Journal of Intelligent Systems and Internet of Things
Vol. 18, No. 02, PP. 72-84, 2026
DOI: https://doi.org/10.54216/JISIoT.180205

Developing a Fast Hybrid Metaheuristic Algorithm to Enhance the Efficiency of Resource-Constrained Applications

Alaa Abdalqahar Jihad[1], Ahmed Subhi Abdalkafor[2], Sameeh Abdulghafour Jassim[3,4,*]
[1]Computer Center, University of Anbar, Anbar, Iraq
[2]College of Computer Science and Information Technology, University of Anbar, Anbar, Iraq
[3]Department of Vocational Education in Anbar, Ministry of Education, Anbar, Iraq
[4]Department of Computer Sciences, College of Science, University of Al Maarif, Al Anbar, 31001, Iraq
Emails: it.alaa.heety@uoanbar.edu.iq; ahmed.abdalkafor@uoanbar.edu.iq; sameeh@uoa.edu.iq

Abstract

The rapid development of intelligent computing has led to Internet of Things (IoT) applications and embedded devices suffering from severe constraints on energy, processing, and memory. This calls for fast and lightweight algorithms that maintain performance accuracy without draining resources or affecting response time. This paper presents a new hybrid metaheuristic algorithm that combines the advantages of four optimization algorithms to achieve efficient results and reduce computational complexity without compromising output quality. Experiments demonstrate significant improvements in performance and execution time compared to traditional algorithms, in addition to the algorithm's ability to scale and handle diverse workloads. The lowest improvement of the proposed algorithm compared to other algorithms was approximately 25.7%. This algorithm opens up prospects for effective applications in smart systems in urban and industrial areas.

Received: February 25, 2025 Revised: June 05, 2025 Accepted: July 02, 2025

Keywords: Metaheuristic Algorithms; Hybrid Algorithms; Resource-Constrained Applications; Internet of Things (IoT)

1. Introduction

Rapid technology advancement in the fields of artificial intelligence, the Internet of Things (IoT), and modern computing applications have become a pillar of support in fields like financial analysis, medical diagnosis, and autonomous driving. These applications impose strict requirements in terms of high performance and fast response to data processing, especially in resource-constrained IoT applications. Therefore, the seek for fast optimization algorithms is essential to ensure the operational efficiency of these systems and achieve optimal performance [1][2].

Internet of Things devices rely on limited computing resources, memory, and energy, which calls for the development of lightweight, fast applications capable of running locally, while leveraging the cloud when needed to process and analyze large volumes of data [3]. Cloud computing helps compensate for these limitations by providing a flexible and scalable infrastructure for receiving, storing, and processing data sent from devices [4][5]. This allows devices to focus their efforts on collecting and transmitting information with minimal energy consumption and without influencing immediate performance. Furthermore, designing lightweight applications relies on low-performance programming techniques, which helps maintain operational continuity and reduce energy consumption.

Nowadays, smart cities and smart homes are among the most prominent technical applications that aim on improving the quality of life through the integration of modern technology with the contemporary life. The smart city systems rely heavily on Internet of Things technologies, where connected devices interact to share data more

efficiently to improve operations such as energy management, traffic optimization, and public services provision [6]. Despite the many benefits of these technologies, they pose significant challenges in managing and analyzing data in real-time. These challenges attract the researchers; attention to development highly efficient optimization algorithms with fast response times for IoT [7][8].

IoT applications in smart cities require optimization algorithms to solve complex problems such as optimal path selection, efficient resources allocation, etc. Therefore, metaheuristic algorithms are gaining increasing significance for their ability to come up with near-optimal solutions in a limited time frame, especially for problems that are difficult to address with traditional approaches [9][10]. This ability is supported by high effectiveness of exploring search spaces. These advantages can be utilized in dynamic and complex environments such as the Internet of Things [11][12].

The importance of this study relies upon the critical need for a hybrid algorithm that is characterized by speed to improve the efficiency of resource-constrained intelligent applications. The research sets out to answer the straightforward question: can a hybrid metaheuristic algorithm achieve an optimal trade-off between the rate of execution and resource utilization efficiency without sacrificing the? The hypothesis states that the combination of different strategies of metaheuristic algorithms can lead to an algorithm that outperforms traditional methods in performance and effectiveness, which is the main point of this research. The study mainly aims to:

1. Reducing the time of computation,
2. Reducing resource consumption,
3. Ensuring accurate results.

The main contribution of this research is to develop a hybrid algorithm that combines the powerful features extracted from the studied metaheuristic algorithms, while providing an applied analysis of the performance of this algorithm, which enhances its practical applicability in the fields of IoT and intelligent applications. This paper continues with the following sections: Section 2 include the literature review. Section 3 explain the hybridization methodology and algorithm fusion mechanism. Section 4 presents the experimental analysis, results, and discussion. Section 5 presents the conclusion and future recommendations.

2. Related Work

Previous studies show that there are a variety of approaches and algorithms used to improve the performance of intelligent systems under the challenges of computational efficiency and resource consumption in modern environments. Researchers have been focusing on developing algorithms that enhance processing speed and accuracy.

In [13], two light and effective algorithms with a single heterogeneous mutation operator (SNUM) were introduced: a single solution algorithm (SNUM) and a combined genetic algorithm (cSNUM). The algorithms have straightforward structure and can be easily integrated with other approaches. The results on the BBOB and CEC-2017 benchmarks showed that both methods were efficient, and cSNUM's performance with different functions, especially when the dimensions of the problem grow larger, rendering it a strong competitor of other algorithms. In [14], Yin-Yang Pair Optimization (YYPO) algorithm was introduced, a low-complexity metaheuristic algorithm with exploration-exploitation balance that operates with two points and three given parameters. The experiments showed competitive performance compared to other algorithms such as artificial bee colony, differential evolution, and gray wolf optimizer with high efficiency and low time complexity, justifying its efficiency in constrained engineering problem solving.

Researchers in [15] proposed a model that attempts to optimize the performance of fog computing in the IoT by using a metaheuristic algorithm that combines EMOPSOC with Fog Picker to address resource scheduling as a multi-objective problem. The model is both dynamic, which maximizes the use of resources and reduces load imbalance. Simulation tests using iFogSim and jMetal indicated that the proposed model outperformed NSGA-II and SPEA-II algorithms, besides improving the allocation efficiency of Fog Picker by 80%. A recent study [16] introduced H-GWO, a hybrid algorithm combining Grey Wolf Optimizer (GWO) and Differential Evolution (DE). This approach enhances community diversity, prevents local optimization, and balances exploration with exploitation. By integrating DE's transformation and intersection techniques with GWO's reversal learning, H-GWO outperformed leading algorithms like PSO, DE, and GWO across nine benchmark functions. When applied with forgetting factor recursive least squares (FFRLS) to identify helical hydraulic rotary actuator (HHRA) parameters under uncertainty, H-GWO demonstrated superior accuracy compared to LS, RLS, FFRLS, PSO, and GWO.

Likewise, the research [17] introduced a hybrid metaheuristic routing (HMBCR) method to solve energy consumption optimization issues. The method integrated brainstorming optimization (BSO-LD) with distributed-fee clustering and water wave optimization (WWO-HC) for identifying the optimal routes. The outcome indicates

HMBCR is more appropriate for enhancing energy efficiency and network lifetime prolongation. The research [18] suggested the Ant Colony Optimization with Fuzzy Logic (ACO-FL) algorithm for USV path planning in dynamic environments while attempting to avoid mobile obstacles in consideration of environmental parameters. Comparison with the original ACO and Bug algorithm indicated that ACO-FL is better in solution quality and convergence rate.

Researchers have explored various metaheuristic and hybrid algorithms in the previous studies to enhances the efficiency of intelligent applications. Despite the effectiveness shown by these algorithms, some challenges still remain, such as striving for a better balance between exploration and exploitation, avoid falling into local optimal solutions, and reduce computational complexity to ensure fast execution in resource-constrained environments. Based on these limitations, the current research seeks to develop a new hybrid algorithm that combines the advantages of different metaheuristic technique to achieve improved performance in terms of speed and adaptability to the requirements of modern intelligent applications.

3. Methodology

This section addresses the utilized optimization algorithms that find usage across numerous applications and analyzes their performance and efficiency. Since IoT applications require fast solutions that can handle tremendous amounts of data and concurrent processes in real-time, the proposed algorithms will be evaluated and ranked according to a predefined list of requirements. The evaluation metrics to the used in this study are:

1. Execution speed.
2. Quality of the solution.
3. Solution reliability.
4. Strengths and weaknesses that govern their performance capacity to process the required load.

Through this analysis, the proposed hybrid algorithm is designed to utilized the strengths of the four algorithms and surpass their limitations to provide improvement in the execution rate and performance. The proposed algorithm is built upon achieving the ultimate equilibrium of exploration and exploitation that achieve solutions that are more precise without draining application's infrastructure resources.

This proposal is based on the following assumptions:

1. The application environment runs on devices with limited resources (memory, energy, processing power), which necessitates efficient algorithms.
2. The information is accurate and sufficient to support the algorithm's performance and ensure the reliability of the generated solutions.
3. The environment in intelligent systems remain relatively stable during the implementation period, without radical changes that may affect the algorithm's performance.
4. The algorithm is adaptable to minor changes in the operation environment without the need for complete reset of the basic parameters, making it more flexible and efficient for intelligent systems.

A. Greedy Randomized Adaptive Search Procedure (GRASP)

Greedy Random Adaptive Search (GRASP) algorithm is a metaheuristic algorithm that is based on a single-solution metaheuristics approach. In this algorithm, each iteration contains two main phases: a construction phase and a local search phase [19]. Utilizing these two distinct phases facilitates the attainment of an optimal equilibrium between exploration and exploitation, thereby yielding solutions of superior quality.

This particular algorithm was selected owing to its rapidity and efficacy in the optimization of solutions pertaining to large-scale issues [20]. The performance and efficacy of this algorithm can be further augmented through parameter calibration and modifications tailored to specific problem contexts.

Algorithm 1: Greedy Randomized Adaptive Search Procedure (GRASP)

---

1. **Initialization Parameters:** Set the number of iterations and the size of the candidate list.
2. **Repeat** until the maximum number of iterations is reached:

a. **Construct a Solution:** Use the Greedy Randomized Construction (GRC) phase.

b. **Improve the Solution:** Enhance the constructed solution with a Local Search (LS) phase.

c. **Evaluate and Update:** Evaluate the improved solution and update the best solution found.

3. **Greedy Randomized Construction (GRC) Phase:**

a. Construct a solution iteratively by greedily selecting candidate elements based on a randomized selection mechanism.

b. Randomize the selection process to introduce diversity and prevent premature convergence.

c. Employ a local criterion or heuristic to guide the selection of elements.

4. **Local Search (LS) Phase:**

a. Apply a local search procedure.

b. Explore the neighborhood of the current solution to find better solutions.

c. Apply techniques like 2-opt, swap, or insertion to traverse different neighborhoods.

5. **Termination:** End the algorithm once a stopping condition is fulfilled (a maximum number of iterations reached).

6. **Output:** Return the best solution obtained.

The exploitation stage is focused on building the solution through a deterministic greedy approach where the potential elements are selected according to a greedy approach toward short-term improvement in solution quality. The exploration step introduces randomness to the solution construction process, so the search is more diversified which increase the possibility of generating better solutions. This method facilitates a large search space while reducing the probability of being caught in local optimum solutions.

B. Firefly Algorithm (FA)

The Firefly Algorithm (FA) is a nature-inspired metaheuristic algorithm derived from the swarm intelligence of fireflies that emit flashes of light to attract fireflies. FA is used in optimization problems by simulating this process and attempting to achieve optimal solutions. The algorithm efficiency relies on the parameters like attractiveness of fireflies and the absorption coefficient, with higher brightness (better solutions) fireflies attracting other fireflies, which enables converging of the solutions towards optimal values [21].

Algorithm 2: Firefly Algorithm (FA)

---

1. **Initialize** the fireflies randomly in the solution space.

2. **Evaluate** each firefly's brightness.

3. **Move Fireflies**:

a. A firefly ***i*** is attracted to another firefly ***j*** if $f(x_j)$ is better than $f(x_i)$.

b. The attractiveness β between two fireflies is given by Eq. 1:

$$\beta = \beta_0 e^{-\gamma d(x_i,x_j)^2} \quad (1)$$

where: β is the brightness of the firefly, β_0 is the initial brightness, γ is the light absorption coefficient that controls the rate of decay, $d(x_i,x_j)$ is the distance between the two solutions x_i and x_j.

c. **Continuous FA Movement,** Eq. 2:

$$x_i^{t+1} = x_i^t + \beta\left(x_j^t - x_i^t\right) + \alpha\epsilon \quad (2)$$

where: x_i^{t+1} is position of firefly i at time t, β is attractiveness between fireflies, α is random perturbation, ϵ is uniformly distributed random number in $[-1,1]$.

d. **Discrete FA**: Instead of moving in Euclidean space, swap positions probabilistically:

- Swap two elements in the permutation based on attractiveness.
- Perform random small mutations to maintain diversity.

4. **Update** brightness after movement.

5. **Repeat** steps until the termination condition is met.

6. **Return** the best firefly as the best solution.

The exploitation step involves moving fireflies toward brighter locations in the search space. Brightness represents the fireflies' fitness, indicating how close they are to optimal solutions. Fireflies with higher brightness attract others more strongly, directing the population toward areas of the search space with potentially better solutions. The exploration step introduces randomness into the fireflies' movement to explore different areas of the search space. This randomness allows the fireflies to move away from their current positions, which can lead to the discovery of new solutions that can be more optimal than the current solutions.

C. Vortex Search (VS) Algorithm

Vortex Search (VS) algorithm is a vortices' motion-based optimization algorithm in space [22]. It is a population-based metaheuristic algorithm. The algorithm generates a population of initial solutions (vortices) by using heuristic or random methods, and then explores the search space by triggering perturbation and attraction mechanisms. The solutions are conditioned based on the motion of the vortices as guided by the best solution achieved, simulating the behavior of vortices in hydrodynamics. The algorithm prefers to have efficient exploration in the search space with attraction and repulsion dynamics to achieve a balance of global exploration and exploitation at the local level.

Algorithm 3: Vortex Search (VS) Algorithm

---

1. **Inputs:**

a. Initial center (μ_0) calculated using Eq. 3:

$$\mu^0 = \frac{(upperlimit + lowerlimit)}{2} \quad (3)$$

b. Initial radius (r_0) (or standard deviation (σ_0)) computed using Eq. 4:

$$\sigma^0 = \frac{\left((upperlimit) - (lowerlimit)\right)}{2} \quad (4)$$

c. t = 0

2. **Repeat**

a. **Generate Candidate Solutions:** Generate candidate solutions ($C_t(s)$) based on a Gaussian distribution around the center (μ_t) with standard deviation (radius) r_t.

b. **Boundary Handling:** If any candidate solution exceeds the search limits (upperlimit, lowerlimit), push the values into the limits based on the following condition:

$$s'_k = \{ rand(upperlimit - lowerlimit) + lowerlimit, \; s_k < lowerlimit \; s_k, \quad lowerlimit \leq s_k \leq upperlimit \; rand(upperlimit - lowerlimit) + lowerlimit, \; s_k > upperlimit \}$$

where: s_k is the k^{th} variable of the solution.

c. **Select Best Solution:** Select (s*) from candidate solutions ($C_t(s)$) based on their fitness values.

d. **Update Best Solution Found:** If $f(s^*) < f(s_{best})$

- $s_{best} = s^*$
- $f(s_{best}) = f(s^*)$

e. **Update Center:** Shift the center to the best solution found so far: $\mu_{t+1} = s_{best}$

f. **Decrease Radius:** Decrease the radius (or standard deviation) for the next iteration:

- $r_{t+1} = \text{Decrease}(r_t)$

where $\text{Decrease}(r_t)$ function decides how the radius is decreased between iterations. This is one key step in the algorithm and can be achieved with different approaches (e.g., linear decrease, exponential decrease).

g. t = t + 1

3. **Until** maximum number of iterations.

4. **Output:** Best solution found (s_{best}).

The VS algorithm achieves a favorable trade-off between exploration and exploitation. The algorithm starts with a broad search range, and it can sweep across a wide zone of the search space. As the range is repeatedly reduced, the algorithm will exploit ever-smaller regions around the currently best-known solution, enabling it to identify

the best-promising regions. The VS algorithm generates candidate solutions using a Gaussian distribution, allowing it to perform a probabilistic search. This search is centered on the current best-known solution, exploring at varying distances from it.

D. Search Group Algorithm (SGA)

SGA is an optimization method that was created to resolve complex engineering problems. The algorithm aims to achieve optimum or near-optimum solutions with a compromise between exploration and exploitation of the design space. The main goal of the algorithm is to generate improved design solutions compared to other heuristic algorithms at the same computational expense [23].

The algorithm begins by looking for fruitful regions in the design space, and then specializes in optimizing the most promising solutions within these regions. This is accomplished through a dynamic process whereby the algorithm adapts its search strategies as it progresses. The algorithm has been successfully applied to many problems such as truss structure optimization, as well as solving difficult problems like topology optimization, natural frequency constraints, and combining continuous and discrete variables in design.

Algorithm 4: Search Group Algorithm (SGA)

1. **Initialization:** Iteration counter k = 0, maximum number of iterations it_{max} , number of iterations before switching to global search it_{max_global}, initial perturbation size α_k, minimum perturbation size α_{min} , perturbation reduction factor b , population size N_{pop} , search group size n_g , number of individuals to mutate N_{mut} , tournament size for selection h , parameter controlling new individual generation distance t , random variable for mutation ε , random variable for family generation v.
2. **Create** a population (P) of N_{pop} individuals.
3. **Select** n_g individuals from the initial population (P) to form the initial search group (R_k). By use tournament selection with tournament size h.
4. **Replace** Individuals by New Members

a. Create new members by mutate N_{mut} individuals in the population (P). using the following equation Eq. 5:

$$x_{mut_j} = E(R_j) + t * \varepsilon * \sigma(R_j), \ for\, j = 1, \dots, n \quad (5)$$

where: x_{mut_j} is the j^{th} variable of the mutated individual. $E(R_j)$ is the mean value of the j^{th} column of the search group matrix (R_k). $\sigma(R_j)$ is the standard deviation of the j^{th} column of the search group matrix (R_k). ε is a random variable. t is a parameter controlling the search distance.

b. A member's likelihood of being replaced is determined by its rank within the current search group. Lower-ranked individuals face a higher probability of replacement.

5. **Generate** families (F_i) using the following equation Eq. 6:

$$x_{new} = R_{ij} + \alpha * v, for\, j = 1, \dots, n \quad (6)$$

where: x_{new} is a new individual generated for the family. R_{ij} is an individual from the search group (R_k). α is the perturbation size. v is a random variable.

6. **Select New Search Group (R_k +1)**

a. **If** $k < it_{max\_global}$: The new search group (R_k +1) is formed by selecting the best member from each family (F_i).

b. **Otherwise:** The new search group (R_k +1) is formed by selecting the best n_g individuals from the entire population (P).

7. **Update** the perturbation size using the following equation Eq. 7:

$$\alpha_k + 1 = b * \alpha_k \quad (7)$$

where: b is the perturbation reduction factor (a parameter of the SGA).

8. **Iteration and Termination**

a. k = k + 1.

b. **If** $k \leq it_{max}$: Go to Step (4) Replace Individuals by New Members.

9. **Return** best solution found (R_1).

The strengths of the SGA lie in its ability to find a delicate trade-off between the broad exploration at the beginning and the focused exploitation towards the termination. In this way, the algorithm can avoid local optimization and possesses very good flexibility in coping with complex problems. In addition, the diversity of solutions among generations increases the chances of obtaining near-optimal or even optimal solutions.

E. Proposed Fast Hybrid Metaheuristic Algorithm

This research aims to develop an efficient hybrid algorithm to improve the performance of intelligent applications in IoT systems, focusing on optimization problems with efficiently finding optimal solutions. We propose a new hybrid algorithm that combines the advantages of four algorithms, aiming to improve performance in IoT systems. This computational methodology is referred to as the Fast Hybrid Adaptive Search Algorithm (FHASA). This algorithm presents a sophisticated hybrid methodology that effectively balances exploration and exploitation, consequently enhancing performance through diverse strategies. Hence, it is meticulously crafted for intelligent computing applications that necessitate efficient and lightweight solutions.

The algorithm employs a variety of performance enhancement techniques in the process of problem solving. It initiates with the generation of initial solutions at random through a method known as Greedy Randomized Adaptive Search Procedure (GRASP), followed by perturbation of these solutions via a gravity-based perturbation mechanism (FA), wherein the gravitational influence is contingent upon the solution's quality. Subsequently, the most optimal solutions are identified through a tournament selection process utilizing the Simple Genetic Algorithm (SGA), aimed at extracting the most effective solutions from the existing population. The magnitude of the alterations is fine-tuned through radius adjustment (VS), which governs the degree of exploration over time. Ultimately, the solutions are refined through a local search technique (2-opt), capitalizing on localized information to enhance the outcomes.

Algorithm 5: Fast Hybrid Adaptive Search Algorithm (FHASA)

---

1. **Input** set of points, n_{Pop}, n_G, $max_{Iterations}$, initial alpha α, beta β, $it_{GlobalMax}$
2. **Compute** the distance matrix between the points
3. **Generate** the initial population using GRASP
4. **Select** the search group from the population using Tournament Selection
5. **Set** the initial radius, by Eq. 8:

$$radius = 0.5 * (number\ of\ points) \quad (8)$$

6. **For** iter = 0 to $max_{Iterations}$ do:

a. **For** each solution in the search group:

- **Apply** FA Perturbation using Eq. 9:

$$A = exp\ exp\left(-\beta * (totalDistance)\right) \quad (9)$$

- **Perform** (α * radius) random swaps

b. **Update** the search group:

c. **Apply** local search (2-opt) on the best solution from the population

- If (iter < $it_{GlobalMax}$), select the best n_G solutions from the search group
- Else, select from the entire population

d. **Merge** the improved solutions with the population and re-order, retaining the best n_{Pop} solutions

e. **Update**:

- radius = radius * β * (1 - iter/ $max_{Iterations}$)
- α = α * β

f. **Update** the global best solution

2. **Return**: the best solution found

The random permutation technique in the FA algorithm provides a diversity in search by swapping some locations based on the gravity function, which enhances exploration and overcomes local optimum. The number of random

permutations is determined based on the product of α and radius, which contributes to diversifying solutions and discovering new regions in the solution space.

Local search (2-opt) enhances solution optimization by exploiting the local structure of the solution, which improves the ordering of points and increases the accuracy of the results.

Tournament selection focuses on high-performance solutions, increasing the likelihood of incremental improvement through selecting the most likely candidates to proceed to further iterations. The algorithm employs multiple strategies to achieve an efficient balance between exploration and exploitation, improving efficiency through using the stored distance matrix to keep computational overhead at a minimum. Adjusting parameters such as α and radius also enhances performance with continued iterations. This mixed strategy amalgamates various methods to bring a total solution, improving the efficacy of the algorithm in applications related to intelligent computing. Figure 1 illustrates the steps of the proposed algorithm.

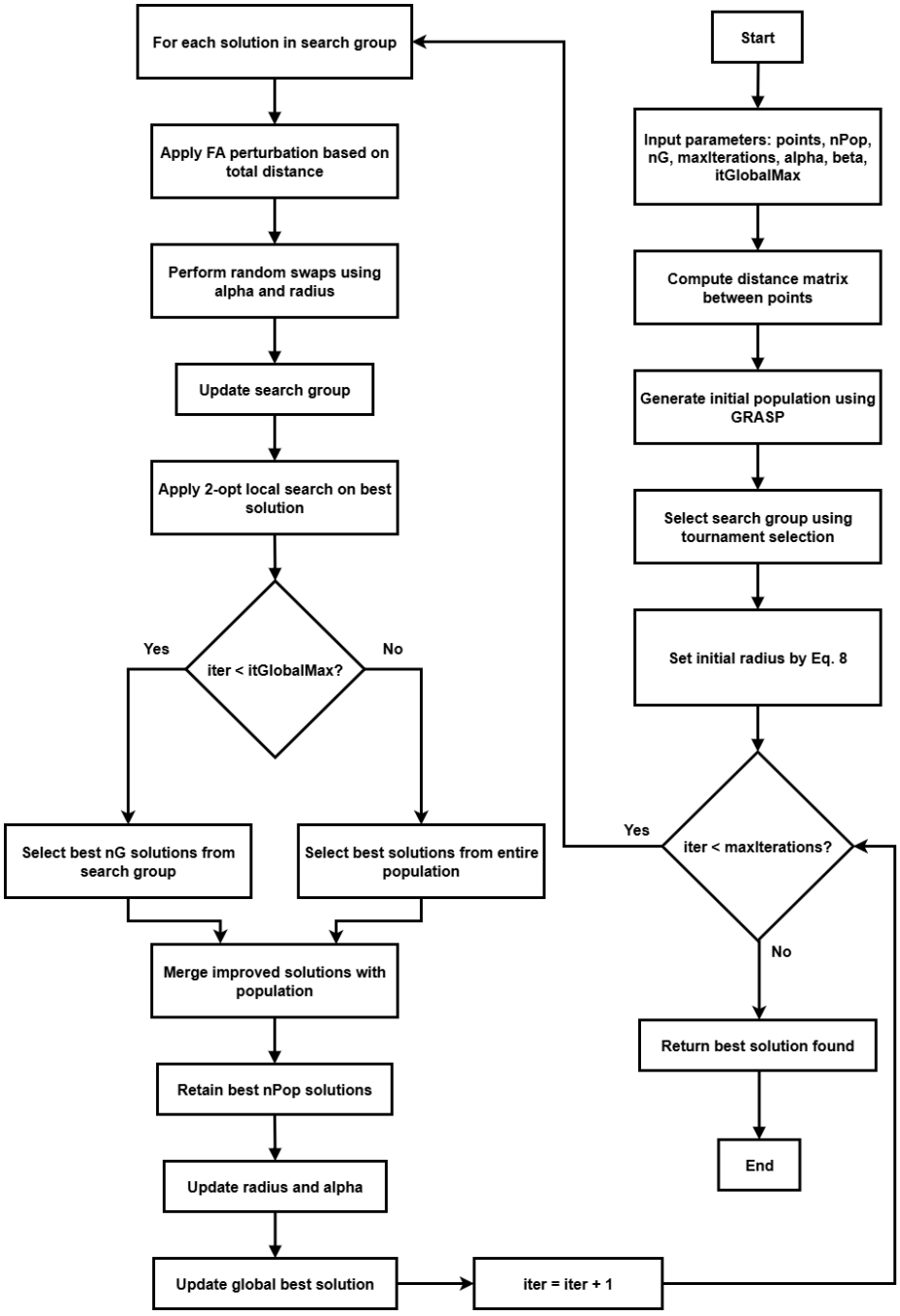


Figure 1. Flowchart of the proposed algorithm

4. Results and Discussion

Dataset

The Orienteering Problem (OP) dataset is used to test the efficiency of the algorithm presented in this work. It is one of the most popular benchmarks for path optimization and optimal planning [24][25][26]. The dataset provides realistic instances used to test the performance of metaheuristic algorithms to offer near-optimal solutions efficiently and in a reasonable time. With this dataset, it is possible to test the ability of the algorithm to find a balance between precision and execution speed compared to other algorithms.

Evaluation Methodology

In this paper, the performance metrics shall be used to verify the effectiveness of the proposed algorithm in promoting efficiency in intelligent applications. It endeavors to achieve a balance between accuracy and processing time while optimizing resource utilization. Special focus will be given to two major measures: Cost (shortest path), which evaluates the ability of the algorithm to reduce the overall cost while performing the tasks, and Speed (execution time), which measures the time spent by the algorithm in finding optimal or near-optimal solutions. Based on these two indicators, the performance of the algorithm will be compared to other traditional algorithms with an emphasis on the ability of rapid response without compromising solution quality.

There is a necessity to use multi-objective optimization to balance such conflicting goals. Multiple goals are dealt with by allocating various weights for every goal depending on its significance and impact on the system. To address this issue, the popular Weighted Sum Method is employed, wherein several objectives are aggregated into one objective function by employing personalized weights that represent the relative importance of each objective.

If we have n objectives to optimize, and $f_1(x)$, $f_2(x)$, ..., $f_n(x)$ are the functions representing the different objectives, the composite objective function can be represented as follows, Eq. 10:

$$f(x) = w_1 * f_{1(x)} + w_2 * f_{2(x)} + \ldots + w_n * f_{n(x)} \qquad (10)$$

where: $f(x)$ is the aggregated objective function. $f_i(x)$ is the individual objective function. w_i is the weight assigned to each objective.

In this paper, a list of crucial parameters that control the performance of the suggested hybrid algorithm is found, and the parameters are well adjusted to obtain an acceptable trade-off between exploration and exploitation. To illustrate the values used, Tables 1, 2, 3, 4, and 5 contain the values of the parameters used.

Table 1: Parameters setting for the GRASP algorithm

Parameter	**Value**
Maximum number of iterations	100
alpha	0.5

Table 2: Parameters setting for the FA algorithm

Parameter	**Value**
Number of fireflies	50
Max generations	100
alpha	0.2
beta	1.0

Table 3: Parameters setting for the VS algorithm

Parameter	**Value**
Number of search agents	50
Maximum number of iterations	100
alpha	0.2
beta	1.0

Table 4: Parameters setting for the SGA algorithm

Parameter	**Value**
Number of search agents	50
Maximum number of iterations	100
alpha	0.2
beta	1.0

Table 5: Parameters setting for the FHASA algorithm

Parameter	**Value**
Maximum number of iterations	100
alpha (Initial expansion coefficient for random permutations)	0.5
beta (Deterioration coefficient and gravity coefficient)	0.95
Iterations before switching to exploitation	50
Tournament Size	3

The algorithms were implemented on a computer with CPU Core (TM) i7-9750H, 2.60 GHz, and 16 GB RAM. The experimental results of the proposed algorithm, shown in Table 6, were obtained. The results in the table represent the average cost (distance) for the thirty times implemented, and the average execution time for run (speed).

Table 6: Results of implementing the proposed algorithm compared to other algorithms

Instance	GRASP		FA		VS		SGA		FHASA	
	AVG Cost	AVG Time	AVG Cost	AVG Time	AVG Cost	AVG Time	AVG Cost	AVG Time	AVG Cost	AVG Time
set_64_1_15	326.9	91.4	332.5	166.9	255.5	289.1	317.5	**7.1**	**94.4**	69.3
set_64_1_20	324.9	90.2	333.9	165.8	254.2	287	314.8	**6**	**95.1**	63.7
set_64_1_45	327.1	89.3	330.7	164.5	252.4	287.8	317	**6.5**	**95.2**	63.3
set_64_1_50	325.1	90.1	331.8	164	253.7	283.6	316.5	**6.3**	**94.8**	63
set_64_1_75	326.5	98.2	331.4	163.5	256	278.5	316.1	**6.1**	**95.5**	63.3
set_64_1_80	328.9	89.8	328.5	164.8	256	279.1	315	**5.8**	**95.9**	62.3
set_66_1_005	469.6	98.1	475.5	174	369.4	284.2	456.9	**9.5**	**134.9**	67.8
set_66_1_010	468.1	97.5	479.6	169.9	367.1	285.8	454.5	**11.6**	**135.9**	68.2
set_66_1_065	471.7	98.1	476.8	170.4	365.8	293.4	459	**6.0**	**135.3**	68
set_66_1_070	466.5	97.4	477.3	173.8	366.2	286.7	457.6	**5.7**	**135.2**	68.8
set_66_1_125	471.7	97.8	479.3	170.7	361.4	286	453.8	**5.9**	**135.3**	68.2
set_66_1_130	468.3	97.6	473.3	170.2	366.8	285.1	456.2	**5.6**	**135.8**	67.6
tsiligirides_problem_1_budget_05	214.6	12.1	217.9	81.7	153.3	109.7	202.4	**3.1**	**84.9**	9.9
tsiligirides_problem_1_budget_10	208.9	11.9	216.5	82.7	152.1	111.8	201.9	**3**	**85**	9.7
tsiligirides_problem_1_budget_46	208.9	11.8	218.2	82.2	151.8	109.8	203.3	**3.1**	**85.3**	10

Instance	GRASP		FA		VS		SGA		FHASA	
	AVG Cost	AVG Time	AVG Cost	AVG Time	AVG Cost	AVG Time	AVG Cost	AVG Time	AVG Cost	AVG Time
tsiligirides_problem_1_budget_50	210.8	11.5	215.6	81.9	151	108.5	203	**3.1**	**85.8**	9.8
tsiligirides_problem_1_budget_80	211.1	11.7	216.2	81.3	153.3	112.8	199.6	**2.9**	**85.1**	9.9
tsiligirides_problem_1_budget_85	211.6	11.8	215.4	82.2	151.9	108.9	200.6	**2.9**	**85.6**	9.9
tsiligirides_problem_2_budget_15	94.7	3.5	97.8	55	60.7	59.7	88.1	**2.1**	**43.8**	3.9
tsiligirides_problem_2_budget_20	94.6	3.5	96.1	55.1	59.9	58.4	89.3	**2.1**	**44.5**	3.8
tsiligirides_problem_2_budget_40	93.2	3.4	97.7	55.2	60.7	58.4	88.3	**2**	**45.1**	3.9
tsiligirides_problem_2_budget_45	94.1	3.4	98.2	55.3	60.4	59	86.7	**2.1**	**44.4**	4
tsiligirides_problem_3_budget_015	244.9	12.5	249.6	84.2	157.6	130.9	230	**3**	**99.4**	10.6
tsiligirides_problem_3_budget_020	244.2	12.5	247.1	84.3	162.8	121.1	232.8	**3.1**	**97.9**	10.8
tsiligirides_problem_3_budget_060	241.1	12.4	243.6	84.3	158.2	116.7	231.8	**3.1**	**98.8**	11.5
tsiligirides_problem_3_budget_065	241.9	12.5	243.6	90.5	163.4	117	227.9	**3.1**	**97.7**	10.8
tsiligirides_problem_3_budget_105	245.3	12.6	247.7	85.3	159.6	116.6	232.3	**3.1**	**99.2**	10.8
tsiligirides_problem_3_budget_110	245.2	12.4	249	85	159.6	114.3	232.2	**3.1**	**99.4**	10.7

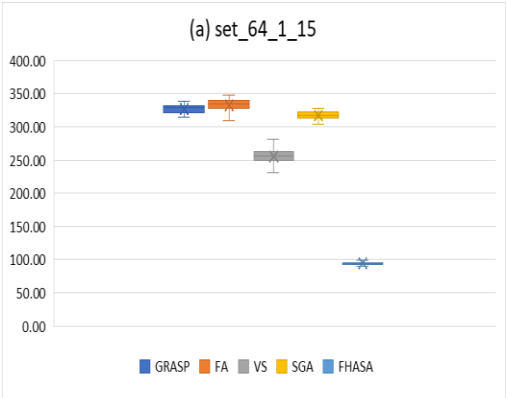
We notice that the proposed algorithm FHASA outperforms other algorithms in terms of the final cost rate. In term of speed, FHASA outperform another algorithm except SGA. The proposed algorithm FHASA comes in second place in terms of speed. To measure the required goal in this research, the Weighted Sum equation is implemented because there is more than one sub-goal. Where the two sub-goals (cost and speed) were considered equally important. After the normalization process and applying the equation, we have the results shown in Table 7, where the value of (P) is the result of applying the Weighted Sum equation.

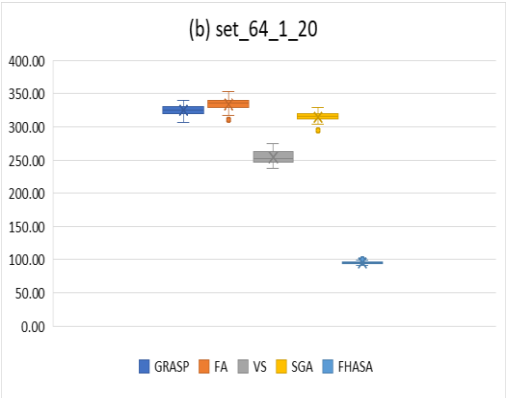
Table 7: Multi-objective implementation of the comparison between the two algorithms, FHASA and SGA

Instance	SGA			FHASA		
	cost_norm	time_norm	P	cost_norm	time_norm	P
set_64_1_15	0.87	0.00	0.87	0.02	0.01	**0.03**
set_64_1_20	0.86	0.00	0.86	0.02	0.01	**0.04**
set_64_1_45	0.88	0.00	0.88	0.02	0.01	**0.04**
set_64_1_50	0.86	0.00	0.86	0.02	0.01	**0.03**
set_64_1_75	0.87	0.00	0.87	0.02	0.01	**0.04**
set_64_1_80	0.89	0.00	0.89	0.03	0.01	**0.04**

Instance	SGA			FHASA		
	cost_norm	time_norm	P	cost_norm	time_norm	P
set_66_1_005	0.88	0.00	0.88	0.02	0.01	**0.04**
set_66_1_010	0.87	0.00	0.87	0.03	0.01	**0.04**
set_66_1_065	0.89	0.00	0.89	0.02	0.01	**0.04**
set_66_1_070	0.89	0.00	0.89	0.02	0.01	**0.04**
set_66_1_125	0.88	0.00	0.88	0.02	0.01	**0.04**
set_66_1_130	0.87	0.00	0.87	0.02	0.01	**0.04**
tsiligirides_problem_1_budget_05	0.79	0.00	0.80	0.05	0.01	**0.07**
tsiligirides_problem_1_budget_10	0.80	0.00	0.80	0.05	0.01	**0.06**
tsiligirides_problem_1_budget_46	0.81	0.00	0.81	0.06	0.01	**0.07**
tsiligirides_problem_1_budget_50	0.81	0.00	0.81	0.06	0.01	**0.07**
tsiligirides_problem_1_budget_80	0.79	0.00	0.79	0.06	0.01	**0.07**
tsiligirides_problem_1_budget_85	0.78	0.00	0.78	0.06	0.01	**0.07**
tsiligirides_problem_2_budget_15	0.70	0.01	0.71	0.06	0.02	**0.08**
tsiligirides_problem_2_budget_20	0.72	0.00	0.72	0.07	0.01	**0.09**
tsiligirides_problem_2_budget_40	0.74	0.00	0.74	0.09	0.01	**0.10**
tsiligirides_problem_2_budget_45	0.71	0.00	0.71	0.08	0.01	**0.09**
tsiligirides_problem_3_budget_015	0.77	0.00	0.77	0.07	0.01	**0.08**
tsiligirides_problem_3_budget_020	0.81	0.00	0.81	0.06	0.01	**0.07**
tsiligirides_problem_3_budget_060	0.81	0.00	0.82	0.07	0.02	**0.08**
tsiligirides_problem_3_budget_065	0.78	0.00	0.78	0.06	0.01	**0.07**
tsiligirides_problem_3_budget_105	0.81	0.00	0.81	0.07	0.01	**0.08**
tsiligirides_problem_3_budget_110	0.82	0.00	0.82	0.07	0.01	**0.08**

The results of the two algorithms (FHASA and SGA) were compared only, where the first one is superior in cost and the second one is superior in speed. The results proved the superiority of the proposed algorithm, which proves the possible advantage of using it in smart applications or others. The plots in Figure 2 represent some comparisons between the implemented algorithms.





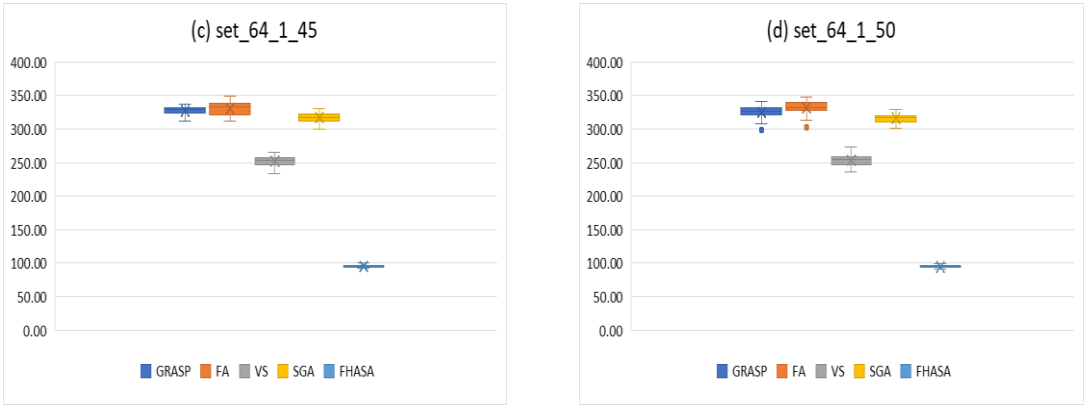


Figure 2. Comparing the proposed algorithm with other algorithms in some instances

5. Conclusion

This paper presents a fast hybrid algorithm that aims to improve the efficiency of intelligent applications in IoT. By integrating the advantages of multiple algorithms, we were able to achieve an optimal balance between time performance and solution accuracy, which contributed to improve the efficiency of optimization algorithm in resource-constrained environments. Experimental results demonstrate the effectiveness of the proposed algorithm in reducing cost and increasing speed compared to state-of-art algorithms, making it suitable for use in intelligent applications that require fast response and resource efficiency. Future work focuses on exploring the possibilities of using reinforcement learning techniques to improve the efficiency of search and exploration in the algorithm.

References

[1] A. Refaat, A. Elbaz, A. E. Khalifa, M. M. Elsakka, A. Kalas, and M. H. Elfar, "Performance evaluation of a novel self-tuning particle swarm optimization algorithm-based maximum power point tracker for porton exchange membrane fuel cells under different operating conditions," *Energy Convers. Manage*, vol. 301, p. 118014, 2024.

[2] W. Zhang, X. Gu, L. Tang, Y. Yin, D. Liu, and Y. Zhang, "Application of machine learning, deep learning and optimization algorithms in geoengineering and geoscience: Comprehensive review and future challenge," *Gondwana Res.*, vol. 109, pp. 1-17, 2022.

[3] L. Kong, J. Tan, J. Huang, G. Chen, S. Wang, X. Jin, and S. K. Das, "Edge-computing-driven internet of things: A survey," *ACM Comput. Surv.*, vol. 55, no. 8, pp. 1-41, 2022.

[4] F. C. Andriulo, M. Fiore, M. Mongiello, E. Traversa, and V. Zizzo, "Edge computing and cloud computing for internet of things: A review," *Informatics*, vol. 11, no. 4, p. 71, Sep. 2024.

[5] A. A. Jihad, S. T. F. Al-Janabi, and E. T. Yassen, "Enhanced iterated local search for scheduling of scientific workflows," in *2021 14th Int. Conf. Develop. eSyst. Eng. (DeSE)*, Dec. 2021, pp. 335-339.

[6] O. Aouedi, T. H. Vu, A. Sacco, D. C. Nguyen, K. Piamrat, G. Marchetto, and Q. V. Pham, "A survey on intelligent Internet of Things: Applications, security, privacy, and future directions," *IEEE Commun. Surv. Tutor*, 2024.

[7] A. Zanella, N. Bui, A. Castellani, L. Vangelista, and M. Zorzi, "Internet of things for smart cities," *IEEE Internet Things J.*, vol. 1, no. 1, pp. 22-32, 2014.

[8] S. Zhu, T. Yu, T. Xu, H. Chen, S. Dustdar, S. Gigan, and Y. Pan, "Intelligent computing: the latest advances, challenges, and future," *Intell. Comput*, vol. 2, p. 0006, 2023.

[9] S. Avasthi, S. Garg, S. L. Tripathi, and R. Chauhan, "Metaheuristics algorithms: Fundamental aspects and applications in optimization problems," in *Metaheuristics-Based Materials Optimiz.* Woodhead Publishing, 2025, pp. 3-24.

[10] S. A. Jassim, A. K. Farhan, and A. H. Radie, "Using a Hybrid Pseudorandom number generator for cryptography in the internet of things," in *2021 4th Int. Iraqi Conf. Eng. Technol. Appl. (IICETA)*, Sep. 2021, pp. 264-269.

[11] A. A. Jihad, S. T. F. Al-Janabi, and E. T. Yassen, "A survey on provisioning and scheduling algorithms for scientific workflows in cloud computing," in *AIP Conf. Proc.*, vol. 2400, no. 1, Oct. 2022.

[12] M. Gendreau and J. Y. Potvin, Eds., *Handbook of Metaheuristics*, 2nd ed. New York: Springer, 2010, p. 9.

[13] S. Khalfi, G. Iacca, and A. Draa, "On the use of single non-uniform mutation in lightweight metaheuristics," *Soft Comput.*, vol. 26, no. 5, pp. 2259-2275, 2022.

[14] V. Punnathanam and P. Kotecha, "Yin-Yang-pair Optimization: A novel lightweight optimization algorithm," *Eng. Appl. Artif. Intell.*, vol. 54, pp. 62-79, 2016.

[15] H. Sabireen and N. Venkataraman, "A hybrid and light weight metaheuristic approach with clustering for multi-objective resource scheduling and application placement in fog environment," *Expert Syst. Appl.*, vol. 223, p. 119895, 2023.

[16] Y. Zheng, R. Sun, Y. Liu, Y. Wang, R. Song, and Y. Li, "A Hybridization Grey Wolf Optimizer to Identify Parameters of Helical Hydraulic Rotary Actuator," *Actuators*, vol. 12, no. 6, p. 220, May 2023.

[17] S. Al-Otaibi, A. Al-Rasheed, R. F. Mansour, E. Yang, G. P. Joshi, and W. Cho, "Hybridization of metaheuristic algorithm for dynamic cluster-based routing protocol in wireless sensor Networks," *IEEE Access*, vol. 9, pp. 83751-83761, 2021.

[18] D. V. Lyridis, "An improved ant colony optimization algorithm for unmanned surface vehicle local path planning with multi-modality constraints," *Ocean Eng.*, vol. 241, p. 109890, 2021.

[19] T. A. Feo and M. G. Resende, "Greedy randomized adaptive search procedures," *J. Global Optim.*, vol. 6, pp. 109-133, 1995.

[20] R. Salimi, S. Azizi, and J. Abawajy, "A greedy randomized adaptive search procedure for scheduling IoT tasks in virtualized fog–cloud computing," *Trans. Emerg. Telecommun. Technol.*, vol. 35, no. 5, p. e4980, 2024.

[21] I. Fister, I. Fister Jr, X. S. Yang, and J. Brest, "A comprehensive review of firefly algorithms," *Swarm Evol. Comput*, vol. 13, pp. 34-46, 2013.

[22] B. Doğan and T. Ölmez, "A new metaheuristic for numerical function optimization: Vortex Search algorithm," *Inf. Sci.*, vol. 293, pp. 125-145, 2015.

[23] M. S. Gonçalves, R. H. Lopez, and L. F. F. Miguel, "Search group algorithm: a new metaheuristic method for the optimization of truss structures," *Comput. Struct*, vol. 153, pp. 165-184, 2015.

[24] I. M. Chao, "Algorithms and solutions to multi-level vehicle routing problems," Ph.D. dissertation, Univ. Maryland, College Park, MD, USA, 1993.

[25] I. M. Chao, B. L. Golden, and E. A. Wasil, "A fast and effective heuristic for the orienteering problem," *Eur. J. Oper. Res.*, vol. 88, no. 3, pp. 475-489, 1996.

[26] M. Fischetti, J. J. S. Gonzalez, and P. Toth, "Solving the orienteering problem through branch-and-cut," *INFORMS J. Comput*, vol. 10, no. 2, pp. 133-148, 1998.