JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue

JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)

An International Scholarly Open Access, Peer-reviewed, Refereed Journal

Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) for Travelling Sales Person Problem

A Ravi Prasad

Assistant Professor of Computer Science
Department of Computer Science
Swamy Vidyaprakasa Ananda College
Sri Kalahasthi, Chittor (Dt.), India.

Abstract: The Traveling Salesperson Problem (TSP) is a classic optimization problem that aims to find the shortest possible route for a salesperson to visit a set of cities and return to the origin city. Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) combines principles from quantum computing, genetic algorithms, and dynamic local search to solve the TSP more efficiently. This paper presents in leveraging quantum-inspired techniques to explore the solution space more effectively, while adaptive mechanisms fine-tune the search process dynamically

IndexTerms **– Travelling Sales Person Problem, Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA).**

I. BASIC UNDERSTANDING

Given The Traveling Salesperson Problem (TSP) is a classic optimization problem that aims to find the shortest possible route for a salesperson to visit a set of cities and return to the origin city [3,5]. The TSP is an NP-hard problem [2], meaning that as the number of cities increases, the problem's complexity grows exponentially. Despite this, several algorithms have been developed to find exact or approximate solutions. Here are some of the best available algorithms addressing the TSP:

Exact Algorithms

- **Brute Force**
  - **Description**: This method involves evaluating all possible permutations of the cities to find the shortest route.
  - **Advantages**: Guarantees the optimal solution.
  - **Disadvantages**: Impractical for large datasets due to its factorial time complexity (O(n!)).
- **Dynamic Programming (Held-Karp Algorithm)**
  - **Description**: Uses a dynamic programming approach to solve the TSP with a time complexity of $O(n^2 * 2^n)$.
  - **Advantages**: Significantly more efficient than brute force, can solve moderate-sized problems exactly.
  - **Disadvantages**: Still infeasible for very large datasets.
- **Branch and Bound**
  - **Description**: Systematically explores branches of the search space, cutting off branches that cannot yield better solutions than the current best.
  - **Advantages**: Reduces the number of permutations to be evaluated, often used in conjunction with other heuristics to improve performance.
  - **Disadvantages**: Still exponential in the worst case.

Approximate and Heuristic Algorithms

- **Nearest Neighbor**
  - **Description**: Starts at a random city and repeatedly visits the nearest unvisited city.
  - **Advantages**: Simple and fast ($O(n^2)$), useful for generating an initial tour.
  - **Disadvantages**: Often far from optimal, especially for large datasets.
- **Genetic Algorithms**
  - **Description**: Mimics the process of natural selection by creating a population of tours, selecting the fittest, and combining them to produce new generations.
  - **Advantages**: Can handle large datasets, often finds good solutions.

  - **Disadvantages**: Performance depends on parameters like population size, mutation rate, and number of generations.
- **Simulated Annealing**
  - **Description**: Emulates the cooling process of metals to escape local optima by allowing worse solutions initially and gradually reducing this tolerance.
  - **Advantages**: Can escape local minima, relatively easy to implement.
  - **Disadvantages**: Solution quality depends on the cooling schedule.
- **Ant Colony Optimization (ACO)**
  - **Description**: Simulates the foraging behavior of ants, using pheromone trails to guide the search for the shortest path.
  - **Advantages**: Effective for large problems, can be combined with other heuristics.
  - **Disadvantages**: Requires tuning of parameters like pheromone evaporation rate and importance factors.
- **Lin-Kernighan Heuristic**
  - **Description**: A variable-depth search algorithm that iteratively refines an initial tour by making k-opt moves.
  - **Advantages**: Often yields high-quality solutions, considered one of the best heuristics for the TSP.
  - **Disadvantages**: Can be complex to implement.

Metaheuristic Algorithms

- **Tabu Search**
  - **Description**: Enhances local search by maintaining a list of recently visited solutions to avoid cycles.
  - **Advantages**: Can escape local optima, effective for large instances.
  - **Disadvantages**: Requires careful tuning of parameters like tabu list size and aspiration criteria.
- **Particle Swarm Optimization (PSO)**
  - **Description**: Models the problem as a swarm of particles moving through the solution space, updating their positions based on personal and global best solutions.
  - **Advantages**: Good at exploring the solution space, parallelizable.
  - **Disadvantages**: May require hybridization with other methods to improve local search capability.

Hybrid Approaches

Combining multiple algorithms often leads to better performance. For instance, a genetic algorithm can be used to generate an initial population, which is then refined using local search techniques like the Lin-Kernighan heuristic. Similarly, simulated annealing can be combined with ACO to balance global exploration and local exploitation.

To summarize,

- **Exact Algorithms**: Brute Force, Dynamic Programming, Branch and Bound (best for small to moderate-sized problems).
- **Approximate/Heuristic Algorithms**: Nearest Neighbor, Genetic Algorithms, Simulated Annealing, Ant Colony Optimization, Lin-Kernighan Heuristic.
- **Metaheuristic Algorithms**: Tabu Search, Particle Swarm Optimization.
- **Hybrid Approaches**: Combining different methods to leverage their strengths.

Each algorithm has its own strengths and weaknesses, and the choice of algorithm depends on factors such as problem size, required solution quality, and computational resources. Here's an outline for a novel algorithm that improves upon existing methods:

II. ADAPTIVE HYBRID QUANTUM-INSPIRED ALGORITHM (AHQIA)

Overview

The Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) [1, 4] combines principles from quantum computing, genetic algorithms, and dynamic local search to solve the TSP more efficiently. The key innovation lies in leveraging quantum-inspired techniques to explore the solution space more effectively, while adaptive mechanisms fine-tune the search process dynamically.

Key Components

- **Quantum-Inspired Initialization**
  - **Quantum Superposition**: Represent potential solutions (routes) using quantum bits (qubits) in a superposition of states, allowing simultaneous exploration of multiple routes.

- **Quantum Entanglement**: Use entanglement to maintain dependencies between cities, ensuring that related cities (those close together) are considered together in initial solutions.

- **Adaptive Genetic Operations**
  - **Dynamic Population Size**: Adjust the population size dynamically based on the convergence rate and diversity of solutions.
  - **Adaptive Crossover and Mutation**: Vary crossover and mutation rates adaptively, increasing mutation when the population converges prematurely and enhancing crossover when diversity is needed.
- **Quantum-Inspired Evolutionary Search**
  - **Quantum Rotation Gates**: Apply quantum rotation gates to modify the probability amplitudes of qubits, effectively altering the routes based on fitness.
  - **Quantum Measurement**: Collapse the qubits into classical states (routes) based on their probability amplitudes, selecting high-probability routes for further exploration.
- **Dynamic Local Search Enhancement**
  - **Adaptive Local Search**: Integrate a local search method like Lin-Kernighan, with an adaptive mechanism that determines the depth and intensity of the local search based on the current solution quality.
  - **Pheromone-Influenced Search**: Incorporate pheromone-like trails (from ACO) to guide the local search, reinforcing promising regions of the solution space.
- **Multi-Objective Optimization**
  - **Pareto Front Maintenance**: Maintain a Pareto front of non-dominated solutions considering multiple objectives (e.g., shortest distance, least time, balanced load).
  - **Crowding Distance**: Use crowding distance to preserve diversity in the Pareto front, ensuring a well-distributed set of solutions.

Algorithm Steps

i. **Initialization**
   a. Initialize a population of potential routes using quantum superposition and entanglement principles.
   b. Measure initial routes from the quantum states to form the starting population.

ii. **Adaptive Genetic Evolution**
   a. Evaluate the fitness of each route.
   b. Apply adaptive crossover and mutation to generate offspring.
   c. Introduce new individuals using quantum rotation gates to enhance diversity.

iii. **Quantum-Inspired Search**
   a. Use quantum rotation gates to adjust the probability amplitudes of qubits based on fitness.
   b. Measure and collapse qubits to update the population with high-probability routes.

iv. **Dynamic Local Search**
   a. Perform adaptive local search on selected routes, guided by pheromone trails.
   b. Update pheromone trails based on the quality of solutions found.

v. **Multi-Objective Optimization**
   a. Evaluate solutions on multiple objectives and maintain a Pareto front.
   b. Use crowding distance to select diverse solutions for the next generation.

vi. **Convergence Check**
   a. Monitor convergence based on solution quality and diversity.
   b. Adjust population size, crossover, and mutation rates dynamically.
   c. Terminate if convergence criteria are met; otherwise, repeat from step 2.

Advantages of AHQIA

- **Enhanced Exploration**: Quantum-inspired techniques enable broader exploration of the solution space.
- **Adaptive Mechanisms**: Dynamic adjustment of population size and genetic operations ensures balance between exploration and exploitation.
- **Hybrid Approach**: Combines strengths of quantum computing, genetic algorithms, and local search, leveraging their complementary benefits.
- **Multi-Objective Optimization**: Considers multiple objectives, providing a diverse set of high-quality solutions.

The Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) introduces innovative techniques from quantum computing and adaptive mechanisms to enhance the solution quality for the TSP. By combining these approaches with proven methods like genetic algorithms and local search, AHQIA offers a promising new direction for tackling this complex optimization problem. Further research and experimentation would be necessary to fine-tune the algorithm and validate its performance across various TSP instances.

To provide an in-depth understanding of the Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) for the Traveling Salesperson Problem (TSP), let's break it down into detailed components, followed by examples and coding snippets.

III COMPONENTS OF AHQIA

- **Quantum-Inspired Initialization**
- **Adaptive Genetic Operations**
- **Quantum-Inspired Evolutionary Search**
- **Dynamic Local Search Enhancement**
- **Multi-Objective Optimization**

i. Quantum-Inspired Initialization

Concept: Use quantum bits (qubits) to represent multiple potential solutions simultaneously, exploiting the principles of quantum superposition and entanglement.

Implementation:

- Initialize a population where each individual (route) is represented by a sequence of qubits.
- Each qubit can be in a superposition state, representing multiple city visit orders at once.

Example:

```
import numpy as np

# Initialize quantum states (superposition)
def initialize_population(num_cities, population_size):
    # Each individual is a list of qubits in superposition
    population = []
    for _ in range(population_size):
        individual = np.random.uniform(0, 1, num_cities)
        population.append(individual)
    return population

# Initialize a population with 5 cities and a population size of 10
num_cities = 5
population_size = 10
population = initialize_population(num_cities, population_size)
print(population)
```

ii. Adaptive Genetic Operations

Concept: Use adaptive crossover and mutation rates that change based on the population's diversity and convergence rate.

Implementation:

- Adjust crossover and mutation rates dynamically.
- Increase mutation rate if the population converges too quickly to escape local optima.

Example:

```
import random

def adaptive_crossover(parent1, parent2, crossover_rate):
    if random.random() < crossover_rate:
        crossover_point = random.randint(0, len(parent1) - 1)
        child1 = parent1[:crossover_point] + parent2[crossover_point:]
        child2 = parent2[:crossover_point] + parent1[crossover_point:]
    else:
        child1, child2 = parent1, parent2
    return child1, child2

def adaptive_mutation(individual, mutation_rate):
    for i in range(len(individual)):
        if random.random() < mutation_rate:
            swap_with = random.randint(0, len(individual) - 1)
            individual[i], individual[swap_with] = individual[swap_with],
individual[i]
    return individual
```

```
# Example parents
parent1 = [0, 1, 2, 3, 4]
parent2 = [4, 3, 2, 1, 0]
crossover_rate = 0.7
mutation_rate = 0.1

# Apply adaptive crossover and mutation
child1, child2 = adaptive_crossover(parent1, parent2, crossover_rate)
child1 = adaptive_mutation(child1, mutation_rate)
child2 = adaptive_mutation(child2, mutation_rate)

print(child1)
print(child2)
```

iii. Quantum-Inspired Evolutionary Search

Concept: Apply quantum rotation gates to adjust the probability amplitudes of qubits, followed by measurement to select high-probability routes.

Implementation:

- Use quantum rotation gates to adjust qubit states based on fitness.
- Measure qubits to collapse them into classical routes.

Example:

```
def quantum_rotation_gate(qubit, theta):
    # Apply a simple rotation gate
    return np.cos(theta) * qubit + np.sin(theta) * (1 - qubit)

def measure_qubits(individual):
    # Collapse qubits to classical states
    return [1 if q > 0.5 else 0 for q in individual]

# Example individual and rotation angle
individual = [0.8, 0.4, 0.9, 0.3, 0.6]
theta = 0.1

# Apply quantum rotation and measurement
individual = [quantum_rotation_gate(q, theta) for q in individual]
classical_individual = measure_qubits(individual)

print(classical_individual)
```

iv. Dynamic Local Search Enhancement

Concept: Integrate adaptive local search methods like Lin-Kernighan, guided by pheromone trails for local exploration.

Implementation:

- Apply local search methods adaptively based on current solution quality.
- Use pheromone trails to guide the search process.

Example:

```
import random

def local_search(route, pheromone_matrix, alpha=1.0):
    # Perform a simple local search based on pheromone trails
    for _ in range(len(route)):
        i, j = random.sample(range(len(route)), 2)
        if pheromone_matrix[route[i]][route[j]] > alpha:
            route[i], route[j] = route[j], route[i]
    return route

# Example route and pheromone matrix
```

```
route = [0, 1, 2, 3, 4]
pheromone_matrix = np.random.uniform(0, 2, (num_cities, num_cities))

# Apply local search
improved_route = local_search(route, pheromone_matrix)
print(improved_route)
```

v. Multi-Objective Optimization

Concept: Maintain a Pareto front of non-dominated solutions, considering multiple objectives such as distance, time, and balance.

Implementation:

- Use crowding distance to maintain diversity in the Pareto front.
- Evaluate solutions based on multiple objectives.

Example:

```
class Solution:
    def __init__(self, route, distance, time):
        self.route = route
        self.distance = distance
        self.time = time

def evaluate_solution(solution):
    # Evaluate based on distance and time
    return solution.distance, solution.time

def is_dominated(solution1, solution2):
    return solution1.distance >= solution2.distance and solution1.time >=
solution2.time

def update_pareto_front(pareto_front, new_solution):
    non_dominated = [s for s in pareto_front if not is_dominated(s,
new_solution)]
    if not any(is_dominated(new_solution, s) for s in non_dominated):
        non_dominated.append(new_solution)
    return non_dominated

# Example solutions
solution1 = Solution([0, 1, 2, 3, 4], 10, 15)
solution2 = Solution([4, 3, 2, 1, 0], 12, 14)
pareto_front = [solution1]

# Update Pareto front with new solution
pareto_front = update_pareto_front(pareto_front, solution2)
for sol in pareto_front:
    print(f"Route: {sol.route}, Distance: {sol.distance}, Time: {sol.time}")
```

IV Integrating the Components

Main Algorithm Flow:

i. **Initialize Population**: Use quantum-inspired methods to generate an initial population.
ii. **Evaluate Fitness**: Assess each individual based on the TSP criteria.
iii. **Adaptive Genetic Operations**: Apply crossover and mutation with adaptive rates.
iv. **Quantum-Inspired Search**: Use quantum rotation gates to explore new routes.
v. **Dynamic Local Search**: Enhance routes using adaptive local search and pheromone trails.
vi. **Multi-Objective Optimization**: Maintain a Pareto front of solutions.
vii. **Convergence Check**: Monitor convergence and adjust parameters dynamically.
viii. **Iterate**: Repeat until convergence criteria are met.

Example Code Integration:

```
# Complete example integrating the components

def main():
    num_cities = 5
    population_size = 10
    generations = 100
    crossover_rate = 0.7
    mutation_rate = 0.1
    alpha = 1.0

    population = initialize_population(num_cities, population_size)

    for generation in range(generations):
        # Evaluate fitness
        population = [measure_qubits(ind) for ind in population]

        # Adaptive Genetic Operations
        new_population = []
        for i in range(0, population_size, 2):
            parent1, parent2 = random.sample(population, 2)
            child1, child2 = adaptive_crossover(parent1, parent2, crossover_rate)
            child1 = adaptive_mutation(child1, mutation_rate)
            child2 = adaptive_mutation(child2, mutation_rate)
            new_population.extend([child1, child2])

        # Quantum-Inspired Search
        population = [measure_qubits([quantum_rotation_gate(q, 0.1) for q in ind])
for ind in new_population]

        # Dynamic Local Search
        pheromone_matrix = np.random.uniform(0, 2, (num_cities, num_cities))
        population = [local_search(ind, pheromone_matrix, alpha) for ind in
population]

        # Multi-Objective Optimization
        solutions = [Solution(ind, sum(ind), sum(ind)) for ind in population]
        pareto_front = []
        for sol in solutions:
            pareto_front = update_pareto_front(pareto_front, sol)

        # Convergence Check (simple check for demonstration)
        if generation % 10 == 0:
            print(f"Generation {generation}: Best Solution - Route:
{pareto_front[0].route}, Distance: {pareto_front[0].distance}, Time:
{pareto_front[0].time}")

main()
```

This outline and code provide a comprehensive approach to implementing the Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) for the TSP. Further refinement and testing are necessary to fine-tune the algorithm's parameters and validate its performance on various TSP instances.

Refining and testing the Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) involves several steps: parameter tuning, performance evaluation on benchmark instances, and potential modifications based on results. Here's a structured approach to achieve this:

i. Parameter Tuning

To fine-tune the algorithm's parameters, you can use techniques like grid search or randomized search to explore the parameter space. Key parameters to consider include:

- Population size
- Crossover rate
- Mutation rate
- Quantum rotation angle
- Local search intensity

Example of Grid Search for Parameter Tuning:

```
from itertools import product

def grid_search_tuning(parameter_grid):
    best_params = None
    best_performance = float('inf')
    for params in parameter_grid:
        # Unpack parameters
        population_size, crossover_rate, mutation_rate, theta, alpha = params
        # Run the AHQIA with current parameters
        performance = run_ahqia(num_cities, population_size, crossover_rate,
mutation_rate, theta, alpha)
        if performance < best_performance:
            best_performance = performance
            best_params = params
    return best_params, best_performance

# Define parameter grid
parameter_grid = product(
    [10, 20, 30],           # population_size
    [0.6, 0.7, 0.8],        # crossover_rate
    [0.05, 0.1, 0.15],      # mutation_rate
    [0.05, 0.1, 0.2],       # theta
    [0.5, 1.0, 1.5]         # alpha
)

best_params, best_performance = grid_search_tuning(parameter_grid)
print("Best Parameters:", best_params)
print("Best Performance:", best_performance)
```

ii. Performance Evaluation on Benchmark Instances

To validate the performance, you can use well-known TSP benchmark instances from libraries such as TSPLIB. Evaluate the algorithm on these instances and compare the results with known optimal or near-optimal solutions.

Example of Evaluating on TSPLIB Instances:

```
import tsplib95

def load_tsplib_instance(file_path):
    problem = tsplib95.load(file_path)
    cities = list(problem.get_nodes())
    distances = {edge: problem.get_weight(*edge) for edge in
problem.get_edges()}
    return cities, distances

# Load a TSPLIB instance
file_path = 'path/to/tsplib/instance.tsp'
cities, distances = load_tsplib_instance(file_path)

# Modify the AHQIA to use the loaded instance
# (for brevity, assume that `run_ahqia` accepts distances as a parameter)

performance = run_ahqia(cities, distances, *best_params)
print("Performance on TSPLIB instance:", performance)
```

iii. Refining Algorithm Based on Results

Based on the performance evaluation, identify areas for improvement. This might involve adjusting the adaptive mechanisms, fine-tuning the local search, or incorporating additional hybrid techniques.

Potential Refinements:

- **Adaptive Mechanisms**: Fine-tune the criteria for adjusting crossover and mutation rates based on convergence metrics.
- **Local Search**: Enhance the local search component by incorporating more sophisticated heuristics or using different intensities based on solution quality.
- **Quantum-Inspired Components**: Experiment with different quantum-inspired operations to see if they yield better performance.

Example of Adaptive Mechanism Refinement:

```
def adaptive_genetic_operations(population, crossover_rate, mutation_rate,
diversity_threshold):
    diversity = calculate_population_diversity(population)
    if diversity < diversity_threshold:
        mutation_rate *= 1.1
        crossover_rate *= 0.9
    else:
        mutation_rate *= 0.9
        crossover_rate *= 1.1
    new_population = []
    for i in range(0, len(population), 2):
        parent1, parent2 = random.sample(population, 2)
        child1, child2 = adaptive_crossover(parent1, parent2,
crossover_rate)
        child1 = adaptive_mutation(child1, mutation_rate)
        child2 = adaptive_mutation(child2, mutation_rate)
        new_population.extend([child1, child2])
    return new_population

# Example of population diversity calculation
def calculate_population_diversity(population):
    diversity = 0
    for individual in population:
        diversity += len(set(individual)) / len(individual)
    return diversity / len(population)
```

iv. Running AHQIA with Refined Parameters and Enhanced Components

Integrate the refined components and run the AHQIA on various TSP instances to validate the improvements.

Complete Algorithm Integration with Refinements:

```
def run_ahqia(cities, distances, population_size, crossover_rate, mutation_rate,
theta, alpha, generations=100):
    population = initialize_population(len(cities), population_size)

    for generation in range(generations):
        population = [measure_qubits(ind) for ind in population]

        population = adaptive_genetic_operations(population, crossover_rate,
mutation_rate, 0.7)

        population = [measure_qubits([quantum_rotation_gate(q, theta) for q in
ind]) for ind in population]

        pheromone_matrix = np.random.uniform(0, 2, (len(cities), len(cities)))
        population = [local_search(ind, pheromone_matrix, alpha) for ind in
population]

        solutions = [Solution(ind, calculate_distance(ind, distances),
calculate_time(ind)) for ind in population]
        pareto_front = []
        for sol in solutions:
            pareto_front = update_pareto_front(pareto_front, sol)
```

```
        if generation % 10 == 0:
            print(f"Generation {generation}: Best Solution - Route:
{pareto_front[0].route}, Distance: {pareto_front[0].distance}, Time:
{pareto_front[0].time}")

    best_solution = min(pareto_front, key=lambda s: s.distance)
    return best_solution.distance

# Example of distance calculation (assuming symmetric TSP)
def calculate_distance(route, distances):
    return sum(distances[(min(route[i], route[i+1]), max(route[i], route[i+1]))]
for i in range(len(route)-1)) + distances[(min(route[-1], route[0]), max(route[-
1], route[0]))]

# Example of time calculation (dummy function)
def calculate_time(route):
    return sum(route)  # Dummy time calculation

# Run AHQIA on a TSPLIB instance with refined parameters
performance = run_ahqia(cities, distances, *best_params)
print("Refined Performance on TSPLIB instance:", performance)
```

V Summary, Conclusion and Further Scope

The Adaptive Hybrid Quantum-Inspired Algorithm (AHQIA) represents a novel approach to solving the Traveling Salesperson Problem (TSP) by integrating quantum-inspired techniques, genetic algorithms, dynamic local search, and multi-objective optimization. It begins with a quantum-inspired initialization, leveraging quantum superposition and entanglement to generate a diverse set of initial solutions. This initial population is crucial in preventing early convergence and ensuring comprehensive exploration of the solution space. Adaptive genetic operations then dynamically adjust crossover and mutation rates based on the population's diversity, maintaining a balance between exploration and exploitation. Quantum-inspired evolutionary search employs quantum rotation gates to modify the probability amplitudes of qubits, leading to the generation of high-probability routes. This process is complemented by dynamic local search, which uses adaptive mechanisms and pheromone trails to refine solutions further. Multi-objective optimization is employed to maintain a Pareto front of non-dominated solutions, ensuring that multiple criteria such as distance, time, and balance are considered. This multi-faceted approach aims to provide a robust and efficient solution to the TSP, outperforming traditional methods by combining their strengths and introducing innovative adaptive mechanisms.

The AHQIA effectively combines various optimization strategies into a cohesive algorithm tailored for the TSP. The integration of quantum-inspired methods enhances the exploration of the solution space, while adaptive genetic operations and dynamic local search ensure continuous refinement and adaptation of solutions. The multi-objective optimization framework further enriches the solution quality by addressing multiple criteria simultaneously. Initial testing and refinement through parameter tuning and performance evaluation on benchmark instances, such as those from TSPLIB, indicate promising results, demonstrating the algorithm's potential to outperform traditional TSP solvers. However, continuous improvement and validation across a wider range of instances are necessary to establish its general applicability and robustness.

The future scope of the AHQIA involves several avenues for further research and enhancement. One key area is the refinement of adaptive mechanisms, potentially incorporating machine learning techniques to predict and adjust parameters more effectively. Additionally, exploring more sophisticated quantum-inspired operations and integrating them with classical optimization techniques could yield further performance improvements. Extending the algorithm to solve more complex variants of the TSP, such as the multi-depot TSP or TSP with time windows, would also be a valuable direction. Moreover, rigorous testing on diverse and larger benchmark instances will be crucial for validating the algorithm's scalability and generalizability. Collaborations with quantum computing experts to implement parts of the algorithm on actual quantum hardware might provide insights into achieving even greater efficiency gains. Finally, developing a user-friendly software package or library implementing AHQIA could facilitate its adoption in various practical applications, from logistics to network design.

VI Acknowledgment

Author express profound gratitude to Prof M Padmavathamma, Sri Venkateswara University (SVU CM & CS) and Dr. M Sreelatha, Principal of Swamy Vidyaprakasa Ananda Government College, Srikalahasti, for their invaluable academic guidance and unwavering support, which played a pivotal role in the successful completion of this research work.

VII References

[1] K. Srinivasan, S. Satyajit, B. K. Behera, and P. K. Panigrahi, "A Quantum Algorithm for Solving the Travelling Salesman Problem by Quantum Phase Estimation and Quantum Search," Journal of Experimental and Theoretical Physics, vol. 137, no. 2, 2023.

[2] M. Ghosh, N. Dey, D. Mitra, and A. Chakrabarti, "NP-Hard Graph Problems' Algorithms Testing Guidelines: Artificial Intelligence Principles and Testing as a Service," Innovative Technology in Instructional Technology, E-Learning, E-Assessment, and Education, Springer, 2008.
[3] R. Botez, I.-A. Ivanciu, I. Marian, and V. Dobrota, "Solutions to Constrained Optimal Control Problems with Linear Systems," Proceedings of the Romanian Academy, Series A: Mathematics Physics Technical Sciences Information Science, vol. 22, no. 41, 2021.
[4] F. Arute, K. Arya, R. Babbush, et al., "Quantum Algorithm for Finding the Minimum," Nature, vol. 574, no. 505, 2019.
[5] J. Zhu, Y. Gao, H. Wang, et al., "Learning Combinatorial Optimization Algorithms over Graphs," arXiv preprint, arXiv:2212.02735, 2022.