International Journal of Computational Intelligence Systems (2024) 17:4
https://doi.org/10.1007/s44196-023-00385-5

RESEARCH ARTICLE

Solving Traveling Salesman Problem Using Parallel River Formation Dynamics Optimization Algorithm on Multi-core Architecture Using Apache Spark

Esra'a Alhenawi[1] · Ruba Abu Khurma[2] · Robertas Damaševičius[3] · Abdelazim G. Hussien[4,5,6]

Received: 7 June 2023 / Accepted: 2 December 2023

Abstract
According to Moore's law, computer processing hardware technology performance is doubled every year. To make effective use of this technological development, the algorithmic solutions have to be developed at the same speed. Consequently, it is necessary to design parallel algorithms to be implemented on parallel machines. This helps to exploit the multi-core environment by executing multiple instructions simultaneously on multiple processors. Traveling Salesman (TSP) is a challenging non-deterministic-hard optimization problem that has exponential running time using brute-force methods. TSP is concerned with finding the shortest path starting with a point and returning to that point after visiting the list of points, provided that these points are visited only once. Meta-heuristic optimization algorithms have been used to tackle TSP and find near-optimal solutions in a reasonable time. This paper proposes a parallel River Formation Dynamics Optimization Algorithm (RFD) to solve the TSP problem. The parallelization technique depends on dividing the population into different processors using the Map-Reduce framework in Apache Spark. The experiments are accomplished in three phases. The first phase compares the speedup, running time, and efficiency of RFD on 1 (sequential RFD), 4, 8, and 16 cores. The second phase compares the proposed parallel RFD with three parallel water-based algorithms, namely the Water Flow algorithm, Intelligent Water Drops, and the Water Cycle Algorithm. To achieve fairness, all algorithms are implemented using the same system specifications and the same values for shared parameters. The third phase compares the proposed parallel RFD with the reported results of metaheuristic algorithms that were used to solve TSP in the literature. The results demonstrate that the RFD algorithm has the best performance for the majority of problem instances, achieving the lowest running times across different core counts. Our findings highlight the importance of selecting the most suitable algorithm and core count based on the problem characteristics to achieve optimal performance in parallel optimization.

Keywords Traveling salesman problem · River formation dynamics algorithm · Optimization · Parallel architecture

1 Introduction

The Traveling Salesman Problem (TSP) is an operational research problem that was formulated by the mathematician Karl Menger in 1930 [1]. It is an extension of the Hamiltonian cycle problem as it concerns on finding a simple path that starts and ends at the same node with minimum cost [2]. TSP is considered an NP-hard problem, and after many

✉ Abdelazim G. Hussien
abdelazim.hussien@liu.se ; aga08@fayoum.edu.eg

Esra'a Alhenawi
ealhenawi@zu.edu.jo

Ruba Abu Khurma
rubaabukhurma82@gmail.com

[1] Faculty of Information Technology, Zarqa University, Zarqa, Jordan

[2] MEU Research Unit, Faculty of Information Technology, Middle East University, Amman 11831, Jordan

[3] Department of Applied Informatics, Vytautas Magnus University, Kaunas, Lithuania

[4] Department of Computer and Information Science, Linköping University, Linköping, Sweden

[5] Faculty of Science, Fayoum University, Faiyum, Egypt

[6] Applied science research center, Applied science private university, Amman 11931, Jordan

Published online: 03 January 2024

experiments, it has been found that it is tough to solve in a polynomial time algorithm [3]. Many types of research have been accomplished to find an efficient heuristics [4, 5].
Metaheuristics, in a broader sense, serve as versatile problem-solving methodologies, particularly for complex optimization challenges [6–8]. These techniques offer a flexible and adaptive approach to navigating through solution spaces without the need for explicit problem-specific information. By leveraging their capacity for exploration and exploitation, metaheuristics efficiently tackle NP-hard problems, such as the Traveling Salesman Problem (TSP), where finding an optimal solution in a reasonable time is a formidable task. They embody an array of algorithms inspired by natural and physical phenomena, swarm behaviors, and mathematical principles. These methods, which include Simulated Annealing, Genetic Algorithms, Whale Optimization Algorithm, Particle Swarm Optimization, Snake Optimizer (SO) [9], dandelion optimizer (DO) [10], Jellyfish Search (JS) [11], Aquila Optimizer (AO) [12, 13], Reptile Search Algorithm (RSA) [14], Golden Jackal Optimization (GJO) [15], Smell Agent Optimization (SAO) [16], and tabu search, among others, excel in addressing a wide spectrum of optimization problems by iteratively refining solutions. Their strength lies in their ability to balance local and global search, adapting to diverse problem landscapes, and delivering near-optimal solutions within practical time constraints. The adaptability and versatility of metaheuristics have made them indispensable tools in various domains, offering efficient solutions where exact algorithms fall short [17–20].
In the literature, several metaheuristics (MAs) have been introduced to solve the TSP problem [21–23]. In another study, Geng et al. suggested an efficient local search algorithm based on simulated annealing and greedy search approaches for solving TSP. To find more efficient solutions, the SA algorithm was combined with three different types of mutations and the greedy search method. A genetic algorithm with the avoidance of particular crossover and mutation for TSP was proposed by Üçoluk [24].

A fresh use of PSO for TSP was suggested by Wang et al. [25]. They redefined certain operators and created some unique techniques using the swap operator and swap sequence. To represent the position and velocity of the particles and solve TSP, Pang et al. [26] presented a modified discrete PSO algorithm with integrated fuzzy matrices. A unique particle swarm optimization (PSO)-based approach for TSP was proposed by Shi et al. in a different paper [27]. They sped up convergence using a crossover elimination method and an ambiguous searching strategy.

To solve TSP, Chen and Chien [28] developed a hybrid strategy based on GA, SA, and ACO with PSO approaches. They used 25 TSPLIB examples for their investigation and compared them to some evolutionary optimization techniques. A new hybrid approach based on particle swarm optimization, ant colony optimization, and 3-opt local search algorithms was proposed by Mahi et al. in a different study to solve TSP. In their study, they employed PSO to find the best values for the parameters α and β, which were used in the ACO algorithm's city selection operations and determine the importance of inter-city pheromone and distances. 3-opt in computer algorithm was used to shorten the tour after the termination condition was met [29].

Dorigo and Gambardella suggested an ant colony system (ACS) and ACS-3-opt for solutions to symmetric and asymmetric TSP in their 1965 computer [29]. Authors in [5] used Meerkat Clan Algorithm for solving the Capacitated Vehicle Routing Problem. [4] solved TSP using a traditional optimization method and clustering phase of Cluster-First Route-Second Method.

Combinatorial ABC (CABC), a brand-new artificial bee colony algorithm that Karaboga and Gorkemli [30] suggested for solving TSP. In a different study, Görkemli and Karaboga suggested the fast Artificial Bee Colony (qABC), an upgraded CABC in which the behavior of the spectator bees is more precisely approximated.

The most common MAs for solving TSP are Ant colony optimization (ACO) [31–33], and genetic algorithms (GAs) [34–36]. In ACO, several solutions are generated, and then, edges that have been picked most of the time have a higher probability of being part of future solutions. Thus, as an effect, multiple solutions were repeatedly generated to tune the heuristics to generate a good tour. GAs also generate a large number of solutions, but instead of optimizing the heuristic for generating the tours, they have a sophisticated merging procedure to incorporate the good characteristics of different tours into one tour.

Adewole in [37] used the GA to solve the TSP problem using several steps. In the first step, he created a random number of routes, and then evaluated the fitness value for each route and selected the best route. In the next step, he reproduced two routers from the best routes, generated a new population, and replaced the two worst with the new routes. Results demonstrated that GA produced a good result for finding an optimum router for the TSP. The authors in [38] described how TSP can be solved using the heuristic method of GA. They generated a fitness formula with operators like selection, crossover, and mutation to get the optimal path. They used Matlab as the front end and for plotting graphs. The graph plotted showed the exact path and total distance covered by the salesman and showed the number of cities traversed. They demonstrated how efficiently the GA worked for the TSP. This was done by creating a solution without having any prior knowledge about the problem using the concepts of statistics and probability. It was also exceedingly helpful in speeding up the search process.

Later in the literature, Parallel MAs have been proposed to utilize the multi-core technology and produce promis-

ing results regarding the performance of TSP. Implementing sequential MAs in a multi-core system is inefficient. This opens a way for multi-threading technique which means executing multiple parts of a program at the same time. Reducing the time complexity is highly probable when implementing multiple computations on multiple cores inside the same processor chip. However, many factors affect the design of a parallel MA such as the time latency between cores, the synchronization time, and the memory design. The design of a good parallel algorithm should consider these factors to alleviate their bad influence.

Also, the most common parallel versions of MAs for solving TSP are Ant colony optimization (ACO) [39, 40] and genetic algorithms (GAs) [41, 42]. In [43], the authors parallelized the sequential ACO algorithm over the Apache Spark framework to improve the performance of TSP. Their design of the Map-Reduce job involved how the map and reduce functions are implemented. Also, how the ACO algorithm has been modified to be able to run in a Map-Reduce framework. The basic idea behind their proposed algorithm was to give copies of the TSP instances to multiple mappers, which will work in parallel to produce the results.

They divided the implementation of the ACO into three stages: the initial stage in which the input to the Map-Reduce job contains the Euclidean two-dimensional coordinates of a TSP problem. Furthermore, the reducer performs updates to the global pheromone array. This will be passed on to the next stage to calculate the best route through each stage. In the intermediate stage, a modified mapper and reducer that work on input in the binary form are applied using the output obtained from the reducer of the initial stage. In the final stage, the reducer compares the solutions of different maps in the stage and finds the best possible one, writing them into a file for the user to access. The proposed ACO took the number of mappers per stage, number of ants per mapper, and number of stages as inputs to display the time for algorithm execution. Also, to compute the shortest path. The results showed that the approximate solutions were quite close to the optimal solution. The obtained lowest error rate was between two and 53.

A study proposed by Hlaing [44] presented an improved ACO algorithm with two approaches. The first approach is called the dynamic candidate list strategy, where the authors capture a suitable number of nodes based on the total number of nodes. When the ant chooses the next city, it computes the probability of the transfer from the current city to the next city. Then, the city whose transfer probability is first needed to consider those preferred cities listed in the candidate list. Their objectives are to get provably optimal or at least close to optimal solutions for the TSP problem. However, the run time of the sequential MAs increases with an increasing number of cities. This causes time complexity to become extremely high, especially when solving large-scale TSP problems.

A parallel cooperative hybrid algorithm (PACO-3opt) with ACO and 3-opt was presented by Gülcü et al. at [45]. After a predetermined number of repetitions, 3-opt is applied to each colony to enhance the solutions, and the best tour is then shared with other colonies. Initially, PACO was developed using TSP. Their research indicated that PACO-3opt performed more effectively and consistently. A hierarchical technique based on ACO and ABC was proposed by Gündüz et al. in a different study to solve the TSP. A superior initial answer for the path improvement-based methodology ABC was provided in their study using the path construction-based method ACO. To create a better discrete artificial bee colony method, Li et al. [46] used the idea of the swap operator.

In [47], the authors solved TSP in two ways using the sequential GA. Then, they parallelized the algorithm using a static population with migration as the pluralization method. Then, they implemented it on the Spark Cluster to solve TSP. They tried to find a near-optimal solution in an acceptable time using an algorithm consisting of five stages: initial population (creating a list of the tours), fitness evaluation, selection, crossover (using the greedy crossover method), and mutation.

In [48], authors developed parallel versions of both ACO, and GA for solving TSP. They generated data from the wireless sensor networks. Results show that both methods reduce the time in half, while GA outperformed ACO.

Recently, several water-based metaheuristic algorithms have appeared to solve TSP. These algorithms include IWD [49, 50] and RFD [51].

RFD was first proposed in 2007, and used in many applications since that day like in telecommunications, software testing, industrial manufacturing processes, and others. It is a distinguished water-based algorithm that has several remarkable features such that it has a natural avoidance of cycles, and the focused elimination of blind alleys [52]. It was inspired by the way that water is used to form rivers. The water drops at decreasing or increasing altitudes at specific places. This satisfies the fact that water moves from higher altitude places to lower altitude places by eroding the ground and depositing sediments. Solutions are represented by paths that consist of a sequence of decreasing altitude nodes. These nodes represent cities in the TSP, which forms a decreasing gradient that is followed by the next water drop. This is to reinforce the best paths with the lowest cost and eliminate the worst paths.

In [51], authors deployed RFD for solving dynamic TSP. They compared their results with results of sequential Ant Colony Optimization (ACO) algorithm when it used for solving TSP. Their results show that ACO is faster than RFD for static and dynamic TSP where RFD works slower in static cases compared to its performance in dynamic cases.

Afaq et al. compared the performance of six natural computing algorithms like River Formation Dynamics (RFD), Intelligent Water Drops (IWD), Bacterial Foraging Opti-

mization Algorithm (BFOA), Ant Colony Optimization (ACO), Gravitational Search Algorithm (GSA), and Particle Swarm Optimization (PSO) for solving TSP. They found that a hybrid algorithms of GA and GSA provides good solutions, while BFOA is one of the algorithms that provides worst results compared to others like IWD [53].

The main recommendation of the papers was to develop parallel versions of these water-based algorithms. The target is to provide better levels of accuracy and speedup in solving the TSP. To our knowledge, there are no previous works in the literature that tried to use parallel version of RFD to solve TSP using Apache Spark. Therefore, we are aware in this paper to solve TSP using a parallel version of this algorithm and comparing its performance with three parallel versions of a water-based algorithms.

The main contributions of this paper are:

- Propose an adaptation for parallelism to solve the TSP using the RFD algorithm using the Apache Spark framework.
- Analyze the running time, accuracy, efficiency, and speedup of RFD on a multi-core environment using 1, 4, 8, and 16 cores. Implementing the one core represents the sequential RFD.
- Conduct a comprehensive analysis of the parallel versions of other water-based paradigms. Then compare them with the proposed parallel RFD in terms of all the evaluation measures.
- Compare the proposed parallel RFD with the reported results of another metaheuristic that was adapted to solve TSP.

The remaining parts of the paper are structured as follows: Sect. 2 describes the TSP problem. Section 3 provides the mathematical methodology of the RFD algorithm. Section 4 is the sequential and parallel adaptation of the RFD to solve TSP. The experiments Setups are given in Sec. 5. In Sec.6 the results of experiments are presented and discussed. A summary of the paper and the main166 recommendation for future works are provided in Sec. 7.

2 Description of the TSP Problem

The Traveling Salesman Problem (TSP) is one of the most well-known problems in combinatorial optimization, belonging to the class of NP-hard problems. It has applications in logistics, planning, scheduling, and many other fields.

Given a list of cities and the distances between each pair of cities, the problem is to find the shortest possible route that visits each city exactly once and returns to the original city. The problem is illustrated in Fig. 1.

Let $G = (V, E)$ be a graph where V is a set of vertices representing cities, and E is a set of edges representing paths between cities. Each edge $(i, j) \in E$ has a non-negative weight w_{ij} representing the distance between city i and city j. A solution to the TSP is a permutation π of the vertex set V, represented as $(v_{\pi(1)}, v_{\pi(2)}, \ldots, v_{\pi(n)})$, where $n = |V|$, which minimizes the total distance (cost) traveled

$$C(\pi) = \sum_{i=1}^{n} w_{v_{\pi(i)}, v_{\pi(i+1)}} + w_{v_{\pi(n)}, v_{\pi(1)}}. \tag{1}$$

The problem then is to find the permutation π that gives the minimum cost

$$\min_{\pi \in S_n} C(\pi), \tag{2}$$

where S_n is the symmetric group of all permutations of n elements.

3 RFD Algorithm

River Formation Dynamics (RFD) is a population-based optimization algorithm, which uses the analogy of the formation of a river system from water drops to find the global optimum solution to a problem.

Consider a population of N drops and a landscape of possible solutions, denoted as $\mathcal{L}$, where the elevation of each point in $\mathcal{L}$ corresponds to the cost function value for that solution. Each drop has a position in the landscape and an associated amount of soil, which increases as it travels and dissolves the soil (moves to higher cost solutions).

Let d_i represent the ith drop and $x_i(t)$ represent the position of d_i at iteration t. Also, let $s_i(t)$ represent the amount of soil carried by drop d_i at iteration t. At the beginning, all drops are assigned random positions, and their soil quantities are initialized to zero, i.e., $x_i(0) \sim U(\mathcal{L})$ and $s_i(0) = 0$.

The behavior of each drop d_i in each iteration is described as follows. First, the drop's next position $x_i'(t)$ is selected randomly among its neighboring positions. The elevation of a position is given by the cost function $f : \mathcal{L} \to \mathbb{R}$ and the quantity of soil is calculated as follows:

$$s_i(t+1) = s_i(t) + \Delta H_i(t), \tag{3}$$

where $\Delta H_i(t) = f(x_i(t)) - f(x_i'(t))$, if $f(x_i(t)) > f(x_i'(t))$ and $\Delta H_i(t) = 0$ otherwise.

The drops move toward the positions with less soil, mimicking the water flow in a river. In the process, they dissolve the soil in the higher regions and deposit it in the lower ones, forming a river that indicates the path to the global optimum.

The landscape's soil update rule is given by

$$f(x_i(t+1)) = f(x_i(t+1)) + s_i(t+1). \tag{4}$$

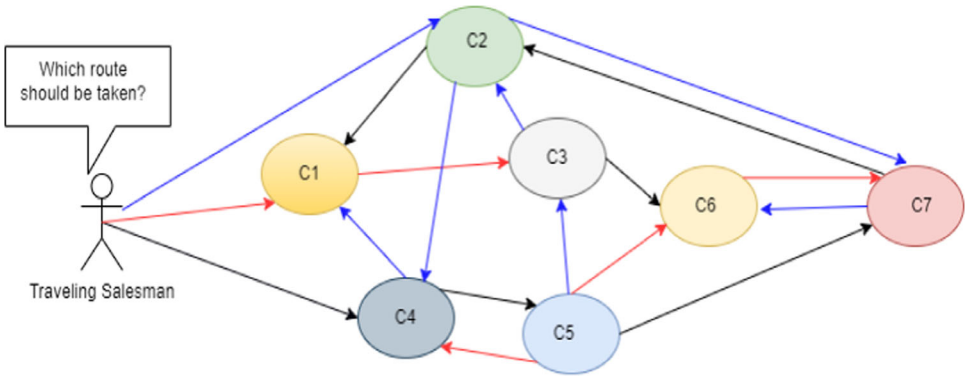

Fig. 1 Traveling salesman problem

The algorithm stops when the maximum number of iterations is reached, or when the solution does not improve significantly for a certain number of iterations.

4 Implementing TSP Using Parallel RFD

In this paper, a parallel version of the RFD algorithm has been developed using Apache Spark Implementation for parallel TSP using Spark goes through the following steps:

1. Preparing Input In this step, we preparing input by specifying two input files one for “nodes" that contain $nid, x, y, altitude$ for each node. Another input file for “drops" that contains $did, nid, erode, deposit, Path$ for each drop. Initially, all nodes’ altitude values are set equal to zero, all drops are on node one ($drop.nid = 1$), and each drop has a path consisting of first_node only 1. Also initially, erode and deposit parameters have a value equal to zero for all drops.
2. Mapping to RDD Nodes After preprocessing the data, the points in each file are mapped into RDD nodes to be accessed by all processors, or machines.
3. Specifying the Number of Iterations We specify the number of iterations. As we increase the number of iterations, we increase the chance of reaching the best path. Each iteration has the following steps.
   - Filtering In this step, the nodes are filtered for retrieving nodes that have drops and distributing these nodes over the available (P) processors.
   - Move Drops In this step, each drop d selects the next node to follow from all current node neighbors except nodes that have been stored in the drop path (means visited nodes) based on the equations below that find the probability of going from the current node (i) to all other unvisited neighbors j

$$p_d(i,j) = \begin{cases} \frac{\text{decreasing}(i,j)}{\sum_{l \in V_d(i)} \text{decreasing}(i,l)}, & If j \in V_d(i) \\ 0 \quad If j \notin V_d(i) \end{cases} \tag{5}$$

$$\text{decreasing}(i,j) = \frac{\text{altitude}(j) - \text{altitude}(i)}{\text{distance}(i,j)}, \tag{6}$$

where $P_d(i,j)$ is the probability that a drop d at a node i chooses the node j from all adjacent nodes of node (i) for moving to it next.
decreasing (i,j) is the differences between node i and node j altitudes divided by the distance between these nodes.
$V_d(i)$ is a set of node i neighbors. Based on this value, there are three cases for the next node selection:

   - Erode Paths In this step, the altitude of the selected node is reduced with a specific value propositional with several iterations to form decreasing gradients that are followed by the next water drops to reinforce the best paths with the lowest cost and eliminate the worst paths. The altitude of the final node which is the first one must remain equal to 0 to represent the sea. In my work, eroding amount in each iteration must be more than its value in the previous iteration to form a decreasing gradient that is followed by the next water drops to reinforce the best paths with the lowest cost and eliminate the worst paths, so i put it equal to the power of iteration number (i^2).
   - Deposit Sediments In this step, for each node that has an altitude smaller than all of its neighboring nodes in this step, we increased its altitude with a value smaller than the erode value to represent the sediments depositing for eliminating the worst paths. Deposit increment value equal to the current iteration number i.
4. Retrieving Best Path After all iterations are finished, we retrieve the shortest path from all discovered paths, where at each N iteration, we store the best path (shortest cost path) where N is the number of cities as we discovered at most D paths from each N iterations where D is the number of drops.

Algorithm 1 Parallel RFD algorithm for solving TSP

Input: Nodes(n_{id}, X, Y, altitude), Drops(d_{id}, n_{id}, erode, deposit, path), $Num\_nodes(N)$, $Max\_iterations\_number(I)$, $Num\_drops(D)$, $first\_node = 1$, $Min\_cost = inf$.
Output: the cost of the shortest simple path from first_node to itself.

$n.altitude \leftarrow 0$ *For all nodes*
$d.n_{id} \leftarrow first\_node, d.erode \leftarrow 0, d.deposit \leftarrow 0, d.path \leftarrow 1$ *For all drops*
Mapping Nodes and Drops files to RDD file
For $(inti = 0; i < I; i++)$ **do**
filter nodes that have drops
divides these nodes between available processors
if $(i\%N == 0)$
For each $(d \in D)$ **do**
if $(d.path.size == N)$
$d.path.cost = d.path.cost + distance(d.path_lastnode, 1)$
$d.n_{id} = first\_node$
$d.path = 1$
$P = min(d.path.cost)$ for all drops
$Min\_cost = min(Min\_cost, P)$
For each $(d \in D)$ **do**
Calculate P(i, j) for all node (i) unvisited neighbor nodes (j)
if $(P(i, j) == 0)$
Select the next node randomly
Update d.path by adding the new visited node
Update d.erode by adding (i^2) to its value
Update node n with $(n.n_{id} == d.n_{id})$ altitude value by adding d.erode value
if $(P(i, j) < 0)$
$d.deposit = i$
Update node n with $(n.n_{id} == d.n_{id})$ altitude value by adding d.deposit value
Start with the next drop
Else
Select the next node using a roulette wheel
Update d.path by adding the new visited node
Update d.erode by adding (i^2) to its value
Update node with $(n.n_{id} == d.n_{id})$ altitude value by adding d.erode value
End For
End For
Print Min_cost
End

5 Experiments Setups

5.1 Apache Spark

Apache Spark is a cluster computing platform that is fast and used for general purposes. It was developed in the UC Berkeley AMP lab based on the Map-Reduce framework using an efficient and concise language called “Scala". Spark has a logical collection of data that is partitioned across available machines, which are called Resilient Distributed Datasets (RDD) files, where input must be allocated to be available for all processors [54].

Spark uses a master–slave model in parallelism by dividing a computer cluster into a master node and multiple worker nodes. The master node is responsible for resource allocation and management in addition to task scheduling, while worker processors perform the map and reduce tasks in parallel. It distributes user programs and data automatically through available processors by its platform [55].

Table 1 RFD parameters

Parameter	Value	Description
NUMBER_OF_DROP	# of Cites	Variate based on the dataset
MAX_ITERATIONS	10,000	Number of iterations
ERODE_INC	0	For node altitude updating. Initially = 0, then it increased by i^2 (power of iteration number)
DEPOSIT_INC	0	For node altitude updating. Initially = 0, then it increased by i (iteration number)

5.2 Testing Environment

Experiments were tested based on the Map-Reduce framework and Spark platform that deployed on Microsoft Azure virtual machine with CPU 4.50 GHz, and RAM 128 GB. All algorithms were assessed over 1 to 16 cores where the final results calculated as an average value of applying each experiment ten times.

For examine and validate the robustness of the proposed parallel RFD algorithm for solving TS Problem, we utilized TSP datasets from various sizes varying between small, medium, and large number of cities. Table 1 displays the main RFD algorithm parameters.

5.3 Datasets

Eight TSP datasets downloaded from the UCI repository, including u1817, rat783, lin318, rd400, d198, ch130, kroA100, and wi29, have been used in this paper for evaluating the proposed framework performance. The selected datasets varied in several cities where each city was represented by its x, y coordination. Table 2 displays these datasets properties.

5.4 Evaluation Measures

This section presents the evaluation metrics that are used for evaluating the proposed method [54].

1. Accuracy: the percentage of retrieving the best route correctly.
2. Running time: is the duration between the end and the beginning of the program running

Running_Time

Table 2 TSP datasets' properties

Dataset name	Dimension	Description
u1817	1817 city	Drilling problem (Reinelt)
rat783	783 city	Rattled grid (Pulleyblank)
rd400	400 city	400 city random TSP (Reinelt)
lin318	318 city	318 city problem (Lin/Kernighan)
d198	198 city	Drilling problem (Reinelt)
ch130	130 city	130 city problem (Churritz)
kroA100 (A100)	100 city	100 city problem A (Krolak/Felts/Nelson)
WI29	29 city	29 locations in Western Sahara

$$= \text{finish\_running\_time} - \text{begin\_running\_time}. \quad (7)$$

3. Speedup: is the improvement in speed of execution of a task executed on two similar architectures with different resources

$$\text{Speed Up} = \frac{T_s}{T_p}, \quad (8)$$

where T_s represents the time in serial, and T_p represents the time in parallel

4. Efficiency: represents the speedup divided by the number of cores

$$\text{Efficiency} = \frac{\text{Speedup}}{p} \quad p : \text{number of available processors.} \quad (9)$$

6 Experimental Results

6.1 Comparison of Parallel RFD with Other Water-Based Algorithms

Water-based optimization algorithms or hydrological optimization algorithms are a group of heuristic optimization methods inspired by natural water behaviors, such as rainfall, evaporation, water flow, etc. Some of these algorithms include WCA, IWD, WFA, and RFD.

WCA [56] is based on the natural water cycle process, including the flow of rivers to the sea, evaporation, rain, and the path followed by water to travel again to the sea. In WCA, solutions are represented as streams that flow toward the sea. The rate at which water flows to the sea is indicative of the fitness of the solution; thus, solutions that represent seas or rivers are of better quality.

Table 3 Accuracy results for target problems

No. of cores	wi29			
	WFA	IWD	RFD	WCA
1	99%	92%	98%	95%
4	99%	95%	99%	97%
8	99%	98%	100%	98%
16	100%	100%	100%	98%
No. of cores	A100			
	WFA	IWD	RFD	WCA
1	72%	63%	94%	65%
4	74%	68%	96%	72%
8	79%	71%	99%	74%
16	80%	75%	100%	78%
No. of cores	ch130			
	WFA	IWD	RFD	WCA
1	63%	61%	91%	72%
4	74%	61%	93%	74%
8	77%	64%	94%	75%
16	79%	66%	97%	80%
No. of cores	d198			
	WFA	IWD	RFD	WCA
1	75%	72%	92%	71%
4	77%	72%	94%	75%
8	81%	77%	94%	77%
16	89%	79%	98%	81%
No. of cores	rd400			
	WFA	IWD	RFD	WCA
1	61%	80%	81%	50%
4	62%	82%	90%	50%
8	62%	85%	94%	54%
16	67%	88%	96%	61%
No. of cores	lin318			
	WFA	IWD	RFD	WCA
1	83%	74%	95%	71%
4	84%	75%	96%	76%
8	84%	75%	96%	79%
16	84%	77%	98%	82%
No. of cores	rat783			
	WFA	IWD	RFD	WCA
1	79%	79%	92%	62%
4	80%	84%	95%	69%
8	84%	87%	97%	71%
16	84%	88%	97%	74%
No. of cores	u1817			
	WFA	IWD	RFD	WCA
1	87%	91%	92%	72%
4	87%	93%	95%	79%
8	87%	96%	97%	77%
16	88%	96%	99%	80%

Table 4 Speedup results for target problems

No. of cores	wi29			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.09	1.14	1.03	1.68
8	1.09	2.13	1.30	2.27
16	1.32	2.12	1.35	2.34
No. of cores	A100			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.26	1.60	1.30	1.92
8	1.27	2.40	1.60	2.56
16	1.44	2.10	2.50	2.58
No. of cores	ch130			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.19	1.20	1.90	2.85
8	1.19	2.40	2.30	2.59
16	1.72	2.10	2.40	2.54
No. of cores	d198			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.30	2.04	1.20	2.31
8	1.34	2.55	1.40	2.81
16	1.51	2.67	1.60	2.87
No. of cores	rd400			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.06	2.10	1.30	1.54
8	1.24	3.20	1.80	4.14
16	1.48	4.90	2.60	7.21
No. of cores	lin318			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.16	1.59	1.20	2.10
8	1.21	1.75	1.30	2.33
16	1.50	1.81	1.62	2.41
No. of cores	rat783			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.03	2.00	1.1000	2.00
8	1.10	2.90	1.200	2.42
16	1.17	2.00	1.40	2.45
No. of cores	u1817			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	1.25	2.38	2.30	1.66
8	1.43	2.80	2.10	2.10
16	1.54	2.11	2.14	2.53

IWD simulates how water points flow in the natural rivers through the best path that has a less soil value. It is a new swarm-based optimization algorithm developed by Shah–Hosseini [57].

WFA inspired behavior of water in nature, in terms of flowing from higher to lower altitudes by the gravity force, and water operations, such as flows splitting, moving, and merging according to the topography of the land, water evaporation, and water precipitation [58].

These algorithms and others like them provide a variety of methods for heuristic optimization. Each method has its strengths and weaknesses and is best suited to different types of problems [59], [60], [61].

We present results as a series of tables showing experimental results for different algorithms (WFA, IWD, RFD, WCA) run on different problems (wi29, A100, ch130, d198, rd400, lin318, rat783, u1817) and with different number of cores (1, 4, 8, 16).

The first set of tables indicates the accuracy achieved by increasing number of cores. Second set of tables is related to speedup values achieved by increasing number of cores used in the computation. The speedup is generally the ratio of the time taken to solve the problem on a single core to the time taken to solve the same problem on multiple cores. A speedup greater than 1 means the algorithm performed better (finished faster) with multiple cores.

Third group of tables displays efficiency. Efficiency is defined as the speedup divided by the number of cores. This indicates how well the algorithm scales with the addition of more cores. The closer the efficiency is to 1, the better the scaling behavior of the algorithm. Table 3 presents accuracy results for target problems. Performance scaling: As the number of cores increases, accuracy increases for all algorithms in various degrees, implying that all the algorithms are capable of exploiting parallel processing to some extent.

Table 4 presents the speedup results for target problems. Performance scaling: As the number of cores increases, all algorithms show some degree of speedup, implying that all the algorithms are capable of exploiting parallel processing to some extent. However, the extent of this speedup varies significantly between algorithms and target problems.

The RFD algorithm appears to have the highest speedup rate as the number of cores increases across most of the target problems. Remarkably, for 'rd400', it attains speedups of 7.21, with 16 cores. This indicates that the RFD algorithm has superior scalability when compared to the others, especially for these particular problems.

In contrast, the WCA and WFA algorithms demonstrate modest speedup growth as the number of cores increases. The speedup factor rarely exceeds 2, which suggests that these algorithms are not effectively utilizing the parallel processing capability provided by the increasing number of cores.

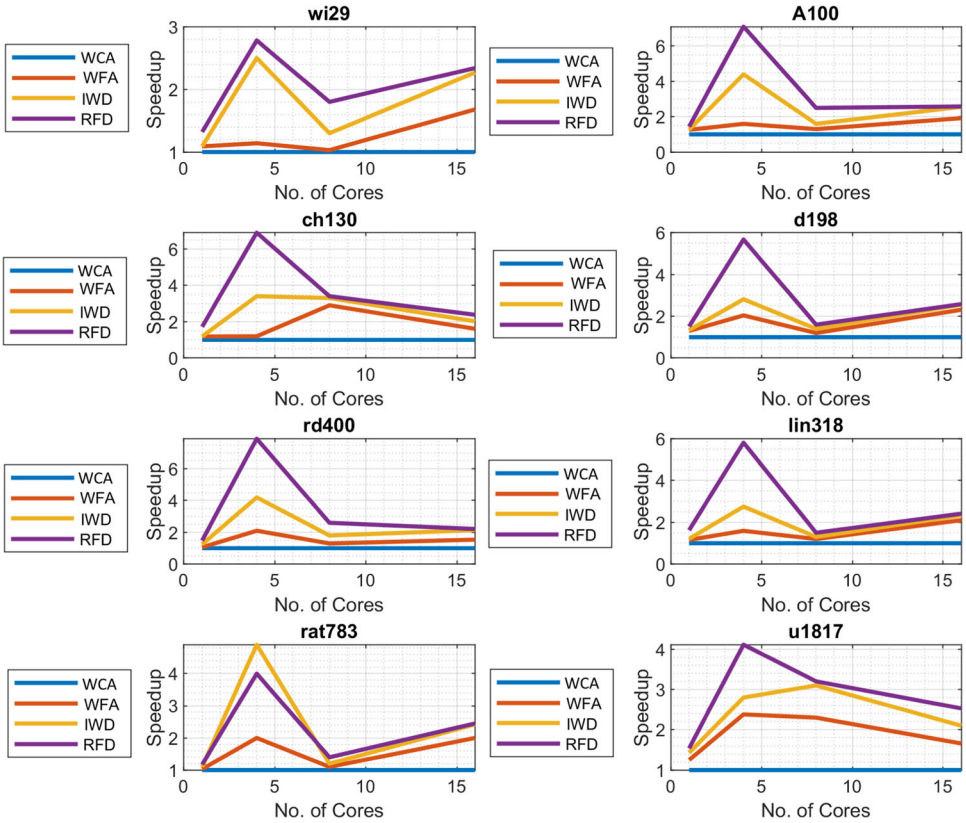

Fig. 2 Visualization of speedup results for target problems

While IWD algorithm does not attain the high speedups demonstrated by RFD, it consistently outperforms WCA and IWD across the board. Interestingly, it shows a substantial speedup even at 4 cores for every target problem.

The degree of speedup is also highly dependent on the specific target problem. For instance, 'ch130' shows the highest speedup for RFD with four cores. This shows that while certain algorithms may, on average, outperform others, the specific nature of the problem being solved is also crucial in determining algorithmic effectiveness.

These results are also visualized in Fig. 2.

Table 5 presents the efficiency results of four algorithms (WCA, IWD, RFD, and WFA) across several target problems (wi29, A100, ch130, d198, rd400, lin318, rat783, and u1817) as the number of cores increases. The efficiency is computed as the speedup divided by the number of cores. A few key observations from the table are as follows:

All four algorithms show a decline in efficiency as the number of cores increases. This is an expected trend as adding more cores often does not result in a linear increase in speedup due to overheads such as communication and synchronization between cores. However, the rate at which efficiency declines varies among the algorithms and target problems.

WCA, RFD, and WFA all have similar trends across all target problems. The efficiency for these algorithms starts at 1.00 with one core and gradually decreases as the number of cores increases. This suggests that although these algorithms are capable of utilizing multiple cores, they are not efficient at higher core counts due to the aforementioned overheads.

The IWD algorithm shows a similar pattern of declining efficiency as the other algorithms but manages to maintain a relatively higher efficiency for several target problems (like A100 and rd400) even as the number of cores increases. Interestingly, for the 'lin318' target problem, the efficiency of all algorithms increases as the number of cores increases. This is an unusual trend as we generally expect efficiency to decrease with an increase in cores. This might suggest a specific characteristic of the 'lin318' problem that allows it to benefit disproportionately from an increase in cores, or it may be a data error.

For 'rat783', the IWD algorithm has a peak efficiency of 0.61 at 8 cores and then drops to 0.25 at 16 cores. This suggests that, for this problem, the overhead costs of using more than 8 cores outweigh the benefits of parallelization for the IWD algorithm.

6.2 Comparison of Parallel RFD with Other Methods from the Recent Literature Research

To illustrate the scientific value added of this paper, and display the applicability of the findings, in this section, we compared the proposed method performance versus some of the most recent research in literature that solve TSP [21] based on the mean tour length, and mean running time was obtained to achieve the best solution. Table 6 displays the most optimal route length, and the mean accuracy for all compared algorithms as accuracy calculate based on the best known route length (BKRL) for each problem. In contrast, Table 7 displays the mean running time results in seconds.

It is obvious from this section that the proposed method provides a competitive accuracy values especially for larger datasets with less running times compared with some methods from the most recent literature called Harris hawk optimization algorithm with modified choice function (MCF) for neighborhood selection named "HHO(MCF)", HHO 5-low-level heuristics "HHO (LLHs)", and "HHO (Random)", as presented in Table 6, and figures from Figs. 3, 4, 5, 7, 8.

6.3 Discussion

This study aimed to analyze and compare the performance of four different algorithms (WCA, IWD, RFD, and WFA) on multiple target problems, with an increasing number of cores. Our metrics of interest were accuracy, running time, speedup, and efficiency. The data collected across these varied contexts have generated insightful findings, as well as giving us some avenues for future research.

As expected, the speedup, in general, tends to improve with an increasing number of cores for all algorithms and target problems. This observation aligns with Amdahl's Law, which states that the maximum improvement to a system's performance is determined by the fraction of the execution time that can be improved. However, the speedup did not scale linearly with the number of cores. Particularly, the RFD algorithm seemed to perform remarkably well in terms of speedup for almost all the target problems. Specifically, in cases such as 'A100', it displayed exponential improvements, achieving a speedup of 7.10 when moving from 1 to 16 cores.

Efficiency, calculated as the ratio of speedup to the number of cores, is crucial in assessing the cost-effectiveness of computational resources. The most efficient algorithm should ideally maintain a ratio near 1 as the number of cores increases. In our results, none of the algorithms managed to maintain high efficiency with the increasing number of cores. While WFA showed promising speedup results, it fell short in terms of efficiency. This implies that although WFA performs well in absolute terms, it may not be as cost-effective as the other algorithms, especially when higher numbers of cores

Table 5 Efficiency results for target problems

No. of cores	wi29			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.27	0.28	0.26	0.42
8	0.14	0.31	0.16	0.28
16	0.08	0.13	0.11	0.17
No. of cores	A100			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.31	0.40	0.33	0.48
8	0.16	0.55	0.20	0.62
16	0.09	0.44	0.52	0.52
No. of cores	ch130			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.30	0.73	0.30	0.40
8	0.15	0.43	0.33	0.46
16	0.22	0.45	0.32	0.47
No. of cores	d198			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.33	0.5100	0.30	0.58
8	0.17	0.3513	0.18	0.32
16	0.09	0.35	0.1000	0.16
No. of cores	rd400			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.26	0.53	0.33	0.39
8	0.16	0.53	0.23	0.27
16	0.09	0.50	0.16	0.14
No. of cores	lin318			
	WFA	IWD	WCA	RFD
1	1.00	1.000	1.00	1.00
4	1.16	1.5934	1.20	2.10
8	1.21	2.75	1.3000	2.33
16	1.62	5.81	1.5000	2.41
No. of cores	rat783			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.26	0.50	0.28	0.50
8	0.14	0.61	0.15	0.30
16	0.07	0.25	0.09	0.15
No. of cores	u1817			
	WFA	IWD	WCA	RFD
1	1.00	1.00	1.00	1.00
4	0.31	0.60	0.58	0.53
8	0.18	0.35	0.39	0.31
16	0.10	0.26	0.20	0.16

Table 6 Comparing RFD mean accuracy with some methods from the recent literature

Dataset	BKRL	HHO (MCF). RL (Acc)	HHO (5-LLH). RL (Acc)	HHO (Random). RL (Acc)	RFD. RL (Acc)
A100	21,282	21,282 (100)	21,282 (100)	21,282 (100)	21,299 (99.9)
ch130	6110	6110 (100)	6110 (100)	6110 (100)	6273 (97.4)
d198	15,780	15,780 (100)	15,780 (100)	15,780 (100)	16,043 (98.3)
u1817	57,201	57326.2 (99.7)	57383.7 (99.6)	57,362 (99.7)	57,216 (99.9)
u2319	234,256	234827.6 (99.7)	234916.7 (99.7)	235,026 (99.6)	234,261 (99.9)
rl5915	565,530	567761.6 (99.6)	572012.3 (99.8)	571,071 (99)	569,742 (99.2)

Table 7 Comparing RFD mean running time with some methods from the recent literature

Dataset	HHO(MCF). Time(s)	HHO(5-LLH). Time(s)	HHO(Random). Time(s)	RFD. Time(s)
A100	0.6	0.4	0.8	0.2
ch130	0.2	0.5	0.4	0.1
d198	11.8	11.6	11.5	9.5
u1817	63.3	35.1	94.4	19.6
u2319	43.2	39.5	40.5	14.7
rl5915	204.6	217.5	209.3	180.7

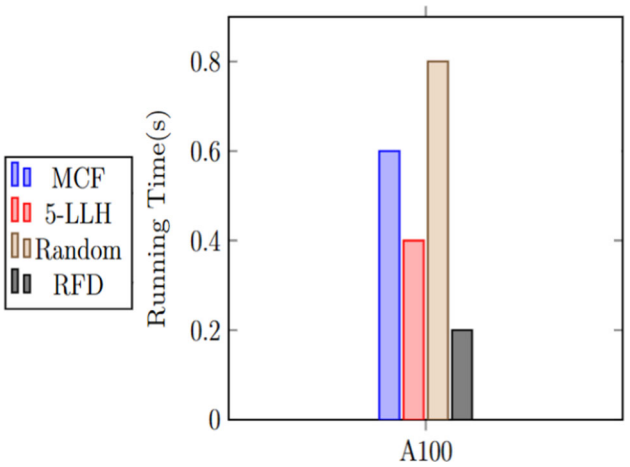

Fig. 3 Time (s) for four methods over A100

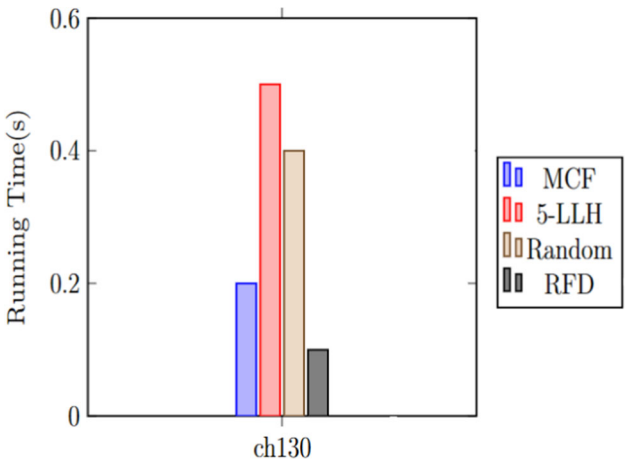

Fig. 4 Time (s) for four methods over ch130

are involved. On the other hand, RFD maintained higher efficiency scores than the others on several target problems.

A potential limitation of this study is that it did not account for the varying nature of the target problems. In future research, it might be beneficial to categorize target problems based on their characteristics and then analyze the performance of the algorithms within those categories. Moreover, exploring the interplay of different factors that might be affecting the speedup and efficiency of these algorithms, such as cache size, communication overhead, and the nature of the problems (whether they are compute-intensive or memory-bound), can be another potential direction for future work.

In conclusion, this study has shown that there is no one-size-fits-all algorithm when it comes to parallel computing. The choice of the algorithm should be guided by the specific requirements of the target problem, available computational resources, and the trade-off between speedup, efficiency, and cost.

7 Conclusion and Related Future Directions

This paper proposes a parallel version of the RFD metaheuristic algorithm to solve the TSP NP-hard problem. Parallel RFD is designed by dividing the population into multiple processors in the Apache Spark parallel framework. The proposed parallel RFD is evaluated using accuracy, running time, speedup, efficiency, and cost measures on eight

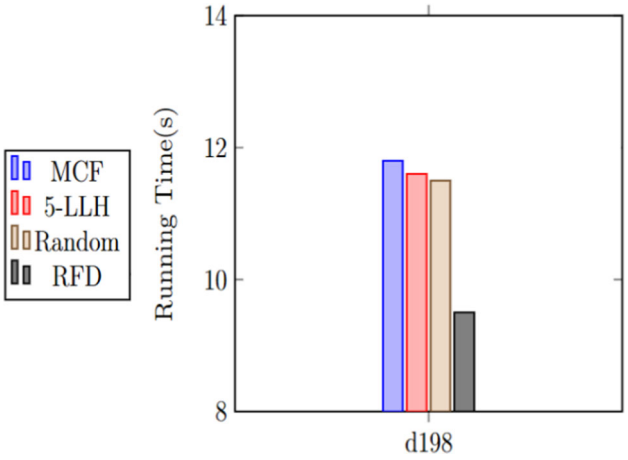

Fig. 5 Time (s) for four methods over d198

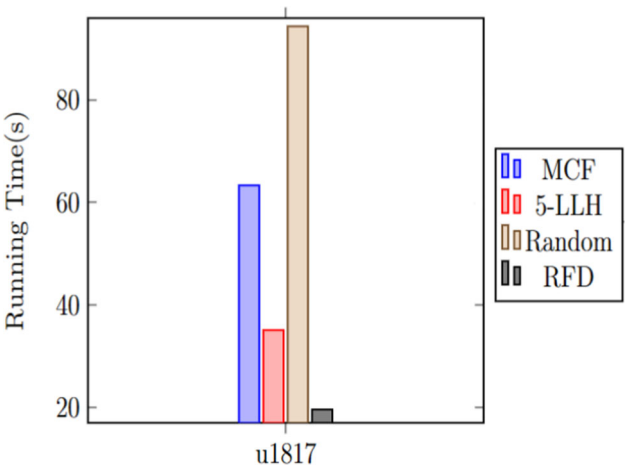

Fig. 6 Time (s) for four methods over u1817

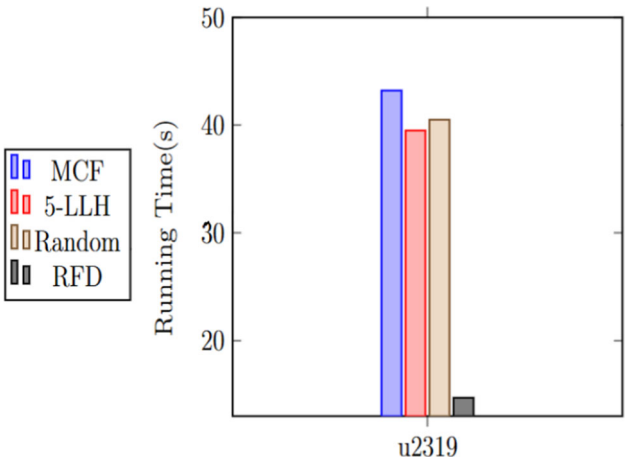

Fig. 7 Time (s) for four methods over u2319

TSP datasets. The comparisons are accomplished by comparing RFD performance on 1, 4, 8, and 16 cores. Then, it is compared with three other parallel water-based algorithms

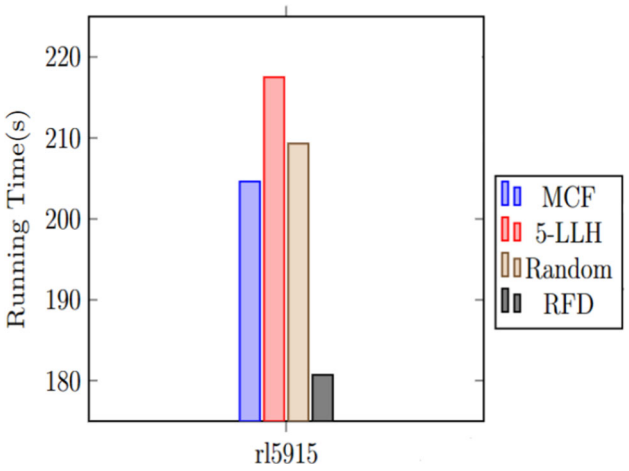

Fig. 8 Time (s) for four methods over rl5915

implemented in the same parallel environment. Then, the results of parallel RFD are compared against the reported results of other metaheuristic algorithms that were proposed to solve the TSP. The results show that in terms of running time, the RFD algorithm demonstrates the best performance for the majority of problem instances, achieving the lowest running times across different core counts.

For future works, we intend to enhance the sequential RFD using the local search operators and compare the enhanced RFD version on 1, 4, 8, 16, 32, and 64 processors. Furthermore, the proposed RFD can be applied to other applications to solve other optimization problems, such as task scheduling and load balancing in the cloud system.

Author Contributions All authors have contributed equally to this work.

Funding Open access funding provided by Linköping University.

Availability of Data and Materials Data are available on request.

Declarations

Conflict of Interest Authors have no conflict of interest.

Ethical Approval (1) This material is the authors' own original work, which has not been previously published elsewhere. (2) The paper is not currently being considered for publication elsewhere. (3) The paper reflects the authors' own research and analysis in a truthful and complete manner.

References

1. Jünger, M., Reinelt, G., Rinaldi, G.: The traveling salesman problem. Handb. Oper. Res. Manag. Sci. **7**, 225–330 (1995)
2. Cooper, J., Nicolescu, R.: The hamiltonian cycle and travelling salesman problems in cp systems. Fund. Inform. **164**(2–3), 157–180 (2019)
3. Jazayeri, A., Sayama, H.: A polynomial-time deterministic approach to the travelling salesperson problem. Int. J. Parallel Emerg. Distrib. Syst. **35**(4), 454–460 (2020)
4. Shalaby, M.A.W., Mohammed, A.R., Kassem, S.S.: Supervised fuzzy c-means techniques to solve the capacitated vehicle routing problem. Int. Arab J. Inf. Technol. **18**(3A), 452–463 (2021)
5. Mahmood, N.: Solving capacitated vehicle routing problem using meerkat clan algorithm. Int. Arab J. Inf. Technol. **19**(4), 689–694 (2022)
6. Gubin, P.Y., Kamel, S., Safaraliev, M., Senyuk, M., Hussien, A.G., Zawbaa, H.M.: Optimizing generating unit maintenance with the league championship method: a reliability-based approach. Energy Reports **10**, 135–152 (2023)
7. Izci, D., Rizk-Allah, R.M., Ekinci, S., Hussien, A.G.: Enhancing time-domain performance of vehicle cruise control system by using a multi-strategy improved run optimizer. Alexandria Eng. J. **80**, 609–622 (2023)
8. Daqaq, F., Hassan, M.H., Kamel, S., Hussien, A.G.: A leader supply-demand-based optimization for large scale optimal power flow problem considering renewable energy generations. Sci. Rep. **13**(1), 14591 (2023)
9. Hashim, F.A., Hussien, A.G.: Snake optimizer: a novel meta-heuristic optimization algorithm. Knowl. Based Syst. **242**, 108320 (2022)
10. Hu, G., Zheng, Y., Abualigah, L., Hussien, A.G.: Detdo: an adaptive hybrid dandelion optimizer for engineering optimization. Adv. Eng. Informat. **57**, 102004 (2023)
11. Hu, G., Wang, J., Li, M., Hussien, A.G., Abbas, M.: Ejs: multi-strategy enhanced jellyfish search algorithm for engineering applications. Mathematics **11**(4), 851 (2023)
12. Huangjing, Yu., Jia, H., Zhou, J., Hussien, A.: Enhanced aquila optimizer algorithm for global optimization and constrained engineering problems. Math. Biosci. Eng. **19**(12), 14173–14211 (2022)
13. Sasmal, B., Hussien, A.G., Das, A., Dhal, K.G.: A comprehensive survey on aquila optimizer. Arch. Comput. Methods Eng., 1–28 (2023)
14. Sasmal, B., Hussien, A.G., Das, A., Dhal, K.G., Saha, R.: Reptile search algorithm: theory, variants, applications, and performance evaluation. Arch. Comput. Methods Eng., 1–29 (2023)
15. Elseify, M.A., Hashim, F.A., Hussien, A.G., Kamel, S.: Single and multi-objectives based on an improved golden jackal optimization algorithm for simultaneous integration of multiple capacitors and multi-type dgs in distribution systems. Appl. Energy **353**, 122054 (2024)
16. Wang, S., Hussien, A.G., Kumar, S., AlShourbaji, I., Hashim, F.A.: A modified smell agent optimization for global optimization and industrial engineering design problems. J. Comput. Des. Eng., qwad062 (2023)
17. Mir, I., Gul, F., Mir, S., Abualigah, L., Zitar, R.A., Hussien, A.G., Awwad, E.M., Sharaf, M.: Multi-agent variational approach for robotics: a bio-inspired perspective. Biomimetics **8**(3), 294 (2023)
18. Hussien, A.G., Abualigah, L., Abu Zitar, R., Hashim, F.A., Amin, M., Saber, A., Almotairi, K.H., Gandomi, A.H.: Recent advances in harris hawks optimization: a comparative study and applications. Electronics **11**(12), 1919 (2022)
19. Chhabra, A., Hussien, A.G., Hashim, F.A.: Improved bald eagle search algorithm for global optimization and feature selection. Alex. Eng. J. **68**, 141–180 (2023)
20. Hashim, F.A., Neggaz, N., Mostafa, R.R., Abualigah, L., Damasevicius, R., Hussien, A.G.: Dimensionality reduction approach based on modified hunger games search: case study on Parkinson's disease phonation. Neural Comput. Appl. **35**, 1–27 (2023)
21. Gharehchopogh, F.S., Abdollahzadeh, B.: An efficient harris hawk optimization algorithm for solving the travelling salesman problem. Clust. Comput. **25**(3), 1981–2005 (2022)
22. Pasha, J., Nwodu, A.L., Fathollahi-Fard, A.M., Tian, G., Li, Z., Wang, H., Dulebenets, M.: Exact and metaheuristic algorithms for the vehicle routing problem with a factory-in-a-box in multi-objective settings. Adv. Eng. Informat. **52**, 101623 (2022)
23. Panwar, K., Deep, K.: Discrete grey wolf optimizer for symmetric travelling salesman problem. Appl. Soft Comput. **105**, 107298 (2021)
24. Üçoluk, G.: Genetic algorithm solution of the tsp avoiding special crossover and mutation. Intell. Automat. Soft Comput. **8**(3), 265–272 (2002)
25. Wang, K.-P., Huang, L., Zhou, C.-G., Pang, W.: Particle swarm optimization for traveling salesman problem. In: Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693), vol. 3, pp. 1583–1585. IEEE (2003)
26. Pang, W., Wang, K.-p., Zhou, C.-g., Dong, L.-j.: Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In: The fourth international conference on computer and information technology, 2004. CIT'04., pp. 796–800. IEEE (2004)
27. Shi, X.H., Liang, Y.C., Lee, H.P., Lu, C., Wang, Q.X.: Particle swarm optimization-based algorithms for tsp and generalized tsp. Inf. Process. Lett. **103**(5), 169–176 (2007)
28. Chen, S.-M., Chien, C.-Y.: Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Syst. Appl. **38**(12), 14439–14450 (2011)
29. Lin, S.: Computer solutions of the traveling salesman problem. Bell Syst. Tech. J. **44**(10), 2245–2269 (1965)
30. Karaboga, D., Gorkemli, B.: A combinatorial artificial bee colony algorithm for traveling salesman problem. In: 2011 International symposium on innovations in intelligent systems and applications, pp. 50–53. IEEE (2011)
31. Rufai, K.I., Usman, O.L., Olusanya, O.O., Adedeji, O.B.: Solving travelling salesman problem using an improved ant colony optimization algorithm. Univ. Ibadan J. Sci. Logics ICT Res. **6**(1 and 2), 158–170 (2021)
32. Wang, Y., Han, Z.: Ant colony optimization for traveling salesman problem based on parameters optimization. Appl. Soft Comput. **107**, 107439 (2021)
33. Yadav, N., Pachung, P., Agrawal, V., Bansal, J.C.: Blended selection in ant colony optimization for solving travelling salesman problem. In: 2022 IEEE World Conference on Applied Intelligence and Computing (AIC), pp. 782–787. IEEE (2022)
34. Raveendran, A.: Evaluation of a spark-enabled genetic algorithm applied to the travelling salesman problem (2020)
35. Lu, H.-C., Hwang, F.J., Huang, Y.-H.: Parallel and distributed architecture of genetic algorithm on apache hadoop and spark. Appl. Soft Comput. **95**, 106497 (2020)
36. Alanzi, E., Bennaceur, H.: Hadoop mapreduce for parallel genetic algorithm to solve traveling salesman problem. Int. J. Adv. Comput. Sci. Appl., **10**(8) (2019)
37. Adewole, P., Akinwale, A.T, Otunbanowo, K.: A genetic algorithm for solving travelling salesman problem. Int. J. Adv. Comput. Sci. Appl. (2011)

38. Asim, M., Gopalia, R., Swar, S.: Traveling salesman problem using genetic algorithm. Int. J. Latest Trends Eng. Technol. **3**(3), 183–190 (2014)
39. Jarrah, A., Bataineh, A.S., Al Almomany, A.: The optimisation of travelling salesman problem based on parallel ant colony algorithm. Int. J. Comput. Appl. Technol. **69**(4), 309–321 (2022)
40. Rhee, Y.: Gpu-based parallel ant colony system for traveling salesman problem. J. Korea Soc. Comput. Inf. **27**(2), 1–8 (2022)
41. Peng, C.: Parallel genetic algorithm for travelling salesman problem. In: International conference on automation control, algorithm, and intelligent bionics (ACAIB 2022), vol. 12253, pp. 259–267. SPIE (2022)
42. Wang, Z., Shen, Y., Li, S., Wang, S.: A fine-grained fast parallel genetic algorithm based on a ternary optical computer for solving traveling salesman problem. J. Supercomput. **79**(5), 4760–4790 (2023)
43. Mohan, A., Remya, G.: A parallel implementation of ant colony optimization for tsp based on mapreduce framework. Int. J. Comput. Appl., **88**(8) (2014)
44. Hlaing, Z.C.S.S., Khine, M.A.: Solving traveling salesman problem by using improved ant colony optimization algorithm. Int. J. Inf. Educ. Technol. **1**(5), 404 (2011)
45. Gülcü, Ş, Mahi, M., Baykan, Ö.K., Kodaz, H.: A parallel cooperative hybrid method based on ant colony optimization and 3-opt algorithm for solving traveling salesman problem. Soft Comput. **22**, 1669–1685 (2018)
46. Li, L., Cheng, Y., Tan, L., Niu, B.: A discrete artificial bee colony algorithm for tsp problem. In: Bio-inspired computing and applications: 7th international conference on intelligent computing, ICIC 2011, Zhengzhou, China, August 11–14. 2011, revised selected papers 7. Springer, pp. 566–573 (2012)
47. Er, H.R., Erdogan, N.: Parallel genetic algorithm to solve traveling salesman problem on mapreduce framework using hadoop cluster (2014). arXiv preprint arXiv:1401.6267
48. Erkartal, R.B., Çetin, Ö., Yılmaz, A.: Data collection from wireless sensor networks: Openmp application on the solution of traveling salesman problem with parallel genetic algorithm and ant colony algorithm. J. Aeronaut. Sp. Technol. **15**(2), 108–124 (2022)
49. Campus, A.K., Selçuklu, K.: Solution of travelling salesman problem using intelligent water drops algorithm
50. Halder, S., Sharma, H.K., Biswas, A., Prentkovskis, O., Majumder, S., Skačkauskas, P.: On enhanced intelligent water drops algorithm for travelling salesman problem under uncertain paradigm. Transp. Telecommun. **24**(3), 228–255 (2023)
51. Rabanal, P., Rodríguez, I., Rubio, F.: Solving dynamic tsp by using river formation dynamics. In: 2008 Fourth International Conference on Natural Computation, vol. 1, pp. 246–250. IEEE (2008)
52. Rabanal, P., Rodríguez, I., Rubio, F.: Applications of river formation dynamics. J. Comput. Sci. **22**, 26–35 (2017)
53. Afaq, H., Saini, S.: On the solutions to the travelling salesman problem using nature inspired computing techniques. Int. J. Comput. Sci. Issues (IJCSI) **8**(4), 326 (2011)
54. Alhenawi, E., Khurma, R.A., Sharieh, A.A., Al-Adwan, O., Shorman, A., Al Shannaq, F.: Parallel ant colony optimization algorithm for finding the shortest path for mountain climbing. IEEE Access **11**, 6185–6196 (2023)
55. Wang, L., Wang, Y., Xie, Y.: Implementation of a parallel algorithm based on a spark cloud computing platform. Algorithms **8**(3), 407–414 (2015)
56. Eskandar, H., Sadollah, A., Bahreininejad, A., Hamdi, M.: Water cycle algorithm—a novel metaheuristic optimization method for solving constrained engineering optimization problems. Comput. Struct. **110–111**, 151–166 (2012)
57. Hosseini, H.S.: Problem solving by intelligent water drops. In: 2007 IEEE congress on evolutionary computation, pp. 3226–3231. IEEE (2007)
58. Yang, F.-C., Wang, Y.-P.: Water flow-like algorithm for object grouping problems. J. Chin. Inst. Industr. Eng. **24**(6), 475–488 (2007)
59. Selvarani, S., Sadhasivam, G.: An intelligent water drop algorithm for optimizing task scheduling in grid environment. Int. Arab J. Inf. Technol **13**(6), 627–634 (2016)
60. Alhenawi, E., Al-Sayyed, R., Hudaib, A., Mirjalili, S.: Improved intelligent water drop-based hybrid feature selection method for microarray data processing. Comput. Biol. Chem. **103**, 107809 (2023)
61. Alhenawi, E., Alazzam, H., Al-Sayyed, R., AbuAlghanam, O., Adwan, O.: Hybrid feature selection method for intrusion detection systems based on an improved intelligent water drop algorithm. Cybernet. Inf. Technol. **22**(4), 73–90 (2022)