Cluster Computing (2022) 25:1981–2005
https://doi.org/10.1007/s10586-021-03304-5

An efficient harris hawk optimization algorithm for solving the travelling salesman problem

Farhad Soleimanian Gharehchopogh[1] · Benyamin Abdollahzadeh[1]

Received: 16 January 2021 / Revised: 8 May 2021 / Accepted: 13 May 2021 / Published online: 31 May 2021

Abstract
Travelling Salesman Problem (TSP) is an Np-Hard problem, for which various solutions have been offered so far. Using the Harris Hawk Optimization (HHO) algorithm, this paper presented a new method that uses random-key encoding to generate a tour. This method helps maintain the main capabilities of the HHO algorithm, on the one hand, and to take advantage of the capabilities of active mechanisms in the continuous-valued problem space on the other hand. For the exploration phase, the DE/best/2 mutation mechanism employed in the exploitation phase, besides the main strategies in the HHO algorithm, was used. Ten neighborhood search operators are used, four of which are introduced. These operators were intelligently selected using the MCF. The Lin-Kernighan local search mechanism was utilized to improve the proposed algorithm's performance, and the Metropolis acceptance strategy was employed to escape the local optima trap. Besides, 80 datasets were evaluated in TSPLIB to demonstrate the performance and efficiency of the proposed algorithm. The results showed the excellent performance of the proposed algorithm.

Keywords Harris hawks optimization algorithm · Travelling salesman problem · Optimization · Neighbourhood search operator

1 Introduction

The TSP is one of the NP-hard problems that many research types have been done on this issue. However, none of the research has been able to find a particular solution to this problem. This problem is discrete from the hybrid optimization problem, and the goal is to find the shortest Hamiltonian path. So, look for a route that first visits all the cities at most once and secondly returns to the city where the route started from that city and also this route is also the shortest possible route. Although this problem has a simple mathematical model and is very easy to understand, it is challenging to solve it. Due to the enlargement of the issue and the increase of cities, computational complexity increases and requires more resources and computational time. Since it is one of the NP-hard problems, it is not solvable, and all the presented solutions are relative, and a new solution can be presented that is more efficient than the previous solutions.

Nowadays, extensive and diverse research has been done by researchers to solve various problems [1–5]. Also, Various methods have been proposed to solve the TSP problem, which are accurate methods suitable for small and medium optimization problems. Because in solving large-scale problems, the computational time is very long in general, some of the exact methods are: branch and bound [6], branch and cut [7], branch and price [8], cutting planes [9], and Lagrangian dual [10]. Moreover, the need to find reasonable (not necessarily optimal) solutions to these problems have led to various approximation algorithms, such as metaheuristics. These methods have advantages such as simplicity and flexibility and generally produce quality solutions in a reasonable amount of time by creating shortcuts. However, in some models based on metaheuristic algorithms, they either have a high computational time, or do not produce an acceptable optimal answer, or are inefficient in the face of large-scale problems [11–15].

✉ Farhad Soleimanian Gharehchopogh
bonab.farhad@gmail.com

Benyamin Abdollahzadeh
benyamin.abdolahzade@gmail.com

[1] Department of Computer Engineering, Urmia Branch, Islamic Azad University, Urmia, Iran

Different neighborhood search operators can be used to improve the performance of metaheuristic algorithms in metaheuristic algorithms. However, each of these operators has unique features that can have the necessary efficiency in different optimization operations stages. For this reason, hyper-heuristic modes of the Modified Choice Function can be used to select neighborhood search operators intelligently [16–18].

For the first time in [19], the term hyper-heuristic has been introduced, divided into two categories: selection hyper-heuristic and generation hyper-heuristic. Moreover, these two categories of selective and productive hyper-heuristics can be divided into two categories based on the nature of heuristics: perturbative and constructive, which is constructive hyper-heuristics [20] create a solution from the ground up gradually [21]. Moreover, in perturbative hyper-heuristics by performing disturbance mechanisms repeatedly in the solution improves the solution. Mechanisms that are generated or selected by hyper-heuristics are called low-level heuristics (LLHs). Selective-based hyper-heuristics have two general levels, low-level and high-level. The lower level includes evaluation function(s) and problem representation, and a set of LLHs.

Moreover, the high-level task of managing the LLH selection is used to generate a new solution and accept the new solution. The mechanism for selecting an LLH from a set of LLHs during the optimization process, in which this selected LLH performs better than other LLHs in this set, is called LLH selection, and several LLH selection methods include choice function [22–24], set of reinforcement learning variants [25, 26], backtracking search algorithm [27], harmony search [28], and Tabu search [29]. Moreover, how to decide to accept a new solution produced by LLH is called move acceptance. Some of the move acceptance methods are Late Acceptance [30], Simulated Annealing [31], Only Improvement [32].

In hyper-heuristics, there are two primary components called diversification and intensification. Due to the different capabilities of each LLH in different stages of the search process, these two components are essential hyper-heuristic elements. It is an intensification component to focus as much as possible on LLHs that perform better than other LLHs. On the other hand, it is a component of diversification to select LLHs that are rarely selected. Therefore, it is essential to strike a balance between diversification and intensification [27]. Furthermore, if an LLH performs well in one iteration step, it should not be used in subsequent steps alone to improve the solution, and if an LLH performs poorly in one iteration step, make this LLH not used permanently.

On the other hand, local search algorithms have been used in many studies to solve hybrid problems. Using two methods [29], Multi-start Local Search (MSLS) and Iterated Local Search (ILS) [30], the performance of local search algorithms is increasingly increasing. If these different mechanisms are combined, they can significantly increase metaheuristic algorithms' performance.

This paper presents a new method using the HHO algorithm [33]. This algorithm was developed to inspire the life, hunting practices, and mathematical modeling of Harris hawks' behaviors in the natural environment. This algorithm has been used in many studies despite being new, including Design and manufacturing problems, multi-level image thresholding problems [34], power flow problems [35], feature selection [11, 36–38], Satellite Image Segmentation [39], design of microchannel heat sinks [40], etc.

The proposed algorithm employs random-key encoding to generate a tour to maintain the core capabilities of the HHO algorithm, on the one hand, and utilizes the capabilities of active mechanisms in the continuous-valued problem space, on the other. Random-key encoding transfers solutions from continuous to discrete space. It motivated us to discretize the main mechanisms of the HHO algorithm operating in continuous space to solve the TSP, a discrete problem.

In this paper, to improve this algorithm's performance, two DE/best/2 mutation mechanisms have been used to increase the HHO algorithm's efficiency in the exploration phase. On the other hand, to improve the proposed algorithm's performance to solve the TSP, ten neighborhood search operators were used, four of which have been presented for the first time. Furthermore, the modified choice function (MCF) was utilized to increase efficiency and select neighborhood search operators. The proposed algorithm's capability and performance were then significantly increased using the Lin-Kernighan (LK) local search mechanism. Finally, the Metropolis acceptance strategy was employed to escape the local optima trap. The proposed algorithm's performance in solving problems with different dimensions was evaluated using the datasets available in TSPLIB [41], including small, medium, and large 100-85900 cities. The most important innovations and contributions of this paper are as follows:

- This paper presents the HHO algorithm for the first time to solve the TSP.
- The random-key encoding scheme is employed to adapt and solve the TSP using the HHO algorithm.
- A DE/best/2 mutation operator mechanism is utilized to improve the HHO algorithm's performance in the exploration phase.
- Ten neighborhood search operators are used for improving the proposed algorithm's performance, four of which are presented for the first time in this paper.

- Modified choice function (MCF) is utilized to select neighborhood search operators at different optimization operations stages intelligently.
- Lin-Kernighan (LK) local search is employed to increase the efficiency of the proposed algorithm.
- The Metropolis acceptance strategy is utilized to escape the local optima trap.
- The proposed algorithm's performance in solving TSP, which consists of 80 instances, has been tested using three criteria (i.e., the average tour length, average percent deviation, and average computation time). It is then compared with the previous seven models, indicating the acceptable performance of the proposed algorithm.
- The proposed algorithm is also compared and evaluated with other models using the Wilcoxon signed-rank test.

The rest of this paper is structured as follows. Section 2 examines related works, Section 3 addresses the basic concepts, Section 4 presents the proposed algorithm, and Section 5 compares and evaluates the proposed algorithm with the other methods presented. Finally, Section 6 concludes the paper.

2 Related works

Various researchers have proposed various methods and algorithms over the years to solve the TSP. However, a definite method or algorithm cannot be proposed for TSP because it is an NP-hard problem. Therefore, researchers have always attempted to try some methods more efficiently than previous ones [42, 43].

In [44], a master-slave-based method is used to solve the TSP, with several colonies cooperating periodically in a distributed computing environment (DCE). This algorithm is hybrid with 3-Opt to improve performance. Each colony runs this algorithm separately and shares the results with other colonies. This method is evaluated with 21 instances. These methods either have a high computation time or do not provide an acceptable optimal solution, or are inefficient in dealing with large-scale problems.

In [45], a hybrid simulated annealing algorithm based on a Symbiotic Organism Search (SOS) is proposed to solve TSP. The purpose of this work is to evaluate the convergence behavior and scalability of this hybrid algorithm to solve TSPs with small- and large-scale traveling. In terms of average execution time, experiments on the solution convergence and percentage deviations were evaluated. The results of the experiments showed an improvement in results.

In [46], a discrete algorithm, i.e., a comprehensive learning Particle Swarm Optimization (PSO) algorithm with the Metropolis acceptance criterion, is proposed to solve the TSP. This algorithm employs two strategies, namely lazy velocity and eager evaluation, to improve performance. Additionally, the Metropolis acceptance criterion is utilized to avoid premature convergence. To solve this problem, hyper-heuristic methods, e.g., Modified Choice Function (MCF), can be employed. In [17], it is done automatically and intelligently by selecting neighborhood search heuristic by employed and onlooker bees.

In [47], the wolf colony search algorithm, which exploits siege strategy, is employed to solve the TSP to achieve several goals, including improving the mining ability, reducing the besiege range, and accelerating convergence time. The proposed algorithm showed better performance in terms of higher solving accuracy and faster convergence speed. Moreover, travel behavior and calling behavior strategies have been exploited to enhance wolf interaction and improve global optimization accuracy.

Additionally, the Lin-Kernighan local search is integrated with this algorithm. This algorithm is evaluated with 64 instances of TSP in TSPLIB. Hyper-heuristic is an automated methodology for selecting or generating a set of heuristics [21]. Moreover, travel behavior and calling behavior strategies have been exploited to enhance wolf interaction and improve global optimization accuracy. Also, in [48], the discrete pigeon-inspired optimization (PIO) (DPIO) algorithm is presented. The Metropolis acceptance criterion is utilized for discretization. A map operator and a new compass operator with comprehensive learning capability have been employed in this algorithm to increase the DPIO's exploration ability. A new landmark operator has also been utilized to increase the exploitation of this algorithm. Thirty-three large-scale instances of TSPLIB with 1000-85900 cities have been tested to evaluate the DPIO algorithm's performance. In [49], a combined wolf pack search and local search solution is presented to solve the TSP to balance exploration and exploitation and not get stuck in a local optimum. The results of experiments on TSPLIB indicate that the results obtained by this algorithm are better and closer compared to other algorithms. This algorithm could compare theoretical optimal values with higher robustness compared to the obtained values.

Moreover, In [50], a new algorithm called the Anglerfish algorithm is proposed to solve TSP. The search operation is performed randomly based on the initial population using the randomized incremental construction technique. It is performed using random sampling and without complicated procedural procedures. Also, in [51], a discrete algorithm, i.e., discrete SOS (DSOS), is enhanced using excellence coefficients self-escape to solve the TSP, called ECSDSOS. The self-escape strategy is utilized to avoid getting stuck in the local optimum. The excellence

coefficients are used to choose shorter edges (routes) for generating better local paths. Instances in TSPLIB are used to evaluate this algorithm. The results show an improvement in the performance of the proposed algorithm relative to the compared algorithms.

In [52], the discrete sine-cosine algorithm (SCA) is presented using a local search to solve the TSP. The proposed algorithm employs two different mathematical models to update each generation's solutions to balance exploration and exploitation. It is also integrated with the 2-opt local search method to improve exploitation. This algorithm exploits a heuristic crossover to increase exploration ability. The experiments and evaluations indicate an improvement in this algorithm's performance from 41 different benchmarks in TSPLIB.

In [53], new versions of the ABC algorithm are introduced to solve the TSP, including the combinatorial version of the standard ABC, called combinatorial ABC (CABC), and an improved version of the CABC algorithm called the quick CABC (qCABC). The efficiency of these two versions of the ABC algorithm was evaluated using 15 TSP benchmarks. The results of evaluating these two algorithms' performance were then compared with eight different variables (GA variants).

In [54], a hybrid model is proposed using a genetic algorithm (GA) and remove-sharp and local-opt with ant colony optimization (ACO) to accelerate convergence and implement positive feedback to optimize the search space and create an efficient solution to solve complex problems. TSP was tested to evaluate the optimality accuracy and performance of the combinatorial model, indicating satisfactory performance. The results also showed better performance of the proposed algorithm relative to the compared models. This model can also solve various problems, such as network routing, scheduling, vehicle routing, etc. Also, in [55], an algorithm based on ACO and the partheno-genetic algorithm is proposed to solve the TSP. The primary purpose is to divide the variables into two parts. It uses the partheno-genetic algorithm to comprehensively search for the best value of the first section variables and then the ACO to precisely determine the best value of the second section variables. The comparative experiment results showed that the combinatorial algorithm effectively solved large-scale TSP and better performance than existing algorithms.

In [56], a new discrete differential evolution algorithm is used to solve TSP. The authors suggest a combination of the following: (1) an improved mapping mechanism for mapping continuous variables to discrete variables and vice versa, (2) a k-means clustering restoration method to increase the number of solutions in the initial population, and (3) a set of strategies mutation to increase the exploitation capability of the algorithm. Finally, two well-known local searches are presented to improve the local capability of the proposed algorithm. The experimental results showed a significant advantage of the proposed algorithm over many comparative methods in terms of the mean of known errors from known solutions; it achieves very competitive results with less computational time compared to other algorithms.

In [57], the ACO algorithm for solving TSP using a self-adaptive method to improve convergence and diversification called DEACO is proposed. It has been evaluated using the samples in TSPLIB, and the results of this work indicate the excellent performance of this model. And, In [58], a new discrete version of the Tree-Seed Algorithm (TSA) for solving TSP is presented. Three operators, swap, shift, and symmetry, have been used to provide the discrete version. This model has been evaluated and compared with several other discrete optimization algorithms, which indicate this model's acceptable performance.

In [59], the Genetic Algorithm is used to solve the multi-objective TSP problem. Numerous samples from TSPLIB with a different number of cities have been used for evaluation, and the results show the acceptable performance of this model. In [60], an improved PSO algorithm is used to solve the TSP, named MPSO. In this model, local search algorithms are also used to improve performance. Also, a new method has been used to move the particle towards the best particle for preventing premature convergence. The results obtained from this model have been compared with several other meta-heuristic algorithms, and the results obtained from this comparison have shown that it achieves better results than other optimization algorithms in most samples.

In [61], the authors present a discrete version of the shuffled frog-leaping algorithm based on heuristic information. In this model, a new operator called nearest neighbor information is designed. Also, four improved search strategies have been used to improve the performance of this model. Opposite roulette selection, on the other hand, is used to maintain population diversification. A large number of samples in TSPLIB have been used for evaluation. The results obtained from this work indicate the excellent performance of this model.

In [62], a new discrete version of the Crow Search Algorithm(CSA) for solving TSP is presented. Three discrete CSA are also proposed to improve performance. The algorithms presented in this paper are based on modular arithmetic, basic operators and dissimilar solutions techniques. One hundred eleven instances and several optimization algorithms have been used To evaluate and compare this model's performance, and the results have shown that it has a good performance in solving TSP. And, In [63], a discrete version of the Farmland Fertility Algorithm(FFA) is presented. In this research, three

Table 1 Comparison of optimization algorithms used to solve TSP

Name	Year	Authors	Algorıthm	Dısadvantage	Advantages
PACO-3Opt [44]	2016	Şaban Gülcü et al.	ACO	Poor results in samples with large dimensions and high execution time	Introducing a new model of ACO algorithm based on the master-slave paradigm
SOS-SA [45]	2017	Ezugwu et al.	SSO and Simulated Annealing	Poor performance on large samples	Provide a hybrid version to solve the TSP
D-CLPSO [46]	2018	Zhong et al.	PSO algorithm	Evaluation is with a small amount of data	Acceptable execution time
ABC-MCF [17]	2019	Choong et al.	Artificial bee colony (ABC) algorithm	Located in optimal local locations	Introduce new neighborhood search operators and intelligently use neighborhood search operators during optimization operations
DPIO [48]	2019	Zhong et al.	PIO	Low and limited evaluations and poor results in large samples	Provide a discrete version of the DPIO algorithm and use the Metropolis acceptance criterion mechanism
WPS-LS [49]	2019	Dong et al.	Wolf Pack Search	Low ratings with small samples	Provide a hybrid version and use the local search engine
AA [50]	2019	Pook et al.	Anglerfish algorithm	Low ratings with small samples	Introducing a new algorithm to solve the TSP
ECSDSOS [51]	2019	Wang et al.	SOS	Low ratings with small samples	Using mechanisms of excellence coefficients and self-escape strategy
DSCA [52]	2019	Tawhid et al.	sine-cosine algorithm	Poor performance in dealing with large samples	Introduce a new discrete version of the SCA algorithm and use the local search mechanism
qCABC [53]	2019	Karaboga et al.	ABC algorithm	As the dimensions increase, so does the samples of performance.	Introducing a new version of the ABC algorithm for solving hybrid problems
Nested Hybrid ACS [54]	2019	Sahana	genetic algorithm and ant colony	Evaluations are only for small samples	Provide a hybrid version and use multiple mechanisms
AC-PGA [55]	2020	Jiang et al.	genetic algorithm and ant colony	High complexity	Provide a hybrid model to solve the MTSP problem
NDDE [56]	2020	Ali et al.	differential evolution (DE) algorithm	Complex implementation and waste execution time	Introduce a new version of the DE algorithm and use the local search mechanism
DEACO [57]	2020	Ebadinezhad	ACO	Performs poorly in dealing with large issues.	Provide a self-adaptive version of the ACO algorithm to solve the TSP
DTSA [58]	2020	Cinar et al.	Tree-Seed algorithm	It only works well in solving small samples	Introducing a new discrete version of the TSA algorithm to solve the TSP
MOGA [59]	2020	George et al.	Genetic Algorithm	No comparison has been made with other methods	Solve TSP problem in multiple objectives
MPS [60]	2021	Yousefikhoshbakht	PSO	The performance of the algorithm in solving large-scale samples is poor	Provide an improved version and use the local search algorithm
DSFL [61]	2021	Huang et al.	shuffled frog-leaping algorithm	Hard implementation	Provide a discrete version of the SFL algorithm as well as the use of multiple mechanisms
DCSA [62]	2021	Al-Gaphari et al.	Crow search Algorithm	In some samples, it has shown poor results	Present a new discrete version of the CSA algorithm and evaluate on a large number of standard examples
DFFA [63]	2021	Abdollahzadeh et al.	Farmland Fertility Algorithm (FFA)	It has a lot of execution time in solving large samples.	Provide a new discrete version of FFA and use the local search mechanism and metropolis acceptance criterion

neighborhood search operators and a crossover operator are used. Also, to improve the performance of this model, the 2-OPT local search technique has been used. The roulette wheel technique has also been used to select local search operators. Examples from TSPLIB and several other optimization algorithms have been used to evaluate this model. The results show that this model performs well in other models.

Continuing previous research on the TSP solution, this paper offers a different approach to solving this problem by improving the HHO algorithm using various mechanisms that help balance the phases of exploration and exploitation, prevent premature convergence, and escape the local optima trap. It will be explained in detail in the section on the proposed algorithm (Table 1).

3 Fundamental research

3.1 Harris hawk optimization algorithm

The description of the HHO algorithm requires the symbols that are briefly described in Table 2. A brief description of this algorithm is given below.

The HHO algorithm uses two different strategies for search operations in the exploration phase. Each of these strategies is selected based on q; If q ≥ 0.5, the first strategy is used to search near one of the other hawks randomly, but if q < 0.5, the second strategy is used for the search operation, expressed in Eq. (1).

$$X(t+1)=\begin{cases} X_{rand}(t)-r_1|X_{rand}(t)-2_{r_2}X(t)|q\geq 0.5 \\ (\mathrm{X_{rabbit}}(t)-\mathrm{X_m}(t))-\mathrm{r_3}(LB+\mathrm{r_4}(UB-LB))q<0.5 \end{cases} \quad (1)$$

where $X_m(t)$ is calculated based on Eq. (2).

$$X_m(t)=\frac{1}{N}\sum_{i=1}^{N}X_i(t) \quad (2)$$

A different mechanism is used to move from the exploration phase to the exploitation phase. In optimization operations, the exploration operation is performed first, followed by the exploitation phase by increasing the iterations, finding the optimal solution, and promising solutions. Equation (3) is used for mathematical modeling.

$$E=2E_0\left(1-\frac{t}{T}\right) \quad (3)$$

If |E| ≥ 1, the algorithm enters the exploration phase, but if |E| < 1, it enters the exploitation phase. The value of E has a decreasing trend during the increase of iterations. The HHO algorithm uses four different strategies to perform optimization operations in the exploitation phase. If E ≥ 0.5, two strategies, besiege and soft besiege with progressive rapid dives, are used. Conversely, if E < 0.5, two strategies, besiege and hard besiege with progressive rapid dives, are used. Each of these strategies will be explained below.

3.2 Soft besiege

If r ≥ 0.5and|E| ≥ 0.5, the HHO algorithm utilizes a soft besiege strategy for optimization operations. In this case, hawks cannot easily hunt rabbits because they have much energy to escape, described in Eqs. (4) and (5).

$$X(t+1)=\Delta X(t)-E|JX_{rabbit}(t)-X(t)| \quad (4)$$

$$\Delta X(t)=X_{rabbit}(t)-X(t) \quad (5)$$

In Eq. (4), ΔX, obtained using Eq. (5), represents the distance of the selected hawk to the rabbit, and E is obtained using Eq. (8). J is also the escape energy of the rabbit, which is obtained using the Eq. $\mathrm{J}=2(1-\mathrm{r_5})$.

3.3 Hard besiege

If r ≥ 0.5and|E| < 0.5, the HHO algorithm uses a hard besiege strategy for optimization operations. In this case, hawks can hunt rabbits with rapid attacks because they no

Table 2 Explanations of symbols used in the mathematical model of HHO.

Description	Symbol
location of ith hawk, Position vector of hawks (search agents)	X_i ,X
Average position of hawks	$X_m(t)$
Position of a random hawk	$X_{rand}(t)$
Position of rabbit (best agent)	X_{rabbit}(t)
Swarm size, iteration counter, the maximum number of iterations	N, t,T
Escaping energy, the initial state of energy	E, E_0
Dimension, upper and lower bounds of variables	D, LB,UB
Random numbers inside (0,1)	r_1, r_2, r_3, r_4, r_5,q

longer have enough energy to escape. The mathematical model of this motion is expressed using Eq. (6).

$$X(t+1) = X_{rabbit}(t) - E|\Delta X(t)| \tag{6}$$

If $|E| \geq 0.5$ but $r < 0.5$, the soft besiege strategy with progressive rapid dives is used. In that case, the rabbit has enough energy to escape, and there is still a soft besiege. This procedure is relatively more innovative than the previous procedure.

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X(t)| \tag{7}$$

This strategy uses a Lévy flight to improve performance. Also, the two states of Eq. (12) are compared with the current solution. The Lévy flight has not been used as a result of Y but has been used in Z, given in Eq. (8).

$$Z = Y + S \times LF(D) \tag{8}$$

In Eq. (8), S is a random number in the dimensions of the problem in the interval [0,1], and $LF(D)$ is the flight of Lévy in the dimensions of the problem, expressed in Eq. (9).

otherwise, the solution obtained from Eq. (8) is compared with the current solution. Suppose $|E| < 0.5 and r < 0.5$, the hard besiege strategy with progressive rapid dives is used for optimization operations. In this case, the rabbit does not have enough energy to escape and is besieged hard before the surprise pounce to catch the rabbit. Equations (12) and (13) apply based on Eq. (11).

$$X(t+1) = \begin{cases} Y if F(Y) < F(X(t)) \\ Z if F(Z) < F(X(t)) \end{cases} \tag{11}$$

In Eq. (11), Y and Z are obtained using Eqs. (12) and (13).

$$Y = X_{rabbit}(t) - E|JX_{rabbit}(t) - X_m(t)| \tag{12}$$

$$Z = Y + S \times LF(D) \tag{13}$$

In this strategy, the solution obtained from Eq. (12) replaces the current solution if it is more efficient than others; otherwise; The solution obtained from Eq. (13) will replace it if it is more efficient than the current solution. Algorithm 1 presents the pseudo-code of the HHO algorithm.

Algorithm 1 Pseudo-code of HHO algorithm

Inputs: **The population size N and maximum number of iterations T**
Outputs: **The location of the rabbit and its fitness value**
Initialize the random population $X_i(i = 1,2,\ldots,N)$
while **(stopping condition is not met) do**
  Calculate the fitness values of hawks
  Using all tours' cost function, and the best solution is calculated and named as an X_{rabbit}
  for **(each hawk (X_i)) do**
    Update the initial energy E_0 and jump strength J ▷ $E_0 = 2\,rand() - 1,\ J = 2(1 - rand())$
    Update the E using Eq. (3)
    if ($|E| \geq 1$) then ▷ **the Exploration phase**
      Update the location vector using Eq. (1) ▷ **Soft besiege**
    if ($|E| < 1$) then ▷ **the Exploitation phase**
      if ($r \geq 0.5\ and\ |E| \geq 0.5$) then ▷ **Soft besiege**
        Update the location vector using Eq. (4)
      else if ($r \geq 0.5\ and\ |E| < 0.5$) then ▷ **Hard besiege**
        Update the location vector using Eq. (6)
      else if ($r < 0.5\ and\ |E| \geq 0.5$) then ▷ **Soft besiege with progressive rapid dives**
        Update the location vector using Eq. (10)
      else if ($r < 0.5\ and\ |E| < 0.5$) then ▷ **Hard besiege with progressive rapid dives**
        Update the location vector using Eq. (11)
Return X_{rabbit}

$$LF(x) = 0.01 \times \frac{u \times \sigma}{|v|^{\frac{1}{\beta}}}, \sigma = \left(\frac{\Gamma(1+\beta) \times \sin\left(\frac{\pi\beta}{2}\right)}{\Gamma\left(\frac{1+\beta}{2}\right) \times \beta \times 2^{\left(\frac{\beta-1}{2}\right)}} \right)^{\frac{1}{\beta}} \tag{9}$$

In Eq. (9), u and v are two random numbers between 0 and 1, and β is a fixed and default number, i.e., 1.5.

$$X(t+1) = \begin{cases} Y if F(Y) < F(X(t)) \\ Z if F(Z) < F(X(t)) \end{cases} \tag{10}$$

According to Eq. (10), the result of Eq. (7) is better than the current solution and, consequently, replaces it;

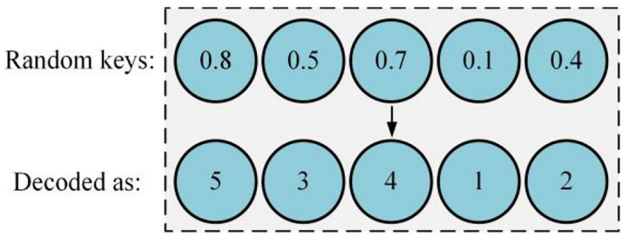


Fig. 1 Random-key encoding scheme

3.4 Random-key encoding scheme

A random-key encoding scheme [64] is an approach to transforming a position and turn it into a combinatorial position in a continuous space. It utilizes a real-number vector by assigning each of the numbers to a certain weight. The weights are then employed to create a combination in the form of a solution. The uniformly drawn random real numbers from the range [0, 1) constitute a vector illustrated in Fig. 1. A combinatorial vector comprises integers ordered according to actual numbers' weights in the first vector, as shown in Fig. 1.

3.4.1 Mutation mechanism

In the exploration phase, the DE/best/2 mutation operator [65], used in many studies, has been used as the primary strategy to increase global search operations performance in the HHO algorithm. The details of this mutation operator are shown in Eq. (14).

$$Xnew = X_{rabbit}(t) + F(X_{r1}(t) - X_{r2}(t)) + (X_{r3}(t) - X_{r4}(t)) \quad (14)$$

In Eq. (14), F is the scaling factor, and $r1$, $r2$, $r3$, and $r4$ are different integers selected from the range [1, N]. N represents the size of the total population. Equation (14) is used in the exploration phase. Mutation operators are used in many models and algorithms to improve optimization performance and global search capability [66–68] that motivated us to use this operator in the proposed algorithm.

3.5 Neighborhood operators

The proposed algorithm uses ten neighborhood search operators [17, 69–75] to improve HHO performance. They are used to achieve optimal solutions. These operators are divided into two groups: point-to-point (pointwise) and sequence operators. In point operators, only one city is selected for change, while in sequence operators, a sequence of cities is selected for change. Point-to-point operators include random swap and random insertion. These ten operators are integrated with the proposed

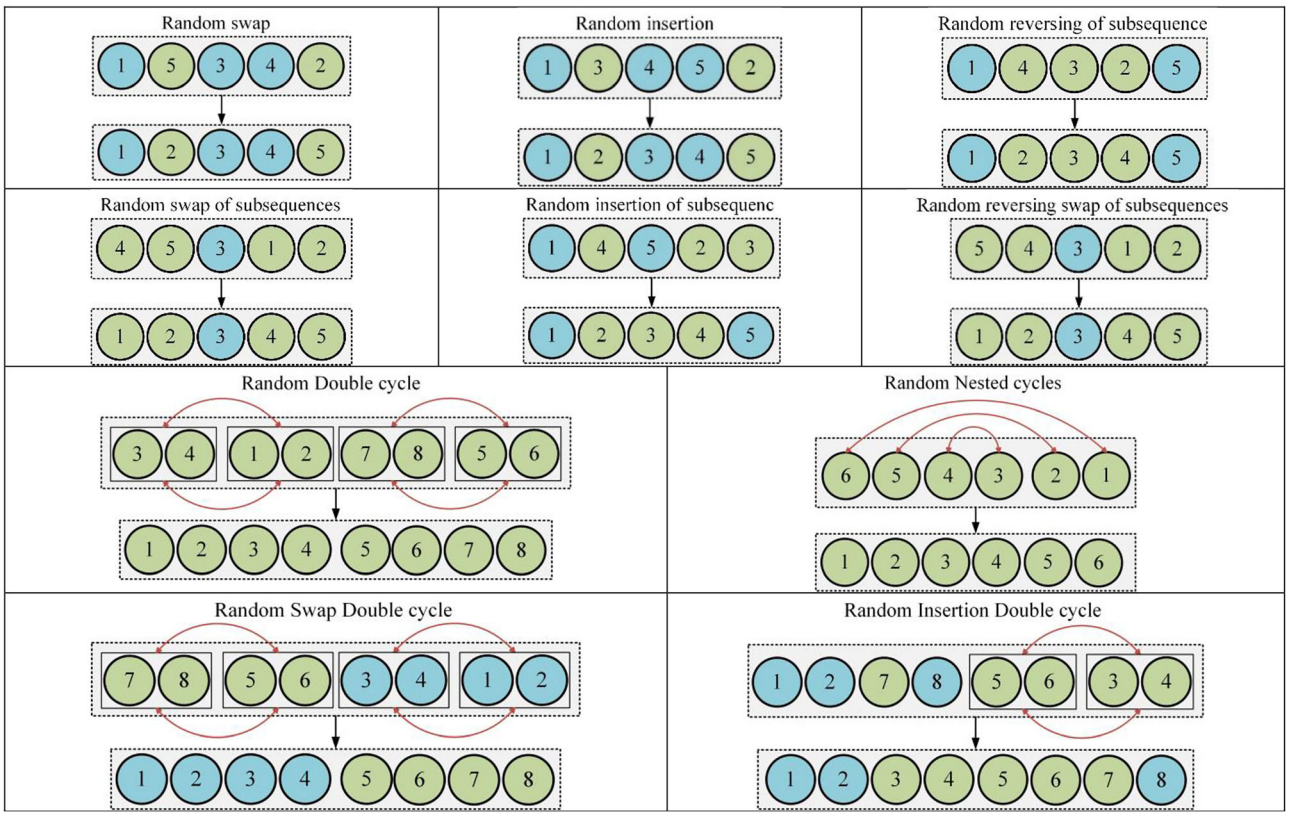


Fig. 2 Function of Neighbourhood operators

algorithm as 10 LLHs (Fig.2), consisting of four primary operations: reverse, insert, swap, and shuffle. RRS specifically performs a reversing of subsequence operation. RI and RIS perform an insert operation. RS and RSS do a swap operation, and SS does a shuffle operation. Of the ten LLHs, three LLHs, RRSS, RIDC, and RSDC, are made up of a combination of two operators.

Random swap (RS): The operator selects two cities at random on a tour and permutation. It then exchanges the location of the two selected cities. Among the advantages of using this operator are keeping more cities adjacent to each other and creating disturbances and minor changes.
Random insertion (RI): This operator selects two cities at random. It then inserts city i in position j. In other words, the insert operator selects a city on tour at random. It then removes that city from the tour, and this action is repeated in the position of another point.
Random Reversing of Subsequence (RRS): The second integrated operator in the HHO algorithm is reversing the subsequence operator. The operator selects a sequence of cities on a random tour. It then reverses the order of cities i to j. In other words, the operator selects two points in the city at random. It then reverses the order of the cities located between the two selected cities.
Random Swap of Sub-Sequences (RSS): The operator selects two sequences with the same number of cities in a tour and permutation. It then exchanges the position and location of two sequences of selected cities with each other.
Random Insertion of Sub-Sequence (RIS): This operator inserts a sequence of randomly selected cities into the position of a randomly selected city.
Random Reversing Swap of Sub-Sequences (RRSS): This operator selects two cities' sequences with the same number at random. It then reverses the cities' order in both sequences and finally replaces the two sequences' positions.
Random Double Cycle (RDC): The operator selects a sequence of cities in pairs (at least 4) in the tour at random. It then replaces the position of the cities in pairs.
Random Nested Cycles (RNC): The operator selects a sequence of cities in pairs (at least 4) in the tour at random. It then replaces the position of the city c_1 with the city c_n, the last city in the selected sequence. It then enters the sequence once and replaces the second city and the last second city in the selected sequence. This process continues until two inland cities are selected in sequence.
Random Swap Double Cycle (RSDC): The operator selects two cities' sequences with the same number in pairs (at least 4) in the tour at random. It then replaces the position of the cities in both sequences in pairs, and finally, replaces the position of the two sequences with each other.
Random Insertion Double Cycle (RIDC): The operator selects a sequence of cities in pairs (at least 4) in the tour at random. It then replaces the position of the cities in pairs, and finally, inserts the position of a sequence of cities into the position of a randomly selected city.

3.6 Modified choice function

Modified choice function heuristic selection variates the choice function, emphasizing intensification over-diversification within the heuristic search process. In [22], a choice-based hyper-heuristic with a score-based approach is presented. The scoring model of each LLH is measured based on the previous performance of that LLH. The score of each LLH consists of three different criteria: f_1, f_2, and f_3. In this model, the first measurement criterion, f_1, calculates the recent performance of each LLH (Eq. (15)):

$$f_1(h_j) = \sum\nolimits_n \alpha^{n-1} \frac{I_n(h_j)}{T_n(h_j)} \tag{15}$$

Where, h_j is the same as LLH^j, $I_n(h_j)$ is the difference between the current solution and the new solution with n^{th} function of h_j, $T_n(h_j)$ is the time spent by nth function of h_j to suggest a new solution, α is a Parameter between 0 and 1, prioritizing recent performance. The second criterion, f_2 Indicates the dependence between a consecutive pair of LLHs (Eq. (16)):

$$f_2(h_k, h_j) = \sum\nolimits_n \beta^{n-1} \frac{I_n(h_k, h_j)}{T_n(h_k, h_j)} \tag{16}$$

where $I_n(h_k, h_j)$ is the value of the difference between the current solution and the new solution using nth consecutive functions of h_k and h_j (i.e., h_j runs right after h_k). $T_n(h_k, h_j)$ is the time spent by nth consecutive functions of functions of h_k and h_j to suggest a new solution, and β is a parameter between 0 and 1, prioritizing recent performance. Criteria f_1 and f_2 are among the intensification components of the selection function, which increase the chances of selecting high-performance LLHs. The third criterion, f_3, records the time elapsed since the last execution of a particular LLH (Eq. (17)):

$$f_3(h_j) = \tau(h_j) \tag{17}$$

where $\tau(h_j)$ is the elapsed time since the last execution of h_j (in seconds). Note that f_3, as a diversification component, plays a role in the selection function, prioritizing those LLHs that have not been used for a long time. The score for each LLH is calculated using the sum of the

weights of all three criteria, f_1, f_2, and f_3, as shown in Eq. (18).

$$F(h_j) = \alpha f_1(h_j) + \beta f_2(h_k, h_j) + \delta f_3(h_j) \quad (18)$$

Where α, β, and δ are parameters that indicate the weights of the criteria f_1, f_2, and f_3 constant values in the initial model. In [76], to increase the improved hyper-heuristic version's efficiency and performance presented in [22], where the parameters can be controlled dynamically during execution. If an LLH improves the solution, the α and β parameters' values increase relative to the degree to which the new solution improves compared to the previous solution. At the same time, there is no improvement in the solution if the LLH is selected. The α and β parameters' values decrease due to the difference in costs between the new solution and the previous solution. Despite the improvements, this version also had some limitations, reviewed in [23]. These restrictions were then lifted. One of these limitations was the reward and penalty mechanism applied by the previous solution, and the new solution commensurate with the resulting improvement. Given the high potential for significant improvement by an insufficient heuristic due to low resolution in the early stages of optimization, it may be advantageous. Also, progress in improving solutions is reduced in the later stages of optimization due to convergence to optimal solutions. It leads to a significant reduction in rewards earned. However, the improvements made in the later stages of optimization are much more critical than those obtained in the early stages. Therefore, this mechanism of reward and penalty is not the correct mechanism. In addition to the above limitations, if no improvement is achieved in the solutions after some iterations, the LLH selection mechanism is done randomly because the α and β parameters decrease due to the reward and penalty mechanism used. Criterion f_3 is a diversification component, which can dominate other criteria. The limitations expressed in [76], an improved version of the selection function, these limitations have been removed in [23], the modified selection function, and this version of the selection function has been provided to manage the parameters better. In this paper, the α and β parameters' selection function is combined into a single parameter, called μ. Finally, the score of each LLH is calculated using Eq. (19).

$$F_t(h_j) = \mu_t[f_1(h_j) + f_2(h_k, h_j)] + \delta f_3(h_j) \quad (19)$$

If an LLH improves the solution, the intensification component is prioritized, and the parameter μ is set to a maximum static value close to 1. At the same time, the parameter δ decreases to a minimum static value close to 0. Conversely, if the LLH does not improve The solution, the parameter μ is fined linearly and bounded below 0.01. This mechanism causes the parameter δ to grow at a uniform and low rate to prevent the rapid loss of intensification components. The parameters μ and δ are calculated using Eqs. (20) and (21). Equation (20) states the difference between the previous solution's cost and its cost.

$$\mu_t(h_j) = \begin{cases} 0.99, & d > 0 \\ \max[0.01, \mu_{t-1}(h_j) - 0.01], & d \le 0 \end{cases} \quad (20)$$

$$\delta_t(h_j) = 1 - \mu_t(h_j) \quad (21)$$

The proposed algorithm uses a modified choice function to automatically select the best LLHs during optimization to increase the proposed algorithm's performance using this mechanism.

3.7 Local-search-based strategies (lin–kernighan local search)

Local search has been used to solve many hybrid optimization problems because it increases the ability to solve such problems. Researchers have proposed different types of local searches, such as opt-2 [77], 3opt [78], and local search [79]. Local search can find local optimum solutions; Therefore, the local search ability is limited to intensification. Using the search is restarted from a specific area of the search space after exploitation for increasing the probability of finding the global optimum. This local search utilizes the retrieval mechanism called multi-start local search (MSLS) [80]. MSLS only allows us to search for different initial solutions and finally obtains a set of local optimum solutions. A global optimum solution or a solution close to the global optimum can be found in the set of local optimum solutions. MSLS has been used to solve many different combinatorial problems, including dynamic TSP [81], generalized quadratic multiple KP [82], and periodic VRP [83].

Also, diversification can be used in local search to increase local search performance. In this case, the solutions are improved by repeated perturbation of the solutions and local search. One method that takes advantage of this mechanism is iterated local search (ILS) [84], in which a recurring initial solution is diversified. At this stage, different solutions are created by perturbation of the solutions. The diversification phase is followed by the intensification phase, in which local search begins based on the solutions generated in the diversification phase.

In [85, 86], Chained Lin-Kernighan (CLK) heuristic used the ILS mechanism to solve TSP. In CLK, a double-bridge motion is followed to exchange the four arcs in the solution with the other four arcs. This operation is performed repeatedly on the solution. ILS-based strategies have been used to solve many different problems, including bin-packing problems [87], scheduling problems [88–90], variants of VRP [91, 92].

Many metaheuristic algorithms have shown good global search performance. The integration of a metaheuristic and local search algorithm strikes a balance between diversification and intensification. Due to the importance of this issue and the increasing performance of the combination of metaheuristic and local search algorithms, shown in many studies [17, 44, 52, 93], the proposed algorithm (MCF-HHO) is integrated with local LK search. Local search is performed after each time changes are made to the solutions using continuous mechanisms or neighborhood operators before the acceptance condition is applied. Because the proposed algorithm uses local search, it is very similar to the ILS and MSLS models. In the initialization phase of the proposed algorithm, a population of solutions is created, creating a different starting point for the local search process. During the proposed algorithm implementation, a series of repeated changes are applied to each section's solutions using an LLH, and the solution is disrupted. LK Local Search follows this. In other words, each solution is subject to the ILS procedure. In light of the above, the proposed algorithm is very similar to the ILS and MSLS models.

3.7.1 Metropolis acceptance strategy

In solving discrete problems such as TSP, the greedy acceptance strategy quickly causes early convergence. To solve this problem, the DHHO model uses the metropolis acceptance criterion to determine whether a new solution will be accepted if it is worse than the current solution. Suppose A is the current tour at cost $f(A)$ and B is the newly produced tour at cost $f(B)$. If $f(B)<f(A)$, i.e., the newly generated tour is better than the current tour, the generated tour will replace the current tour as a new solution. Conversely, if $f(B)>f(A)$, the newly generated network, is worse than the current network, the metropolis acceptance criterion uses a probability mechanism to determine whether the newly generated solution is accepted. The probability of accepting the solution is calculated using Eq. (21).

$$p = \begin{cases} 1, iff(B) \leq f(A) \\ e^{-(f(B)-f(A))/t} otherwise \end{cases}, t = \beta \times t \tag{22}$$

In Eq. (21), $t > 0$ is a temperature parameter. A parameter control strategy must be used to set the initial value of t and adjust the value of this parameter during the search to use the metropolis acceptance criterion in the HHO algorithm. The HHO algorithm is simplified using Eq (15) at the end of each iteration. β is a parameter between 0 and 1, indicating the downward trend of the parameter t. An increase in and approaching this number to 1 reduces the downtrend, and vice versa; if this number approaches 0, the downtrend will increase.

4 Proposed algorithm

This section describes in detail how to integrate the mechanisms described in the previous sections. Equation (22) replaces Eq. (1). In Eq. (22), the mutation mechanism is DE/best/2, used as a powerful operator with high capability in the exploration phase. If $|E| \geq 1$, the proposed algorithm enters the exploration phase and uses Eq. (22). After each Eq. (22) generates the solution, the Lin-Kernighan local search mechanism is executed on the newly generated solution. Finally, the final solution produced replaces the current solution using the metropolis acceptance criterion.

$$Xnew = \begin{cases} X_{rand}(t) - r_1|X_{rand}(t) - 2r_2X(t)|q \geq 0.5 \\ X_{rabbit}(t) + F(X_{r1}(t) - X_{r2}(t)) + (X_{r3}(t) - X_{r4}(t))q < 0.5 \end{cases} \tag{23}$$

Conversely, if $|E| < 1$, the proposed algorithm enters the exploitation phase. In the exploitation phase, two different approaches are used: the first approach relates to the main mechanisms of the HHO algorithm and the second approach to improving the tour using neighborhood search operators. At the beginning of the exploitation phase, rand() generates a random number between 0 and 1. If X≥q (0 <q <1)), the first approach is activated; Otherwise, the second approach is used. If the HHO algorithm uses the first approach and $r \geq 0.5$and$|E| \geq 0.5$, a soft besiege strategy is used to improve the solution.

Conversely, if $r \geq 0.5$and$|E| < 0.5$, a hard besiege strategy is used to improve the tour. Following the soft besiege and hard besiege strategies, the Lin-Kernighan local search mechanism is applied to improve the new tour. Eventually, it will be replaced as the current solution using the metropolis acceptance criterion. However, at the beginning of the exploitation phase, if rand() < q, the HHO algorithm uses the second approach to improve the tour. In this step, a neighborhood search operator is selected using the modified choice function to improve the tour. Then, the Lin-Kernighan local search mechanism is applied to and improves the newly generated tour. Finally, the newly produced tour is selected as the current solution using the metropolis acceptance criterion. Algorithm 2 shows the pseudo-code of the proposed algorithm. Figure 3 shows the flowchart of the proposed algorithm.

Algorithm 2 Pseudo-code of the proposed algorithm

Inputs: The population size N and maximum number of iterations T
Outputs: The location of the rabbit and its fitness value
Initialize the random population $X_i(i = 1,2,\ldots,N)$
while (stopping condition is not met) **do**
  Calculate the fitness values of hawks
Using all tours' cost function, and the best solution is calculated and named as an X_{rabbit}
  for (each hawk (X_i)) **do**
    Update the initial energy E0 and jump strength J
    Update the E using Eq. (3)
    if ($|E| \geq 1$) **then** ▷ the Exploration phase
      $Xnew = Update\ the\ location\ vector\ using\ Eq.(22)$
      $Lin\text{–}Kernighan\ localSearch(Xnew)\ //optional$
      $Accept\ Xnew\ using\ the\ Metropolis\ acceptance\ criterion(21)$
    if ($|E| < 1$) **then** ▷ the Exploitation phase
      if ($rand \geq q$) **then**
        if ($r \geq 0.5\ and\ |E| \geq 0.5$) **then** ▷ Soft besiege
          $Xnew = Update\ the\ location\ vector\ using\ Eq.\ (4)$
          $Lin\text{–}Kernighan\ localSearch(Xnew)\ //optional$
          $Accept\ Xnew\ using\ the\ Metropolis\ acceptance\ criterion(21)$
        else if ($r \geq 0.5\ and\ |E| < 0.5$) **then** ▷ Hard besiege
          $Xnew = Update\ the\ location\ vector\ using\ Eq.(6)$
          $Lin\text{–}Kernighan\ localSearch(Xnew)\ //optional$
          $Accept\ Xnew\ using\ the\ Metropolis\ acceptance\ criterion(21)$
      else if
        $Select\ LLH\ Based\ On\ MCF.$
        $Xnew = Update\ position\ of\ hawk\ (X_i)\ by\ applying\ a\ selectedLLH.$
        $Lin\text{–}Kernighan\ localSearch(Xnew)\ //optional$
        $updateChoiceFunction(selectedLLH)\ //Eq.(19)$
        $Accept\ Xnew\ using\ the\ Metropolis\ acceptance\ criterion(21)$
  $t = \beta \times t$ // Eq. (25)
Return X_{rabbit}

According to Algorithm 2, the computational complexity of the HHO algorithm is $O(N \times (T + TD + 1))$, where N is the number of search factors, T represents the maximum number of iterations, and D represents the dimensions of the problem. In the proposed algorithm, like the HHO algorithm, only one update operation of the mechanisms in the exploration and operation phases is performed on the search agents. Moreover, local Search and MAC operations are performed on each search agent after each update operation. On the other hand, the proposed algorithm uses MCF and LLH mechanisms only in the operation phase if it is rand $\geq$ q. Therefore, the proposed algorithm performs three update processes and local Search and MAC only once in each iteration on each search factor, which complicates the computation.

4.1 Experimental settings

The proposed algorithm has been tested using a computer with the following specifications: Intel Core i7-7700k 4.50 GHz processor and 16 GB of RAM. It is then implemented using the Matlab programming language. A total of 75 instances of the existing datasets in TSPLIB [41]with dimensions of 100-85900 cities have been used. The number in the name of each instance indicates the number of cities in that instance; for example, the number of Kroa100 instance cities is equal to 100. Each TSP instance has been examined a total of 30 times using the proposed algorithm. The best net obtained at each run is placed in the set $X = \{c1, c2, \ldots, c30\}$. The computational time to get the best tour is also placed in another set, i.e., $T = \{t1, t2, \ldots, t30\}$. Here, X is a set of best tours, and T is a set of computational time to get the best tours. Eventually, these two sets are averaged. For this purpose, μ_T Moreover, *Average* are used to display the averages of the T and X sets, respectively. Finally, PDav(%) represents the mean percentage of deviation using Eq. (24).

$$\text{PDav}(\%) = \frac{\text{Average} - \text{BKS}}{\text{BKS}} \times 100 \tag{24}$$

Here, *BKS* indicates the known optimal tour length. Two criteria, namely the percentage of deviation from the known optimum and the computation time, have been used to obtain the best tour (measured in seconds) to evaluate the proposed algorithm's performance. Table 3 shows the parameter setting of the HHO algorithm.

4.2 Experimental results

To evaluate the effect of LLHs performance, if LLHs are integrated with MCF in this section, the three modes of the proposed algorithm, namely, a random selection of LLHs and integration of four main LLHs, i.e., RIS, SS, RSS, and RRS with MCF, and integration of all LLHs with MCF are

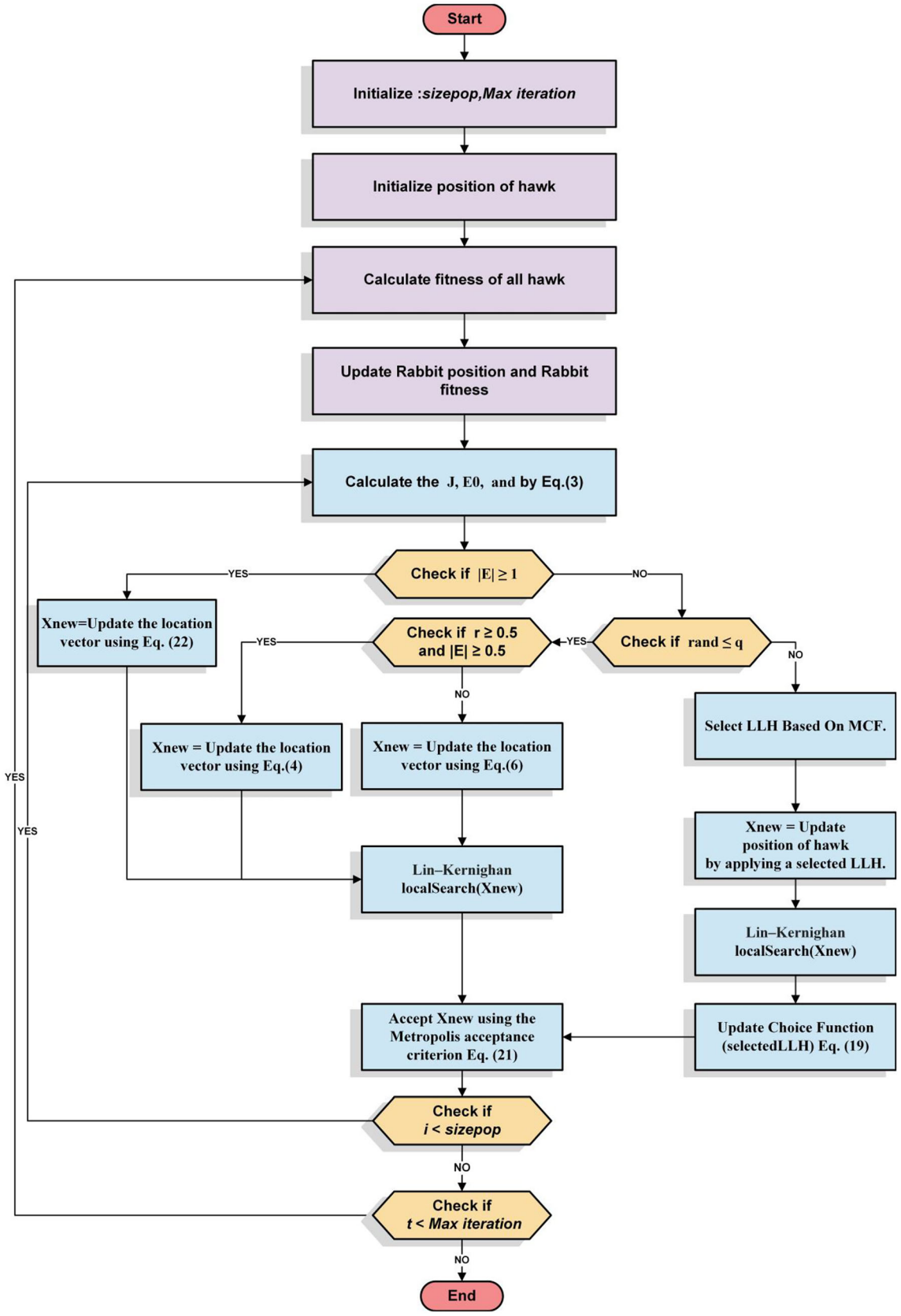


Fig. 3 Flowchart of proposed algorithm

Table 3 HHO algorithm parameter settings

Parameter	Value
POP	10
Max iteration	1000
t	30
β	0.99
F	0.5

compared. This comparison has also been performed to evaluate MCF's performance if integrated with the Harris Hawk Optimization Algorithm. A stop condition of 1000 iterations is provided for all three modes HHO (MCF), HHO(5-LLH), and HHO (Random). To evaluate the performance of these three algorithms, the mean tour length, average percentage of deviation (PDav(%)), and mean computation time was obtained to achieve the best solution (μ_T) (Table 4).

According to the results in Table 3, HHO-MCF has shown the best performance in finding the best tour. HHO-MCF obtained an average of 0.092 (PDav(%)) based on 80 standard TSP instances. The HHO (5-LLH) and HHO (Random) models averaged 0.323 and 0.278, respectively, indicating their more unsatisfactory performance than the HHO-MCF model. On the other hand, with 30 execution times, the HHO-MCF model has solved 34 out of a total of 80 instances of the known optimum. Moreover, the HHO-MCF model has been shown to perform better in most instances; however, the HHO (5-LLH) and HHO (Random) models performed better in some instances.

The Wilcoxon signed-rank test [94] with a 95% confidence interval was used to make a statistical comparison between three models, namely HHO (MCF), HHO (5-LLH), and HHO (Random) in terms of performance. In the Wilcoxon signed-rank test, the difference in PDav(%) values in the two algorithms are used to compare and rank. Instances with equal values are not used in the two algorithms. After discarding equal instances, N indicates the number of use instances and R^+ indicates the set of instance scores. On the other hand, R^- represents the total score of instances in which the proposed algorithm has shown worse performance than the compared algorithm. In the Wilcoxon signed-rank test, the value of W is compared to a critical value, $W_{Cri,N}$. $W \leq W_{Cri,N}$ indicates a significant difference between the two algorithms in terms of performance. In contrast, $W > W_{Cri,N}$ means that there is no significant difference between the two algorithms in terms of performance. Table 5 presents the Wilcoxon signed-rank test results to compare three models, HHO (MCF), HHO(5-LLH), and HHO (Random), indicating the remarkable performance of the HHO (MCF) model.

4.3 Competitiveness HHO-MCF

To further evaluate the performance of the proposed algorithm with other optimization algorithms, which are proposed to solve the TSP problem, which are DFFA [63], DSCA [52], HDABC [95], and MCF-ABC [17] has been tested and compared. These algorithms selected and implemented in this comparison are selected based on similar methods for optimization operations and based on these algorithms' new and powerful. This comparison is based on *Average*, *PDav*, and $\boldsymbol{\mu_T(s)}$ criteria are done. On the other hand, for better evaluation, the Wilcoxon statistical test [96] with a 5% degree of significance has been performed to detect significant differences according to the results obtained from the proposed algorithm compared to other optimization methods. All experiments were performed using ten populations with a maximum of 1000 replications. All results were stored based on an average of 30 independent executions and compared based on these results. Comparison of algorithm settings DFFA [63], DSCA [52], HDABC [95], and MCF-ABC [17] are provided in the main work that has been performed. Also, the parameter settings of the comparing optimization algorithms are given in Table 6.

Tables 7, 8, 9 show the results obtained from testing and evaluating the proposed algorithm and other comparative optimization algorithms using samples of small, medium and large dimensions.

According to Table 7, the results obtained from the proposed algorithm's performance and other comparable algorithms are shown using small-sized samples. The results show that the proposed algorithm and MCF-ABC have achieved the known optimal value in all samples and have good performance in solving small-scale problems. DFFA algorithms have also shown that it has achieved the known optimal value in many samples in the face of small-scale problems. However, the HDABC and DSCA algorithms have achieved the known optimal value in almost half of the problems. According to the $\mu_T(s)$ criterion, Maybe the proposed algorithm has the best performance and achieved quality solutions in less time than other optimization algorithms. Table 8 shows the results of testing the proposed algorithm and other comparable optimization algorithms using medium-sized samples.

Table 8 shows the results obtained from testing the proposed algorithm and other comparable optimization algorithms using medium-sized samples. The results show that the proposed algorithm has a much better performance than other optimization algorithms and can obtain quality solutions compared to other optimization algorithms. Also, the proposed algorithm has been able to achieve the known optimal value in most samples. On the other hand, the

Table 4 Performance evaluation of HHO (MCF), HHO(5-LLH), and HHO (Random) based on 75 TSP standard instances

No.	Instance	BKS	HHO (MCF)			HHO (5-LLH)			HHO (Random)		
			Average	PDav(%)	$\mu_T(s)$	*Average*	PDav(%)	$\mu_T(s)$	*Average*	PDav(%)	$\mu_T(s)$
1	kroA100	21282	21282.0	0.000	0.6	21282.0	0.000	0.4	21282.0	0.000	0.8
2	kroB100	22141	22141.0	0.000	0.3	22141.0	0.000	0.4	22141.0	0.000	0.5
3	kroC100	20749	20749.0	0.000	0.1	20749.0	0.000	0.5	20749.0	0.000	0.3
4	eil101	629	629.0	0.000	0.7	629.0	0.000	0.8	629.0	0.000	0.3
5	lin105	14379	14379.0	0.000	0.2	14379.0	0.000	0.6	14379.0	0.000	0.4
6	pr107	44303	44303.0	0.000	0.3	44303.0	0.000	0.7	44303.0	0.000	0.6
7	gr120	6942	6942.0	0.000	0.9	6942.0	0.000	0.6	6942.0	0.000	0.8
8	pr124	59030	59030.0	0.000	0.5	59030.0	0.000	0.7	59030.0	0.000	0.4
9	bier127	118282	118282.0	0.000	0.3	118282.0	0.000	0.8	118282.0	0.000	0.7
10	ch130	6110	6110.0	0.000	0.2	6110.0	0.000	0.5	6110.0	0.000	0.4
11	pr136	96772	96772.0	0.000	0.8	96772.0	0.000	0.7	96772.0	0.000	0.9
12	gr137	69853	69853.0	0.000	1.1	69853.0	0.000	1.2	69853.0	0.000	1.2
13	pr144	58537	58537.0	0.000	5.1	58537.0	0.000	6.4	58537.0	0.000	6.9
14	ch150	6528	6528.0	0.000	0.2	6528.0	0.000	0.7	6528.0	0.000	0.5
15	kroA150	26524	26524.0	0.000	0.4	26524.0	0.000	0.5	26524.0	0.000	0.3
16	kroB150	26130	26130.0	0.000	0.8	26130.0	0.000	0.6	26130.0	0.000	0.9
17	pr152	73682	73682.0	0.000	8.2	73682.0	0.000	7.8	73682.0	0.000	8.0
18	u159	42080	42080.0	0.000	0.5	42080.0	0.000	0.6	42080.0	0.000	0.4
19	si175	21407	21407.0	0.000	0.9	21407.0	0.000	0.9	21407.0	0.000	0.7
20	brg180	1950	1950.0	0.000	0.4	1950.0	0.000	0.6	1950.0	0.000	0.4
21	rat195	2323	2323.0	0.000	0.8	2323.0	0.000	0.4	2323.0	0.000	1.1
22	d198	15780	15780.0	0.000	11.8	15780.0	0.000	11.6	15780.0	0.000	11.5
23	kroA200	29368	29368.0	0.000	3.1	29368.0	0.000	3.2	29368.0	0.000	3.1
24	kroB200	29437	29437.0	0.000	0.5	29437.0	0.000	0.8	29437.0	0.000	0.8
25	gr202	40160	40160.0	0.000	1.5	40160.0	0.000	3.2	40160.0	0.000	2.7
26	tsp225	3916	3916.0	0.000	0.6	3916.0	0.000	0.8	3916.0	0.000	0.7
27	ts225	126643	126643.0	0.000	0.8	126643.0	0.000	0.9	126643.0	0.000	0.4
28	pr226	80369	80369.0	0.000	7.7	80369.0	0.000	9.3	80369.0	0.000	8.6
29	gr229	134602	134602.0	0.000	1.7	134602.0	0.000	2.4	134602.0	0.000	3.2
30	gil262	2378	2378.0	0.000	0.8	2378.0	0.000	0.9	2378.0	0.000	1.1
31	pr264	49135	49135.0	0.000	0.6	49135.0	0.000	1.2	49135.0	0.000	1.2
32	a280	2579	2579.0	0.000	0.7	2579.0	0.000	0.6	2579.0	0.000	0.8
33	pr299	48191	48191.0	0.000	1.3	48191.0	0.000	1.2	48191.0	0.000	1.2
34	lin318	42029	42029.0	0.000	3.2	42029.0	0.000	8.2	42029.0	0.000	5.3
35	rd400	15281	15281.0	0.000	6.8	15281.0	0.000	5.9	15281.0	0.000	6.1
36	fl417	11861	11861.0	0.000	12.4	11861.0	0.000	14.5	11861.0	0.000	13.8
37	gr431	171414	171414.0	0.000	23.6	171414.0	0.000	33.3	171414.0	0.000	27.9
38	pr439	107217	107217.0	0.000	6.2	107217.0	0.000	7.4	107217.0	0.000	5.9
39	pcb442	50778	50778.0	0.000	3.6	50778.0	0.000	3.8	50778.0	0.000	4.1
40	d493	35002	35002.0	0.000	51.2	35003.5	0.004	59.6	35003.7	0.005	49.7
41	att532	27686	27686.9	0.003	43.5	27688.2	0.008	47.2	27687.5	0.005	39.4
42	ali535	202339	202339.0	0.000	18.7	202341.9	0.001	21.6	202347.6	0.004	20.9
43	si535	48450	48493.1	0.089	72.3	48484.8	0.072	89.1	48512.3	0.129	91.2
44	pa561	2763	2763.0	0.000	15.4	2763.3	0.011	17.2	2763.7	0.025	20.3
45	u574	36905	36905.8	0.002	6.5	36906.1	0.003	9.1	36906.5	0.004	8.7
46	rat575	6773	6774.2	0.018	13.8	6775.4	0.035	12.4	6774.7	0.025	14.5
47	p654	34643	34643.6	0.002	51.2	34644.5	0.004	49.3	34645.1	0.006	52.8

Table 4 (continued)

No.	Instance	BKS	HHO (MCF)			HHO (5-LLH)			HHO (Random)		
			Average	PDav(%)	$\mu_T(s)$	*Average*	PDav(%)	$\mu_T(s)$	*Average*	PDav(%)	$\mu_T(s)$
48	d657	48912	48913.9	0.004	21.9	48917.7	0.012	30.5	48915.9	0.008	24.6
49	gr666	294358	294392.1	0.012	58.6	294390.3	0.011	75.7	294402.2	0.015	69.1
50	u724	41910	41913.4	0.008	32.9	41921.6	0.028	34.8	41917.8	0.019	33.4
51	rat783	8806	8806.0	0.000	9.4	8806.0	0.000	11.6	8806.0	0.000	10.2
52	dsj1000	18659688	18662009.8	0.012	117.3	18663849.2	0.022	119.1	18664175.4	0.024	103.9
53	pr1002	259045	259062.5	0.007	43.5	259262.1	0.084	51.4	259056.3	0.004	59.3
54	si1032	92650	92650.0	0.000	15.1	92653.1	0.003	13.2	92655.6	0.006	18.5
55	U1060	224094	224124.1	0.013	30.2	224138.4	0.020	34.5	224149.5	0.025	32.7
56	vm1084	239297	239315.6	0.008	63.7	239330.9	0.014	58.3	239342.8	0.019	68.6
57	pcb1173	56892	56901.3	0.016	25.4	56912.3	0.036	30.7	56909.1	0.030	36.8
58	d1291	50801	50829.6	0.056	45.9	51464.8	1.307	72.4	51404.7	1.188	110.2
59	rl1304	252,948	253108.2	0.063	26.7	253214.4	0.105	32.5	253301.5	0.140	25.1
60	rl1323	270,199	270328.1	0.048	37.4	270584.5	0.143	41.8	270412.8	0.079	45.5
61	d1655	62128	62213.5	0.138	64.8	62200.5	0.117	78.6	62287.4	0.257	107.3
62	vm1748	336,556	336857.3	0.090	70.1	337004.3	0.133	82.5	336824.9	0.080	87.3
63	u1817	57201	57326.2	0.219	63.3	57383.7	0.319	35.1	57362.3	0.282	94.4
64	rl1889	316536	317298.4	0.241	95.1	317501.6	0.305	106.5	317260.8	0.229	112.5
65	u2152	64253	64435.1	0.283	32.6	64467.2	0.333	57.2	64493.6	0.374	66.1
66	u2319	234,256	234827.6	0.244	43.2	234916.7	0.282	39.5	235026.4	0.329	40.5
67	pr2392	378032	378667.9	0.168	67.5	378868.4	0.221	84.3	378987.5	0.253	89.8
68	pcb3038	137694	137944.8	0.182	125.4	137911.9	0.158	127.8	138177.2	0.351	114.6
69	fl3795	28772	28829.7	0.201	321.7	31900.8	10.874	486.4	31379.1	9.061	591.7
70	fnl4461	182566	182968.2	0.220	62.8	183242.1	0.370	49.8	183177.9	0.335	68.2
71	rl5915	565530	567761.6	0.395	204.6	572012.3	1.146	217.5	571071.4	0.980	209.3
72	Rl5934	556045	557898.1	0.333	512.3	558973.5	0.527	583.2	559769.7	0.670	549.9
73	pla7397	23260728	23322408.5	0.265	506.1	23385448.6	0.536	497.2	23399754.3	0.598	467.4
74	rl11849	923288	927897.8	0.499	642.3	935837.4	1.359	728.6	936252.8	1.404	771.5
75	Usa13509	19982859	20103912.4	0.606	2057.4	20220179.9	1.188	1924.9	20088279.3	0.528	2317.9
76	Brd14051	469388	471976.2	0.551	1163.7	475239.8	1.247	1269.1	474804.2	1.154	1374.4
77	D15112	1573084	1581138.6	0.512	652.7	1587269.4	0.902	690.4	1585243.9	0.773	668.9
78	D18512	645244	648751.9	0.544	805.2	653861.7	1.336	719.8	648537.1	0.510	775.8
79	Pla33810	66050535	66438091.7	0.587	7976.6	66859473.2	1.225	4867.0	66789347.4	1.119	6208.4
80	pla85900	142382641	143400437.3	0.715	11996.7	144296843.5	1.344	10857.1	144127112.6	1.225	9366.7
Average:			0.092		354.7	0.323		306.9	0.278		314.3

Table 5 Wilcoxon signed-rank test to compare three models: HHO (MCF) and HHO(5-LLH) and HHO (Random)

Comparisons HHO (MCF) vs.…	*N*	R^+	R^-	*W*	$W_{\text{Cri,N}}$	Significant difference
HHO (5-LLH)	40	769.5	50.5	50.5	264	Yes
HHO (Random)	40	746.5	73.5	73.5	264	Yes

MCF-ABC algorithm is ranked second in performance and has achieved the known optimal value in some samples. According to the $\mu_T(s)$ criterion, the proposed algorithm in solving medium-sized problems has been able to achieve quality solutions in less time and perform much better than other comparable algorithms in most samples Has shown

Table 6 Parameter settings of optimization algorithms for comparison and evaluation of the proposed algorithm

Algorithm	Parameter	Value
HDABC	length of threshold list	300
	Limit	100
DSCA	R_{l1}	random numbers between 0 and 1
	R_{l2}	random numbers between 0 and 1
DFFA	K-Value	3
	Pop	10
	T	8
	Alpha	0.99
	Swap Prob.	0.2
	Inverse Prob.	0.5
	Insertion Prob.	0.3
MCF-ABC	limit	200
MCF-HHO	t	30
	β	0.99
	F	0.5

itself. Table 9 shows the results of experiments performed using large samples.

According to the results of Table 9, which are the results obtained from the proposed algorithm and other comparable algorithms using large-sized samples, it is clear that the proposed algorithm has performed significantly better in the face of large-scale problems and has been able to In almost all samples to obtain quality and better solutions than the comparable algorithms. On the other hand, the MCF-ABC algorithm has the second-best performance and can obtain acceptable solutions. The third-best performance in experiments performed on large-scale samples is related to DFFA, which can obtain acceptable results in some samples. According to the $\boldsymbol{\mu_T(s)}$ criterion, the proposed algorithm can still obtain quality solutions in less time than other comparable optimization algorithms, and in almost all samples, it takes less time to find solutions that have spent quality. In this paper, to further investigate the performance of the *PDav* criteria proposed algorithm, we compared the proposed algorithm with D-CLPSO [46], SOM [97], and DPIO [48] algorithms in large-scale comparative samples. The results are shown in Table 10.

According to the results shown in Table 10, the proposed algorithm has a good and significant performance compared to the compared algorithms and has achieved much better results than the compared algorithms. The proposed algorithm has excellent and acceptable performance in the face of complex and significant issues. To further evaluate the proposed algorithm, it is compared with CLK [78], with the results shown in Table 11. The CLK source code [85], available in the Concorde TSP solver software, has been used to compare the proposed algorithm with the CLK. Large-scale instances are used for comparison. It is a single-solution-based model in which the population is equal to 1. This model is allowed to perform a maximum of 10,000 iterations. The default settings in Concorde are maintained to run as follows: initialization method (i.e., Quick-Boruvka), choice of the kick (i.e., 50-step random-walk kick) and the level of backtracking (i.e. (4, 3, 3, 2)-breadth).

Table 11 compares the proposed algorithm with the CLK model. Both models use a similar LK local search strategy. Examining this comparison results shows that the proposed algorithm performs better in small-scale instances, and the CLK model performs better in large-scale instances. The Wilcoxon signed-rank test with a confidence interval of 95% was utilized for comparing the proposed algorithm with other algorithms in terms of performance.

Table 12 compares the proposed algorithm with other competing performance optimization algorithms using the Wilcoxon signed-rank test with a confidence interval of 95%. The results indicate significantly better performance of the proposed algorithm compared to the other six models. In all comparisons R^{+}greaterthan$R^{-}$and$W$smallerthan$W_{Cri,N}$ except for the CLK, that is Wgreaterthan$W_{Cri,N}$.

5 Discussion of the results

In this subsection, the proposed algorithm's performance is discussed based on the tests and evaluations performed. Table 4 shows the evaluation of MCF performance and the impact of neighborhood search operators on its

Table 7 Comparison of the proposed algorithm with other models in small dimensions instance

Instance	Proposed Algorithm			MCF-ABC [17]			DFFA [63]			HDABC [95]			DSCA [52]		
	Average	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$
kroA100	21282.0	0.000	0.6	21282.0	0.000	0.5	21282.0	0.000	0.9	21282.0	0.000	0.4	21282.0	0.000	1.2
kroB100	22141.0	0.000	0.3	22141.0	0.000	0.6	22141.0	0.000	0.7	22159.2	0.082	0.5	22161.8	0.094	1.3
kroC100	20749.0	0.000	0.1	20749.0	0.000	0.4	20749.0	0.000	1.1	20749.0	0.000	0.7	20749.0	0.000	1.6
eil101	629.0	0.000	0.7	629.0	0.000	0.9	629.1	0.015	1.5	629.4	0.063	1.9	630.2	0.191	0.8
lin105	14379.0	0.000	0.2	14379.0	0.000	0.4	14382.4	0.024	1.2	14381.5	0.017	0.5	14384.5	0.038	0.9
pr107	44303.0	0.000	0.3	44303.0	0.000	0.1	44303.0	0.000	0.2	44324.1	0.047	0.5	44303.0	0.000	0.5
gr120	6942.0	0.000	0.6	6942.0	0.000	0.8	6942.0	0.000	0.5	6945.8	0.055	0.9	6944.7	0.039	0.2
pr124	59030.0	0.000	0.5	59030.0	0.000	0.9	59030.0	0.000	1.7	59030.0	0.000	1.1	59030.0	0.000	1.1
bier127	118282.0	0.000	0.3	118282.0	0.000	0.6	118282.0	0.000	1.1	118301.4	0.016	1.7	118314.8	0.027	1.8
ch130	6110.0	0.000	0.2	6110.0	0.000	0.1	6110.0	0.000	0.4	6114.2	0.069	0.2	6116.3	0.103	0.7
pr136	96772.0	0.000	0.5	96772.0	0.000	0.7	96772.0	0.000	1.2	96772.0	0.000	0.7	96772.0	0.000	1.6
gr137	69853.0	0.000	1.1	69853.0	0.000	1.3	69875.9	0.033	1.9	69889.7	0.053	0.8	69905.7	0.075	2.5
pr144	58537.0	0.000	4.1	58537.0	0.000	6.2	58537.0	0.000	5.9	58537.0	0.000	4.1	58549.8	0.022	7.1
ch150	6528.0	0.000	0.1	6528.0	0.000	0.1	6528.0	0.000	0.4	6544.9	0.259	0.8	6528.0	0.000	0.3
kroA150	26524.0	0.000	0.4	26524.0	0.000	0.5	26524.0	0.000	0.7	26539.6	0.059	1.2	26524.0	0.000	1.5
kroB150	26130.0	0.000	0.8	26130.0	0.000	0.6	26130.0	0.000	1.4	26130.0	0.000	0.8	26130.0	0.000	1.8
pr152	73682.0	0.000	8.2	73682.0	0.000	9.4	73682.0	0.000	8.9	73682.0	0.000	10.8	73698.1	0.021	10.1
u159	42080.0	0.000	0.5	42080.0	0.000	0.7	42080.0	0.000	1.5	42080.0	0.000	1.2	42096.4	0.039	2.2
si175	21407.0	0.000	0.9	21407.0	0.000	0.5	21407.0	0.000	0.8	21422.7	0.073	0.7	21418.9	0.056	0.7
brg180	1950.0	0.000	0.4	1950.0	0.000	0.6	1952.7	0.138	0.9	1954.5	0.231	1.4	1952.7	0.138	0.5
rat195	2323.0	0.000	0.8	2323.0	0.000	0.9	2327.2	0.181	1.7	2334.6	0.499	1.6	2329.2	0.267	1.2
d198	15780.0	0.000	11.8	15780.0	0.000	12.1	15792.5	0.079	14.5	15799.3	0.122	12.9	15787.8	0.049	14.6
kroA200	29368.0	0.000	3.1	29368.0	0.000	4.3	29368.0	0.000	6.7	29381.8	0.047	4.9	29368.0	0.000	8.5
kroB200	29437.0	0.000	0.4	29437.0	0.000	0.2	29437.0	0.000	0.5	29444.1	0.024	0.8	29437.0	0.000	0.8
gr202	40160.0	0.000	0.7	40160.0	0.000	0.9	40160.0	0.000	1.7	40185.3	0.063	2.6	40175.6	0.039	2.4
tsp225	3916.0	0.000	0.6	3916.0	0.000	0.9	3916.0	0.000	1.6	3918.4	0.061	2.1	3917.9	0.049	2.3
ts225	126643.0	0.000	0.8	126643.0	0.000	0.5	126643.0	0.000	0.8	126643.0	0.000	1.4	126781.2	0.109	1.1
pr226	80369.0	0.000	7.7	80369.0	0.000	8.2	80369.0	0.000	8.2	80369.0	0.000	7.9	80397.7	0.036	9.9
gr229	134602.0	0.000	1.7	134602.0	0.000	2.2	134697.6	0.071	4.6	134710.6	0.081	4.1	134701.4	0.074	5.2
gil262	2378.0	0.000	0.2	2378.0	0.000	0.5	2379.8	0.076	0.8	2382.8	0.202	1.2	2380.1	0.088	1.5
pr264	49135.0	0.000	0.5	49135.0	0.000	0.7	49135.0	0.000	1.8	49135.0	0.000	1.5	49135.0	0.000	0.9
a280	2579.0	0.000	0.7	2579.0	0.000	0.5	2580.2	0.047	2.1	2582.4	0.132	1.4	2585.8	0.264	2.3
pr299	48191.0	0.000	1.3	48191.0	0.000	1.9	48195.5	0.009	3.4	48236.5	0.094	2.7	48219.5	0.059	5.8

Table 8 Comparison of the proposed algorithm with other models in medium dimensions' instance

Instance	Proposed Algorithm			MCF-ABC [17]			DFFA [63]			HDABC [95]			DSCA [52]		
	Average	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$
lin318	42029.0	0.000	3.2	42029.0	0.000	4.1	42064.3	0.084	6.3	42121.2	0.219	2.9	42111.5	0.196	7.2
rd400	15281.0	0.000	6.8	15281.0	0.000	5.9	15296.7	0.103	7.4	15306.5	0.167	8.1	15308.8	0.182	10.4
fl417	11861.0	0.000	12.4	11861.0	0.000	14.7	11887.5	*0.223*	11.7	11946.8	0.723	10.6	11901.2	*0.339*	8.2
gr431	171414.0	0.000	23.6	171414.0	0.000	22.9	171507.1	*0.054*	21.7	171837.4	0.247	14.8	171846.9	*0.253*	34.5
pr439	107217.0	0.000	6.2	107217.0	0.000	7.3	107217.0	0.000	6.5	107364.7	0.138	9.3	107315.4	*0.092*	9.7
pcb442	50778.0	0.000	3.6	50778.0	0.000	4.4	50791.6	*0.027*	6.9	50801.3	0.046	4.8	50834.7	*0.112*	3.9
d493	35002.0	0.000	51.2	35005.2	0.009	67.5	35008.4	*0.018*	79.5	35019.1	0.049	66.2	35023.9	*0.063*	67.1
att532	27686.9	0.003	43.5	27687.5	0.005	39.7	27699.2	*0.048*	33.4	27705.9	0.072	59.7	27702.6	*0.060*	42.8
ali535	202339.0	0.000	18.7	202339.0	0.000	20.4	202457.8	*0.059*	28.1	202747.5	0.202	23.1	202802.3	*0.229*	25.4
si535	48493.1	0.089	72.3	48499.8	0.103	66.1	48528.5	*0.162*	79.3	48524.7	0.154	62.4	48537.8	*0.181*	89.3
pa561	2763.0	0.000	15.4	2763.2	0.007	21.2	2765.1	*0.076*	15.2	2764.3	0.047	14.2	2763.5	*0.018*	22.6
u574	36905.8	0.002	6.5	36905.0	0.000	5.8	36975.3	*0.190*	6.2	37032.9	0.347	6.8	37006.2	*0.274*	4.8
rat575	6774.2	0.018	13.8	6775.1	0.031	11.7	6784.7	0.173	18.7	6807.4	0.508	19.3	6795.7	0.335	25.9
p654	34643.6	0.002	51.2	34644.1	0.003	63.4	34709.6	*0.192*	89.5	34729.8	0.251	49.7	34715.4	*0.209*	90.5
d657	48913.9	0.004	21.9	48916.8	0.010	18.6	48935.9	*0.049*	12.8	48944.2	0.066	22.4	48952.9	*0.084*	10.2
gr666	294392.1	0.012	58.6	294399.6	0.014	69.5	295087.2	*0.248*	70.6	295018.6	0.224	67.9	295002.1	*0.219*	63.7
u724	41913.4	0.008	32.9	41916.2	0.015	27.1	41954.4	*0.106*	29.3	41996.5	0.206	25.5	42041.8	*0.314*	32.4
rat783	8806.0	0.000	9.4	8806.0	0.000	11.3	8806.0	0.000	15.9	8847.1	0.467	11.6	8882.3	*0.866*	10.8
dsj1000	18662009.8	0.012	117.3	18661076.5	0.007	129.4	18686694.5	*0.145*	144.1	18694871.3	0.189	139.8	18699767.7	*0.215*	128.6
pr1002	259062.5	0.007	43.5	259075.9	0.012	38.9	259478.8	0.167	25.3	260669.7	0.627	36.3	261941.6	1.118	29.3
si1032	92650.0	0.000	15.1	92650.0	0.000	12.7	92798.1	*0.159*	19.1	92874.5	0.242	11.4	92828.5	*0.193*	17.1
U1060	224124.1	0.013	30.2	224131.4	0.017	40.8	224934.6	*0.375*	59.4	224917.9	0.368	34.9	224978.2	*0.395*	42.7

Table 9 Comparison of the proposed algorithm with other models in high dimensions' instance

Instance	Proposed Algorithm			MCF-ABC [17]			DFFA [63]		
	Average	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$
vm1084	239315.6	0.008	63.7	239325.4	0.012	75.1	239876.1	*0.242*	110.2
pcb1173	56901.3	0.016	15.4	56899.7	**0.014**	19.3	57061.5	*0.298*	21.5
d1291	50829.6	0.056	32.9	50838.1	0.073	39.2	51057.2	0.504	51.7
rl1304	253108.2	0.063	26.7	253327.8	0.150	41.7	254095.3	0.454	55.3
rl1323	270328.1	0.048	37.4	270528.2	0.122	51.2	271089.5	0.330	49.6
d1655	62213.5	0.138	64.8	62229.8	0.164	86.9	62397.9	0.434	79.2
vm1748	336857.3	0.090	70.1	336976.2	0.125	87.3	338012.9	0.433	91.5
u1817	57326.2	0.219	60.3	57340.5	0.244	59.5	57424.7	*0.391*	67.3
rl1889	317298.4	0.241	95.1	317367.3	0.262	124.3	317578.4	0.329	101.5
u2152	64435.1	0.283	18.6	64432.6	**0.280**	25.1	64439. 5	0.290	39.4
u2319	234827.6	0.244	43.2	235018.4	**0.325**	47.5	235097.8	0.359	61.2
pr2392	378667.9	0.168	67.5	378689.4	0.174	78.7	380255.3	*0.588*	96.6
pcb3038	137944.8	0.182	97.4	138005.9	0.227	110.2	138416.8	0.525	137.9
fl3795	28829.7	0.201	321.7	28821.2	**0.171**	385.6	28984.6	*0.739*	405.7
fnl4461	182968.2	0.220	59.8	182994.5	0.235	49.2	183497.1	0.510	78.6
rl5915	567761.6	0.395	204.6	567801.8	0.402	229.8	569938.4	*0.780*	287.3
rl5934	557898.1	0.333	512.3	558027.7	0.357	601.5	561798.2	*1.035*	725.4
pla7397	23322408.5	0.265	437.1	23324982.3	0.276	489.1	23511293.5	*1.077*	547.2
rl11849	927897.8	0.499	642.3	927984.2	0.509	712.6	932412.2	*0.988*	627.8
Usa13509	20103912.4	0.606	2057.4	20107564.5	0.624	2174.2	20258426.3	*1.379*	2351.7
Brd14051	471976.2	0.551	973.7	472054.1	0.568	1065.3	473071.6	*0.785*	1195.3
D15112	1581138.6	0.512	652.7	1582954.7	0.627	729.4	1591532.8	*1.173*	637.8
D18512	648751.9	0.544	805.2	649165.5	0.608	1154.8	651789.7	*1.014*	1376.2
Pla33810	66438091.7	0.587	7976.6	66443178.4	0.594	8034.2	67042973.9	*1.503*	8647.7
pla85900	143400437.3	0.715	11996.7	143498824.2	0.784	12985.7	144810748.5	*1.705*	13164.5

HDABC [95]			DSCA [52]		
Average	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$
239756.2	0.192	54.8	241328.2	*0.849*	95.4
57271.8	0.668	15.2	57154.6	0.462	31.8
51412.5	1.204	59.9	51209.8	0.805	68.4
254107.4	0.458	31.2	254964.3	0.797	43.7
271162.7	0.357	52.4	272039.5	0.681	61.5
63016.9	1.431	97.6	62894.5	1.234	59.1
339197.6	0.785	83.4	340158.7	1.070	98.6
57891.3	1.207	71.3	57834.5	*1.107*	85.6
319854.7	1.048	142.6	320109.3	1.129	127.3
64984.9	1.139	33.4	64459.7	0.322	47.9
234911.6	0.280	72.5	236734.8	1.058	50.7
381625.4	0.951	69.5	381714.6	*0.974*	136.2
139214.5	1.104	126.7	139722.4	1.473	155.1
29062.3	1.009	422.9	29149.5	*1.312*	474.5
184987.8	1.327	56.3	187443.9	2.672	61.9
571735.2	1.097	247.1	571987.6	*1.142*	310.2
565418.6	1.686	771.9	563245.2	*1.295*	698.7
23524872.9	1.136	516.4	23564264.8	*1.305*	505.3
937897.4	1.582	682.7	938056.3	*1.600*	658.4

Table 9 (continued)

HDABC [95]			DSCA [52]		
Average	*PDav*	$\mu_T(s)$	*Average*	*PDav*	$\mu_T(s)$
20312749.7	1.651	1978.6	20344117.4	*1.808*	2572.6
477587.5	1.747	1299.5	476195.1	*1.450*	1078.5
1596105.3	1.463	819.3	1595410.9	*1.419*	615.2
653412.2	1.266	1267.3	654027.5	*1.361*	1542.1
67351482.6	1.970	8469.8	67478362.2	*2.162*	8894.9
145279432.4	2.035	1352.4	147289104.7	*3.446*	13746.7

Table 10 Comparison of the proposed algorithm with DPIO, SOM, and D-CLPSO based on *PDav* criteria

Instance	Proposed algorithm	DPIO [48]	SOM [97]	D-CLPSO [46]
U1060	**0.013**	0.374	5.12	-
vm1084	**0.008**	0.327	5.86	-
pcb1173	**0.016**	0.392	7.50	0.737
d1291	**0.056**	0.668	9.66	0.645
d1655	**0.138**	0.369	9.60	0.819
u1817	**0.219**	0.561	9.68	-
rl1889	**0.241**	0.688	9.54	0.716
u2152	**0.283**	0.838	10.43	-
pr2392	**0.168**	0.612	7.04	1.042
pcb3038	**0.182**	0.624	7.88	0.998
fl3795	**0.201**	1.52	16.13	-
fnl4461	**0.220**	0.961	5.62	1.222
rl5915	**0.395**	1.005	12.94	-
rl5934	**0.333**	1.041	13.02	1.205
pla7397	**0.265**	1.441	10.19	1.427
rl11849	**0.499**	1.062	11.49	-
Usa13509	**0.606**	1.168	7.62	1.466
Brd14051	**0.551**	1.051	6.18	1.328
D15112	**0.512**	0.984	5.95	–
D18512	**0.544**	1.049	6	1.362
Pla33810	**0.587**	1.726	13.23	2.10
pla85900	**0.715**	1.378	10.94	1.64

improvement. According to these evaluations, the proposed algorithm using MCF and all ten neighborhood search operators has obtained higher quality tours than other compared models. This evaluation also shows that MCF has a significant impact on obtaining quality solutions. Because the intelligent choice of neighborhood search operators during optimization operations can be significant, on the other hand, it turns out that the use of MCF also leads to quality tours in a shorter period.

Table 7 shows the results obtained from the proposed algorithm's performance with DSCA, HDABC, DFFA and MCF-ABC algorithms in small samples. The proposed algorithm in all small dimensions has been able to achieve the known optimal value. However, the compared algorithms also performed well in small samples. Nevertheless, the proposed algorithm in terms of time has achieved quality tours in less time. However, the results of evaluations of medium-sized samples are shown in Table 8. The proposed algorithm has a much better performance than the compared algorithms compared to the medium-sized models, and in most of the samples, it has been able to obtain higher quality tours than the compared algorithms in a shorter period. Finally, Table 9 shows the results obtained from the proposed algorithm and the comparable algorithms in solving large-sized samples. According to the evaluations made with large samples, it has been

Table 11 Comparison of the proposed algorithm with the CLK model

Method	Instance													
	pr1002	si1032	vm1084	pcb1173	d1291	d1655	u1817	u2152	pr2392	fl3795	fnl4461	rl5915	pla7397	rl11849	pla85900
CLK [85]	0.126	0.007	0.038	0.029	0.231	0.157	0.333	0.529	0.269	0.707	**0.142**	**0.293**	0.274	**0.437**	**0.692**
Proposed algorithm	**0.007**	**0.001**	**0.008**	**0.016**	**0.056**	**0.138**	**0.219**	**0.283**	**0.168**	**0.201**	0.220	0.395	**0.265**	0.499	0.715

determined that the proposed algorithm has much better performance and efficiency than the compared algorithms because it has been able to make quality tours for a shorter time than the compared algorithms.

On the other hand, in the comparison made in Table 10, the proposed algorithm has shown that it has a significantly good performance compared to DPIO, SOM and D-CLPSO algorithms and has much more capability and efficiency results obtained. It has been significantly better than the proposed algorithm. Compared to CLK, the proposed algorithm has shown complete superiority in small and medium-sized samples, but in partially large CLK samples, it has achieved slightly better results.

In general, the good results obtained from the proposed algorithm indicate the selection of excellent and complementary mechanisms. Because each of these mechanisms has increased the ability and balance of intensification and diversification components, as well as Multi-start Local Search (MSLS) [80] and Iterated Local Search (ILS) [84] systems have been created.

6 Conclusion and future works

The HHO algorithm is a new metaheuristic algorithm used to solve specific problems. This paper utilizes random-key encoding of the continuous optimization problem space to solve the TSP, a discrete permutation problem. The primary purpose of using random-key encoding is to maintain robust HHO strategies. Furthermore, a mutation operator, DE/best/2, was employed in the exploration phase to increase the proposed algorithm's performance in the exploitation phase. Ten neighborhood search operators have been utilized in the exploitation phase, four of which were presented to solve the TSP. A hyper-heuristic mechanism based on an MCF was used to select each neighborhood search operator. The MCF allows intelligent and automatic selection of neighborhood search operators during optimization.

On the other hand, a local search strategy called Lin-Kernighan (LK) was employed to improve the proposed algorithm's performance. Finally, the Metropolis acceptance strategy was utilized to escape the local optima trap. This paper presents and evaluates three models, namely HHO (MCF), HHO (5-LLH), and HHO (Random). The proposed algorithm was tested with 80 instances using two criteria: the computation time to obtain the best tour and the average deviation, compared with the seven previously presented models. The results indicated an acceptable performance of the proposed algorithm. The algorithm presented in this paper can be used to solve other problems. It is a good solution for use in other issues. In the future, the proposed algorithm can be used to solve combinatorial

Table 12 Wilcoxon signed-rank test to compare the proposed algorithm with other models

Comparisons HHO (MCF) vs.…	N	R^+	R^-	W	$W_{\text{Cri,N}}$	Significant difference
MCF-ABC [17]	25	305	20	20	89	Yes
DFFA [63]	25	325	0	0	89	Yes
HDABC [95]	25	325	0	0	89	Yes
DSCA [52]	25	325	0	0	89	Yes
DPIO [48]	22	253	0	0	65	Yes
SOM [97]	22	253	0	0	65	Yes
D-CLPSO [46]	14	105	0	0	21	Yes
CLK [85]	15	90	30	30	25	NO

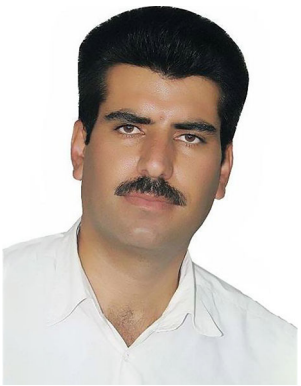
optimization problems such as vehicle routing and scheduling and mixed-integer programming problems. Different types of TSP instances can also be evaluated, including asymmetric, spherical, and generalized TSP.

References

1. Vahdat-Nejad, H., Navabi, M.S., Khosravi-Mahmouei, H.: A context-aware museum-guide system based on cloud computing. Int. J. Cloud Appl. Comput. (IJCAC) **8**(4), 1–19 (2018)
2. Sarrab, M., Alshohoumi, F.: Assisted-fog-based framework for iot-based healthcare data preservation. Int. J. Cloud Appl. Comput. (IJCAC) **11**(2), 1–16 (2021)
3. Hossain, K., Rahman, M., Roy, S.: Iot data compression and optimization techniques in cloud storage: current prospects and future directions. Int. J. Cloud Appl. Comput. (IJCAC) **9**(2), 43–59 (2019)
4. Kapgate, D.: Predictive data center selection scheme for response time optimization in cloud computing. Int. J. Cloud Appl. Comput. (IJCAC) **11**(1), 93–111 (2021)
5. Aliyu, M., et al.: Efficient Metaheuristic Population-Based and Deterministic Algorithm for Resource Provisioning Using Ant Colony Optimization and Spanning Tree. International Journal of Cloud Applications and Computing (IJCAC) **10**(2), 1–21 (2020)
6. Lawler, E.L., Wood, D.E.: Branch-and-bound methods: a survey. Oper. Res. **14**(4), 699–719 (1966)
7. Padberg, M., Rinaldi, G.: Optimization of a 532-city symmetric traveling salesman problem by branch and cut. Oper. Res. Lett. **6**(1), 1–7 (1987)
8. Barnhart, C., et al.: Branch-and-price: COLUMN generation for solving huge integer programs. Oper. Res. **46**(3), 316–329 (1998)
9. Laporte, G., Nobert, Y.: A cutting planes algorithm for the m-salesmen problem. J. Operat. Res. Soc. **31**(11), 1017–1023 (1980)
10. Laporte, G.: The traveling salesman problem: an overview of exact and approximate algorithms. Eur. J. Oper. Res. **59**(2), 231–247 (1992)
11. Abdollahzadeh, B., Gharehchopogh F.S.: A multi-objective optimization algorithm for feature selection problems. Eng. Comput. (2021) pp. 1-19.
12. Meng, Q., Zhang, J.: Optimization and application of artificial intelligence routing algorithm. Clust. Comput. **22**(4), 8747–8755 (2019)
13. Gharehchopogh, F.S., Maleki, I., Dizaji, Z.A.: Chaotic vortex search algorithm: metaheuristic algorithm for feature selection. Evol. Intell. (2021) p. 1-32.
14. Abedi, M., Gharehchopogh, F.S.: An improved opposition based learning firefly algorithm with dragonfly algorithm for solving continuous optimization problems. Intell. Data Anal. **24**(2), 309–338 (2020)
15. Cao, Y., et al.: An improved global best guided artificial bee colony algorithm for continuous optimization problems. Clust. Comput. **22**(2), 3011–3019 (2019)
16. Liu, W., et al.: Improved artificial bee colony algorithm based on self-adaptive random optimization strategy. Clust. Comput. **22**(2), 3971–3980 (2019)
17. Choong, S.S., Wong, L.-P., Lim, C.P.: An artificial bee colony algorithm with a modified choice function for the traveling salesman problem. Swarm Evol. Comput. **44**, 622–635 (2019)
18. Drake, J.H., Özcan, E., Burke, E.K.: Modified choice function heuristic selection for the multidimensional knapsack problem. In: Genetic and Evolutionary Computing, pp. 225–234. Springer (2015)
19. Denzinger, J.r. and M. Fuchs, *High performance ATP systems by combining several AI methods.* 1996.
20. Burke, E.K., et al.: A classification of hyper-heuristic approaches. In: Handbook of metaheuristics, pp. 449–468. Springer (2010)
21. Burke, E.K., et al.: Hyper-heuristics: a survey of the state of the art. J. Operat. Res. Soc. **64**(12), 1695–1724 (2013)
22. Cowling, P., Kendall, G., Soubeiga, E.: A hyperheuristic approach to scheduling a sales summit. In International Conference on the Practice and Theory of Automated Timetabling. Springer, (2000)
23. Drake, J.H., Ã–zcan, E., Burke, E.K.: An improved choice function heuristic selection for cross domain heuristic search. in International Conference on Parallel Problem Solving from Nature. Springer, (2012)
24. Drake, J.H., Ã–zcan, E., Burke, E.K.: A modified choice function hyper-heuristic controlling unary and binary operators. in 2015 IEEE Congress on Evolutionary Computation (CEC). IEEE, (2015)
25. Ã–zcan, E., et al.: A reinforcement learning: great-deluge hyper-heuristic for examination timetabling, In Modeling, Analysis, and Applications in Metaheuristic Computing: Advancements and Trends. pp. 34-55, IGI Global, (2012)
26. FalcÃ£o, D., Madureira, A., Pereira, I.: Q-learning based hyper-heuristic for scheduling system self-parameterization. in 2015 10th Iberian Conference on Information Systems and Technologies (CISTI). IEEE, (2015)
27. Lin, J., Wang, Z.-J., Li, X.: A backtracking search hyper-heuristic for the distributed assembly flow-shop scheduling problem. Swarm Evol. Comput. **36**, 124–135 (2017)
28. Dempster, P., Drake, J.H.: Two frameworks for cross-domain heuristic and parameter selection using harmony search. In: Harmony Search Algorithm, pp. 83–94. Springer (2016)
29. Zamli, K.Z., Alkazemi, B.Y., Kendall, G.: A tabu search hyper-heuristic strategy for t-way test suite generation. Appl. Soft Comput. **44**, 57–74 (2016)
30. Jackson, W.G., Ã–zcan, E., Drake, J.H.: Late acceptance-based selection hyper-heuristics for cross-domain heuristic search. In

2013 13th UK Workshop on Computational Intelligence (UKCI). IEEE, (2013)
31. Kalender, M., et al.: A greedy gradient-simulated annealing selection hyper-heuristic. Soft. Comput. **17**(12), 2279–2292 (2013)
32. Chakhlevitch, K., Cowling, P.: Hyperheuristics: recent developments. In: Adaptive and multilevel metaheuristics, pp. 3–29. Springer (2008)
33. Heidari, A.A., et al.: Harris hawks optimization: algorithm and applications. Futur. Gener. Comput. Syst. **97**, 849–872 (2019)
34. Abd Elaziz, M., et al.: A competitive chain-based Harris Hawks Optimizer for global optimization and multi-level image thresholding problems. Appl. Soft Comput. 106347 (2020)
35. Hussain, K., Zhu, W., Salleh, M.N.M.: Long-term memory Harrisâ€™M hawk optimization for high dimensional and optimal power flow problems. IEEE Access **7**, 147596–147616 (2019)
36. Zhang, Y., et al.: Boosted binary Harris hawks optimizer and feature selection. Structure **25**, 26 (2020)
37. Hans, R., Kaur, H., Kaur, N.: Opposition-based Harris Hawks optimization algorithm for feature selection in breast mass classification. J. Interdis. Math. **23**(1), 97–106 (2020)
38. Abdel-Basset, M., Ding, W., El-Shahat, D.: A hybrid Harris Hawks optimization algorithm with simulated annealing for feature selection. Artif. Intell. Rev. 1–45 (2020)
39. Jia, H., et al.: Dynamic harris hawks optimization with mutation mechanism for satellite image segmentation. Remote Sensing **11**(12), 1421 (2019)
40. Abbasi, A., B. Firouzi, and P. Sendur, *On the application of Harris hawks optimization (HHO) algorithm to the design of microchannel heat sinks.* Engineering with Computers, 2019: p. 1-20.
41. Reinelt, G.: TSPLIB http://www.iwr.uni-heidelberg.de/groups/comopt/software/TSPLIB95/. (1991)
42. Gharehchopogh, F.S., Gholizadeh, H.: A comprehensive survey: whale optimization algorithm and its applications. Swarm Evol. Comput. **48**, 1–24 (2019)
43. Gharehchopogh, F.S., Shayanfar, H. Gholizadeh, H.: A comprehensive survey on symbiotic organisms search algorithms. Artif. Intell. Rev. 1-48 (2019)
44. Gülcü, S., et al.: A parallel cooperative hybrid method based on ant colony optimization and 3-Opt algorithm for solving traveling salesman problem. Soft. Comput. **22**(5), 1669–1685 (2018)
45. Ezugwu, A.E.-S., A.O. Adewumi, and M.E. FrÃ®ncu, *Simulated annealing based symbiotic organisms search optimization algorithm for traveling salesman problem.* Expert Systems with Applications, 2017. **77**: p. 189-210.
46. Zhong, Y., et al.: Discrete comprehensive learning particle swarm optimization algorithm with Metropolis acceptance criterion for traveling salesman problem. Swarm Evol. Comput. **42**, 77–88 (2018)
47. Sun, Y., et al.: A new wolf colony search algorithm based on search strategy for solving travelling salesman problem. Int. J. Comput. Sci. Eng. **18**(1), 1–11 (2019)
48. Zhong, Y., et al.: Discrete pigeon-inspired optimization algorithm with Metropolis acceptance criterion for large-scale traveling salesman problem. Swarm Evol. Comput. **48**, 134–144 (2019)
49. Dong, R., et al.: Hybrid optimization algorithm based on wolf pack search and local search for solving traveling salesman problem. J. Shanghai Jiaotong Univ. (Science) **24**(1), 41–47 (2019)
50. Pook, M.F., Ramlan, E.I.: The Anglerfish algorithm: a derivation of randomized incremental construction technique for solving the traveling salesman problem. Evol. Intel. **12**(1), 11–20 (2019)
51. Wang, Y., Wu, Y., Xu, N.: Discrete symbiotic organism search with excellence coefficients and self-escape for traveling salesman problem. Comput. Ind. Eng. **131**, 269–281 (2019)
52. Tawhid, M.A., Savsani, P.: Discrete sine-cosine algorithm (DSCA) with local search for solving traveling salesman problem. Arab. J. Sci. Eng. **44**(4), 3669–3679 (2019)
53. Karaboga, D., Gorkemli, B.: Solving traveling salesman problem by using combinatorial artificial bee colony algorithms. Int. J. Artif. Intell. Tools **28**(01), 1950004 (2019)
54. Sahana, S.K.: Hybrid optimizer for the travelling salesman problem. Evol. Intel. **12**(2), 179–188 (2019)
55. Jiang, C., Wan, Z., Peng, Z.: A new efficient hybrid algorithm for large scale multiple traveling salesman problems. Expert Syst. Appl. **139**, 112867 (2020)
56. Ali, I.M., Essam, D., Kasmarik, K.: A novel design of differential evolution for solving discrete traveling salesman problems. Swarm Evol. Comput. **52**, 100607 (2020)
57. Ebadinezhad, S.: DEACO: Adopting dynamic evaporation strategy to enhance ACO algorithm for the traveling salesman problem. Eng. Appl. Artif. Intell. **92**, 103649 (2020)
58. Cinar, A.C., Korkmaz, S., Kiran, M.S.: A discrete tree-seed algorithm for solving symmetric traveling salesman problem. Eng. Sci. Technol. Int. J. **23**(4), 879–890 (2020)
59. George, T., Amudha, T.: Genetic algorithm based multi-objective optimization framework to solve traveling salesman problem. In: Advances in Computing and Intelligent Systems, pp. 141–151. Springer (2020)
60. Yousefikhoshbakht, M., *Solving the Traveling Salesman Problem: A Modified Metaheuristic Algorithm.* Complexity, 2021. **2021**.
61. Huang, Y., Shen, X.-N., You, X.: A discrete shuffled frog-leaping algorithm based on heuristic information for traveling salesman problem. Appl. Soft Comput. **102**, 107085 (2021)
62. Al-Gaphari, G.H., Al-Amry, R., Al-Nuzaili, A.S.: Discrete crow-inspired algorithms for traveling salesman problem. Eng. Appl. Artif. Intell. **97**, 104006 (2021)
63. Benyamin, A., Farhad, S.G., Saeid, B.: Discrete farmland fertility optimization algorithm with metropolis acceptance criterion for traveling salesman problems. Int. J. Intell. Syst. **36**(3), 1270–1303 (2021)
64. Bean, J.C.: Genetic algorithms and random keys for sequencing and optimization. ORSA J. Comput. **6**(2), 154–160 (1994)
65. Das, S., Mullick, S.S., Suganthan, P.N.: Recent advances in differential evolutionâ€“an updated survey. Swarm Evol. Comput. **27**, 1–30 (2016)
66. Abd Elaziz, M., et al.: Task scheduling in cloud computing based on hybrid moth search algorithm and differential evolution. Knowl.-Based Syst. **169**, 39–52 (2019)
67. Jadon, S.S., et al.: Hybrid artificial bee colony algorithm with differential evolution. Appl. Soft Comput. **58**, 11–24 (2017)
68. Xiong, G., et al.: Parameter extraction of solar photovoltaic models by means of a hybrid differential evolution with whale optimization algorithm. Sol. Energy **176**, 742–761 (2018)
69. Pan, Q.-K., et al.: A discrete artificial bee colony algorithm for the lot-streaming flow shop scheduling problem. Inf. Sci. **181**(12), 2455–2468 (2011)
70. Szeto, W.Y., Wu, Y., Ho, S.C.: An artificial bee colony algorithm for the capacitated vehicle routing problem. Eur. J. Oper. Res. **215**(1), 126–135 (2011)
71. Liu, X., Su, J., Han, Y.: An improved particle swarm optimization for traveling salesman problem. In International Conference on Intelligent Computing. Springer, (2007)
72. Wang, K.-P., et al.: Particle swarm optimization for traveling salesman problem. In Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693). IEEE, (2003)
73. Irnich, S., Funke, B., GrÃnert, T.: Sequential search and its application to vehicle-routing problems. Comp. Operat. Res. **33**(8), 2405–2429 (2006)

74. Sha, D., Hsu, C.-Y.: A hybrid particle swarm optimization for job shop scheduling problem. Comput. Ind. Eng. **51**(4), 791–808 (2006)
75. KÄran, M.S., ÄÅŸcan, H.M., GÃndÃz, M.: The analysis of discrete artificial bee colony algorithm with neighborhood operator on traveling salesman problem. Neural Comput. Appl. **23**(1), 9–21 (2013)
76. Cowling, P., Kendall, G., Soubeiga, E.: A parameter-free hyperheuristic for scheduling a sales summit. in Proceedings of the 4th Metaheuristic International Conference, MIC. Citeseer, (2001)
77. Croes, G.A.: A method for solving traveling-salesman problems. Oper. Res. **6**(6), 791–812 (1958)
78. Lin, S.: Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. **44**(10), 2245–2269 (1965)
79. Lin, S., Kernighan, B.W.: An effective heuristic algorithm for the traveling-salesman problem. Oper. Res. **21**(2), 498–516 (1973)
80. MartÃ-, R., et al.: Multi-start methods. Handbook of Heuristics, p. 1-21 (2016)
81. Li, W.: A parallel multi-start search algorithm for dynamic traveling salesman problem. In International Symposium on Experimental Algorithms. Springer, (2011)
82. Avci, M., Topaloglu, S.: A multi-start iterated local search algorithm for the generalized quadratic multiple knapsack problem. Comput. Oper. Res. **83**, 54–65 (2017)
83. Michallet, J., et al.: Multi-start iterated local search for the periodic vehicle routing problem with time windows and time spread constraints on services. Comput. Oper. Res. **41**, 196–207 (2014)
84. LourenÃo, H.R., Martin, O.C., StÃtzle, T.: Iterated local search: Framework and applications, In Handbook of metaheuristics. Springer. p. 129-168 (2019)
85. Applegate, D., Cook, W., Rohe, A.: Chained Lin-Kernighan for large traveling salesman problems. INFORMS J. Comput. **15**(1), 82–92 (2003)
86. Martin, O., Otto, S.W., Felten, E.W.: Large-step Markov chains for the traveling salesman problem. Citeseer, (1991)
87. Kramer, R., Dell'Amico, M., Iori, M.: A batching-move iterated local search algorithm for the bin packing problem with generalized precedence constraints. Int. J. Product. Res. **55**(21), 6288–6304 (2017)
88. Subramanian, A., Battarra, M., Potts, C.N.: An iterated local search heuristic for the single machine total weighted tardiness scheduling problem with sequence-dependent setup times. Int. J. Prod. Res. **52**(9), 2729–2742 (2014)
89. Dong, X., et al.: Self-adaptive perturbation and multi-neighborhood search for iterated local search on the permutation flow shop problem. Comput. Ind. Eng. **87**, 176–185 (2015)
90. Nguyen, S., et al.: Automatic programming via iterated local search for dynamic job shop scheduling. IEEE Trans. Cybernet. **45**(1), 1–14 (2014)
91. Li, J., et al.: Iterated local search embedded adaptive neighborhood selection approach for the multi-depot vehicle routing problem with simultaneous deliveries and pickups. Expert Syst. Appl. **42**(7), 3551–3561 (2015)
92. Silva, M.M., Subramanian, A., Ochi, L.S.: An iterated local search heuristic for the split delivery vehicle routing problem. Comput. Oper. Res. **53**, 234–249 (2015)
93. Herrera, B.A.L.D.M., Coelho, L.D.S., Steiner, M.T.A.: Quantum inspired particle swarm combined with lin-kernighan-helsgaun method to the traveling salesman problem. Pesquisa Operacional **35**(3), 465–488 (2015)
94. Wilcoxon, F., Katti, S., Wilcox, R.A.: Critical values and probability levels for the Wilcoxon rank sum test and the Wilcoxon signed rank test. Select. Tables Math. Statist. **1**, 171–259 (1970)
95. Zhong, Y., et al.: Hybrid discrete artificial bee colony algorithm with threshold acceptance criterion for traveling salesman problem. Inf. Sci. **421**, 70–84 (2017)
96. Rey, D., Neuhäuser, M.: Wilcoxon-signed-rank test. In: International encyclopedia of statistical science, pp. 1658–1659. Springer, Berlin, Heidelberg (2011)
97. Wang, H., Zhang, N., CrÃput, J.-C.: A massively parallel neural network approach to large-scale Euclidean traveling salesman problems. Neurocomputing **240**, 137–151 (2017)



Farhad Soleimanian Gharehchopogh is an Assistant Professor in the Department of Computer Engineering at Urmia Branch, Islamic Azad University, Urmia, IRAN, where he has been since 2015. From 2016 to 2018 he served as Department Chair. He received a B.S. in Computer Engineering from Shabestar Branch, Islamic Azad University, in 2002, and an M.S. in Computer Engineering from the Cukurova University at Adana, Turkey in 2011. He received his Ph.D. in Computer Engineering from the Hacettepe University in 2015. His research interests span both Data Mining and Natural Language Processing. Much of his work has been on improving the understanding, design, and performance of metaheuristic algorithms, mainly through the application of NLP, Complex Network, Clustering and Classification, and etc. https://scholar.google.com/citations?hl=en&user=hLUbLLsAAAAJ&view_op=list_works&authuser=1&sortby=pubdate. Email: bonab.farhad@gmail.com

Benyamin Abdollahzadeh received the M.Sc. degree in Computer Engineering from Islamic Azad University, Urmia, Iran, in 2019. He is member of IEEE society as a student. His research interest includes Optimization, Clustering, and Classification. Email: benyamni.abdolahzade@gmail.com