10 runs, a

comparison is conducted between the average cost for classical ACO and QIACO with fixed number of ants equal to 2 and fixed number of iterations equal to 100 as in Figure 9.

Figure 9. Comparison between classical ACO and QIACO (Fixed number of sites =1 and different entities from 3 to 15).

Here, with fewer numbers of joins (less than 5), the same cost (best cost) was obtained in ACO and QIACO. When extra entities are added to the joined query, the search space complexity is exponentially increased. Stating from 9 entities, the QIACO gives average cost better than the average cost given by classical ACO. The classical ACO cannot cover all the search space and may fail in the local minimum, but, the diversity in QIACO covers a much larger space and leads to better cost.

In the third experiment, the suggested model was tested for a fixed number of entities (10 entities) and a different number of sites (from 1 to 5 sites). Over 10 runs, two ants were used with fixed number of iterations equal to 100. Figures 10 and 11 display the minimum and average cost for QIACO and classical ACO.

As shown the figures, with the increase in the number of sites, the minimum and average cost are also increasing, but the QIACO still gives a better cost than classical ACO. The results reveal that QIACO reaches a better performance regarding both the minimum cost and the average cost than classical ACO regardless of number of sites used.

Figure 10. Comparison between minimum cost for classical ACO and QIACO (fixed number of entities = 10 and sites from 1 to 5).

Figure 11. Comparison between average cost for classical ACO and QIACO (fixed number of entities = 10 and sites from 1 to 5).

In the fourth experiment, four queries, shown in Figure 12, from the TPC-H benchmark are used to test the performance of QIACO versus the classical ACO. These queries are chosen because they fetch data from many tables.

```
select *
from customer c, order o, lineitem l, supplier s, nation n,
region r
where c.custkey = o.custkey and o.orderkey
= l.orderkey and l.supkey = s.supkey and
c.nationkey = s.nationkey and s.nationkey =
n.nationkey and n.regionkey = r.regionkey and r.name =
'[REGION]' and o.orderdate >= date '[DATE]' and
o.orderdate < date '[DATE]' + interval '1' year
```

Query 5 (Q5)

```
select *
from customer c, order o, lineitem l, supplier s, nation n1,
nation n2
where c.custkey = o.custkey and o.orderkey
= l.orderkey and l.supkey = s.supkey and
c.nationkey = n1.nationkey and s.nationkey =
n2.nationkey and ((n1.name = '[NATION1]' and
n2.name = '[NATION2]') or (n1.name = '[NATION2]'
and n2.name = '[NATION1]')) and l.shipdate between
date '1995-01-01' and date '1996-12-31'
```

Query 7 (Q7)

```
select *
from customer c, order o, lineitem l, supplier s, part p, nation
n1, nation n2, region r
where c.custkey = o.custkey and o.orderkey = l.orderkey and
l.supkey = s.supkey and p.partkey = l.partkey and
c.nationkey = n1.nationkey and n1.regionkey = r.regionkey
and s.nationkey = n2.nationkey and r.name = '[REGION]'
and p.type = '[TYPE]' and o.orderdate between date '1995-
01-01' and date '1996-12-31'
```

Query 8 (Q8)

```
select *
from order o, lineitem l, supplier s, part p, partsupp ps,
nation n
where o.orderkey = l.orderkey and l.supkey = s.supkey
and p.partkey = l.partkey and s.nationkey = n.nationkey
and ps.supkey = l.supkey and ps.supkey = l.supkey
and ps.partkey = l.partkey and p.name like
'%[COLOR]%'
```

Query 9 (Q9)

Figure 12. TPC-H benchmark queries [36].

Here, a different number of ants, from one to five ants, were employed to get the cost for each query using both methodology classical ACO and QIACO, then the average cost for each query was calculated. As shown in Figure 13, although the average cost in all queries tends to favor the method of quantum ACO, it appears clearly in the case of query No. 8. In query No. 8, a larger search space was produced as a result of an existing number of tables greater than that exist in the other queries. In this case, QIACO can cover larger search space that leads to better QEP. The effect of using more ants to seek for better QEP in query No. 8 is shown in Figure 14. Although the QIACO gives better cost than ACO in all ant numbers, the increase in the number of ants used in both methods leads to a better result. When the number of ants is five, a larger space can cover both methodologies and so ACO and QIACO can reach the same QEP that gives the same optimum cost.

Figure 13. Comparison between average cost for ACO and QIACO using TPC-H benchmark queries.

Figure 14. Effect of using different number of ants on ACO and QIACO (query No. 8).

In the next experiment, the average time used by different number of ants to get the best QEP for query No. 8 in ACO and QIACO was compared as shown in Figure 15. Although adding more ants leads to better cost results (as seen in Figure 14), it is at the expense of the complexity and therefore the time. In QIACO, the complexity of computing the tensor product that is used while merging the qubits negatively affects execution time and this complexity increases with the number of ants added to the model. As shown in Figure 15, when the number of ants is one, the effect of the tensor product in QIACO does not exist and in this case, the time required to get the best QEP in QIACO is less than the classical ACO. When the number of ants increases, the tensor product clearly affects the search time and in this case, QIACO needs more time to reach to the best QEP than the classical ACO.

Figure 15. Time to get best QEP by ACO and QIACO (query No. 8).

6. Conclusions

In this work, we proposed a novel ACO methodology based on the quantum-inspired paradigm to optimize a query by finding the best execution path, in terms of cost, for running query on distributed database. The QIACO model, each entity is represented as a qubit and the quantum partial negation gate is employed in order to update the entity's qubit amplitudes. The gate conditionally is applied on the entity's qubit based on pheromone value and join cost. In our model, the average cost that is obtained by QIACO gives a better performance than the cost obtained by the classical ACO. This is because in QIACO, a much wider solution space can be analyzed due to the structure of the model which is not prescribed in advance but is left to the system, arising from qubit superposition via quantum gates. By merging the ACO and the quantum superposition concepts, we successfully enhanced the cost of the running query over the distributed database. The results show that the calculation cost obtained by QIACO is better than that of the classical ACO. The improvement in performance ranged, in some cases, between 77% and 99%, but at the expense of time. The tensor product used in QIACO affects the search time and needs more time to reach the best QEP than the classical ACO. Additionally, the results imply that, although the classical ACO is used successfully with simple joins that have a small number of join entities, QIACO can also be used with simple and complex queries with numerous join entities. In future work, the plan is to use the hyper-graph, instead of the classical graph, to represent the search space. Using the properties and algorithms of sets in the hyper-graph when representing the search space may affect the search methodology and so enhance the query cost. In addition, the complexity of our proposed algorithm and the results obtained through it will be compared with others quantum versions of swarm intelligence algorithms such as particle swarm optimization, artificial bee colony optimization, and firefly algorithm.

Author Contributions: Conceptualization, A.Y.; Methodology, S.A.M.; Supervision, A.Y. and S.M.D. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Ramakrishnan, R. *Databases Management Systems*, 3rd ed.; McGraw-Hill Inc.: New York, NY, USA, 2003.
2. Tiwari, M.P.; Chande, S.V. Query Optimization Strategies in Distributed Databases. *Int. J. Adv. Eng. Sci.* **2013**, *3*, 23–29.
3. Dokeroğlu, T.; Cosar, A. Dynamic Programming with Ant Colony Optimization Metaheuristic for Optimization of Distributed Database Queries. In Proceedings of the 26th International Symposium on Computer and Information, London, UK, 26–28 September 2011; pp. 107–113.
4. Sharma, M.; Gurvinder, S.; Rajinder, S. A Review of Different Cost-based Distributed Query Optimizers. *Prog. Artif. Intell.* **2019**, *8*, 45–62. [\[CrossRef\]](#)
5. Hameurlain, A.; Morvan, F. Evolution of Query Optimization Methods. *Lect. Note Comput. Vis.* **2009**, *5740*, 211–242.
6. Chen, M.; Yu, P. Using Join Operations as Reducers in Distributed Query Processing. Proceedings of 2nd International Symposium on Databases in Parallel and Distributed System, Dublin, Ireland, 2–4 July 1990.
7. Pramanik, S.; Vineyard, D. Optimizing join queries in distributed databases. *IEEE Trans. Softw. Eng.* **1988**, *14*, 1319–1326. [\[CrossRef\]](#)
8. Raipurkar, A.; Bamnote, G.R. Query Processing in Distributed Database through Data Distribution. *Int. J. Adv. Res. Comput. Commun. Eng.* **2013**, *2*, 1134–1139.
9. Kossmann, D.; Konrad, S. Iterative dynamic programming: A new class of query optimization algorithms. *ACM Trans. Database Syst.* **2000**, *25*, 43–82. [\[CrossRef\]](#)
10. Ioannidis, Y.E.; Younkyung, K. Randomized Algorithms for Optimizing Large Join Queries. *ACM Sigmod Rec.* **1990**, *19*, 312–321. [\[CrossRef\]](#)
11. Horng, J.T.; Cheng-Yan, K.; Baw-Jhiune, L. A Genetic Algorithm for Database Query Optimization. In Proceedings of the First IEEE Conference on Evolutionary Computation, Orlando, FL, USA, 27–29 June 1994; pp. 432–444.

12. Sevinc, E.; Cosar, A. An Evolutionary Genetic Algorithm for Optimization of Distributed Database Queries. *Comput. J.* **2010**, *54*, 717–725. [\[CrossRef\]](#)
13. Ristè, D.; Da Silva, M.P.; Ryan, C.A.; Cross, A.W.; Córcoles, A.D.; Smolin, J.A.; Gambetta, J.M.; Chow, J.M.; Johnson, B.R. Demonstration of quantum advantage in machine learning. *NPJ Quantum Inf.* **2017**, *3*, 16. [\[CrossRef\]](#)
14. Kuo, S.; Yao-Hsin, C.; Chi-Yuan, C. Quantum-inspired algorithm for cyber-physical visual surveillance deployment systems. *Comput. Netw.* **2017**, *117*, 5–18. [\[CrossRef\]](#)
15. Mohsin, S.A.; Darwish, S.M.; Younes, A. Dynamic Cost Ant Colony Algorithm for Optimize Distributed Database Query. In Proceedings of the Artificial Intelligence and Computer Vision (AICV2020), Cairo, Egypt, 8–10 April 2020.
16. Dorigo, M.; Birattari, M.; Stützle, T. Ant Colony Optimization. *IEEE Comput. Intell. Mag.* **2006**, *1*, 28–39. [\[CrossRef\]](#)
17. Dorigo, M.; Stutzle, T. *Ant Colony Optimization*; MIT Press: Cambridge, MA, USA, 2004.
18. Han, K.H.; Kim, J.H. Quantum-inspired evolutionary algorithm for a class of combinatorial optimization. *IEEE Trans. Evol. Comput.* **2002**, *6*, 580–593. [\[CrossRef\]](#)
19. Narayan, A.; Chellapilla, P. A novel quantum evolutionary algorithm for quadratic knap-sack problem. In Proceedings of the 2009 IEEE International Conference on Systems, Man and Cybernetics, San Antonio, TX, USA, 11–14 October 2009; pp. 1388–1392.
20. Kaye, P.; Laflamme, R.; Mosca, M. *An Introduction to Quantum Computing*; Oxford University Press: New York, NY, USA, 2007.
21. Rana, M.S.; Mohammad, K.S.; Shohel, A. Distributed Database Problems, Ap-proaches and Solutions—A Study. *Int. J. Mach. Learn. Comput.* **2018**, *8*, 472–476.
22. Steinbrunn, M.; Moerkotte, G.; Kemper, A. Heuristic and Randomized Optimization for the Join-Ordering Problem. *Int. J. Very Large Data Bases* **1997**, *6*, 191–208. [\[CrossRef\]](#)
23. Zhou, Z. Using Heuristics and Genetic Algorithms for Largescale Database Query Optimization. *J. Inf. Comput. Sci.* **2007**, *2*, 261–280.
24. Ban, W.; Jiming, L.; Jichao, T.; Shiwen, L. Query optimization of distributed database based on par-allel genetic algorithm and max-min ant system. In Proceedings of the 8th International Symposium on Computational Intelligence and Design (ISCID), Hangzhou, China, 12–13 December 2015; Volume 2, pp. 581–585.
25. Li, N.; Liu, Y.; Dong, Y.; Gu, J. Application of Ant Colony Optimization Algorithm to Multi-Join Query Optimization. In Proceedings of the 3rd International Symposium on Intelligence Computation and Applications, Berlin/Heidelberg, Germany, 19–21 December 2008.
26. Golshanara, L.; Rankoohi, S.M.T.R.; Shah-Hosseini, H. A multi-colony ant algorithm for optimizing join queries in distributed database systems. *Knowl. Inf. Syst.* **2014**, *39*, 175–206. [\[CrossRef\]](#)
27. Zhang, G. Quantum-inspired evolutionary algorithms: A survey and empirical study. *J. Heuristics* **2011**, *17*, 303–351. [\[CrossRef\]](#)
28. Chmiel, W.; Kwiecień, J. Quantum-Inspired Evolutionary Approach for the Quadratic Assignment Problem. *Entropy* **2018**, *20*, 781. [\[CrossRef\]](#)
29. Darwish, S.M.; Shendi, T.A.; Younes, A. Quantum-inspired genetic programming model with application to predict toxicity degree for chemical compounds. *Expert Syst.* **2019**, *36*, e12415. [\[CrossRef\]](#)
30. Dey, S.; Bhattacharyya, S.; Maulik, U. New quantum inspired meta-heuristic techniques for multi-level colour image thresholding. *Appl. Soft Comput.* **2016**, *46*, 677–702. [\[CrossRef\]](#)
31. Tiwari, P.; Chande, S.V. Optimal Ant and Join Cardinality for Distributed Query Optimization Using Ant Colony Optimization Algorithm. In Proceedings of the 2nd International Symposium on Emerging Trends in Expert Applications and Security, Singapore, 17–18 February 2018.
32. Younes, A. Reading a single qubit system using weak measurement with variable strength. *Ann. Phys.* **2017**, *380*, 93–105. [\[CrossRef\]](#)
33. Duan, H. Ant colony optimization: Principle, convergence and application. In *Handbook of Swarm Intelligence*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 373–388.
34. Kazharov, A.A.; Kureichik, V.M. Ant colony optimization algorithms for solving transportation problems. *J. Comput. Syst. Sci. Int.* **2010**, *49*, 30–43. [\[CrossRef\]](#)
35. James, M. Test Run—Ant Colony Optimization. *Msdn Mag.* **2012**, *27*, 2. Available online: <https://docs.microsoft.com/en-us/archive/msdn-magazine/2012/february/test-run-ant-colony-optimization> (accessed on 30 December 2020).
36. Deshpande, A.; Hellerstein, J.M. Decoupled query optimization for federated database systems. In Proceedings of the 18th International Conference on Data Engineering, San Jose, CA, USA, 26 February–1 March 2002; pp. 716–727.