Noname manuscript No.
(will be inserted by the editor)

A Discrete Variant of Cuckoo Search Algorithm to Solve the Travelling Salesman Problem and Path Planning for Autonomous Trolley inside Warehouse

Mohamed Reda · Ahmed Onsy · Mostafa A. Elhosseini · Amira Y. Haikal · Mahmoud Badawy

Received: date / Accepted: date

Abstract Recently, order picking routing (OPR) for robots inside modern warehouses have become one of the most challenging problems. The process of OPR can be formulated as a Travelling Salesman Problem (TSP). Traditional techniques used to solve this problem usually require a long execution time and are problem-specific. Meta-heuristic optimisation techniques have been applied to solve this problem and have shown outstanding results. In this study, we solve the OPR problem using a newly proposed discrete variant of the cuckoo search algorithm. Five modifications were made to the current discrete cuckoo search algorithm. The proposed variant was applied to a traditional TSP problem. Then, the proposed algorithm was customised to solve the OPR problem in a warehouse environment. Finally, the proposed algorithm was applied to a physical prototype. It was then compared with genetic, particle swarm optimisation, and ant colony optimisation algorithms. Simulation and practical results proved the significant performance of the proposed algorithm over all other algorithms, especially in solving complex problems.

Keywords Discrete Cuckoo Search Algorithm · Meta-heuristic Optimization · Order Picking Routing Problem · Path planning · Travelling Salesman Problem

1 Introduction

Warehouses are responsible for storing goods and picking orders for customers. Once the customer confirms his/her order, the human pickers prepare the order by picking the goods from their storage location to the shipment location [64]. To fulfil the needs of large number of customers, the delivery process must be accurate and rapid [18].

There are two main types of warehouses: traditional warehouses and modern autonomous warehouses. A traditional warehouse is based on a long conveyor belt that moves through the entire warehouse. The ordered items are

Address(es) of author(s) should be given

picked by humans and placed on the belt. The picking process for serving millions of customers takes a huge amount of time, money, and labour. However, the design of this system has several drawbacks [45]. The system is static and difficult to modify after installation. In addition, the system lacks scalability, because it is difficult to extend. Furthermore, the items always have the same storage location, so re-slotting the newly arrived items is quite difficult.

The second type of warehouse is the modern autonomous warehouse. The growing number of customers has made the need for autonomous warehouses crucial. In such warehouses, robots, which are autonomous trolleys, are responsible for picking items from their storage locations and bringing them to the human operators to pick the order. In this system, the human picker remains at the same location to pick multiple orders. This system reduces the cost to human operators and increases the speed of the order picking process. The main principle of an autonomous warehouse is based on artificial intelligence [44]. The main advantages of the autonomous warehouse are as follows: (i) the same item can be placed in different locations, so there are no permanent locations anymore, (ii) the system automatically finds the best shelf based on the available space, and (iii) the robots find the best path to the items automatically.

The most challenging problem in modern warehouses is the order picking routing (OPR) problem. This is the most time-consuming process inside a warehouse system because there are a large number of permutations for picking orders [59]. The routing policy for the order picking process can be formulated as the well-known travelling salesman problem (TSP). The main objective is to pick all items in a customer order with minimum travel distance [15], [61].

In the TSP, the salesman must visit several cities during the trip. Each city should be visited once during the entire trip and then returned to the starting point. The main objective is to find a route that has the minimum travelling distance. This is an NP-hard problem, which means that solving it is difficult because the number of possible solutions increases exponentially with problem dimensions. The order picking problem is similar to a TSP, where the salesman is the robot (trolley) and the cities are the items that should be picked to fulfil the customer's order.

Recently, several techniques have been presented to solve path planning and OPR. There are three main methodologies for solving these problems: exact routing algorithms, heuristic algorithms, and meta-heuristic algorithms [29]. The first type is the exact algorithms that always find an optimal solution, such as the RR (Ratliff and Rosenthal) Algorithm [14]. The drawback of exact algorithms is that they usually take a very long time to reach an optimal solution, as most of them are based on exhaustive search. Therefore, they are not feasible for real-time applications.

The second type is the heuristic algorithms, which are problem-specific. However, in most cases, the results are not optimal [55]. The drawback of these algorithms is that they are problem-dependant. Therefore, the heuristic approach for one problem cannot be used for another problem.

The third type is the meta-heuristic algorithms, which are high-level problem-independent algorithms that have proven to find approximate solutions to the problem. Examples of these algorithms include genetic algorithms (GA) [60], particle swarm optimisation (PSO) [35], and ant colony optimisation (ACO) [7]. Meta-heuristic algorithms have proven their efficiency in finding near-optimal solutions, especially for problems with a large number of dimensions.

The most commonly used meta-heuristic optimisation algorithm to solve the OPR problem as an application of the TSP problem is the genetic algorithm [38]. The genetic algorithm appears to find good solutions for the travelling salesman problem; however, it depends significantly on the encoding of the problem and the type of mutation, crossover, and selection methods used. It has multiple parameters to be tuned, such as crossover probability and mutation probability. These factors make parameter tuning difficult, and multiple combinations should be tested [17].

The present study considers solving the OPR problem using a cuckoo search-based meta-heuristic approach. The main reason for choosing this algorithm is that it has only one parameter to be tuned, which is called the probability of discovery. Additionally, it can balance the global search (exploration) and local search (exploitation), which are the most important factors that make a meta-heuristic algorithm reach an optimal solution faster and more accurately [66], [67]. Despite the prominent results of the cuckoo search algorithm (CSA) in continuous and discrete optimisation problems, it has not yet been used to solve the OPR problem inside warehouses. The main contributions of this study are summarised as follows:

1. A new discrete variant of CSA is proposed, namely, discrete damped cuckoo search (DDCS).
2. Modification and improvement of CSA:
   (a) Applying different steps of the 2-pt moves on the not-so-bad solutions.
   (b) Using random key encoding for the solution to preserve the good continuous characteristics of the original CSA algorithm.
   (c) Adapting the Lévy flight Random Walk on the permutation solutions rather than the continuous solutions.
   (d) Using adaptive parameters instead of fixed parameters for the CSA algorithm.
   (e) Applying the concept of population division on the discovered solutions, and applying different types of mutation operators on the sub-groups.
3. The proposed DDCS algorithm is designed to solve the OPR problem. To the best of our knowledge, CSA has not been applied to the OPR problem in a warehouse environment yet.
4. The superiority of DDCS:
   (a) The proposed DDCS algorithm outperforms GA, PSO, and ACO algorithms on the well-known TSP problems benchmark and warehouse OPR problems.

(b) Convergence speed of the DDCS algorithm was tested, and it was found to have a faster convergence than all the other algorithms. Therefore, it can be applied to real-time problems.
(c) The results of the proposed DDCS algorithm show the best distribution on box plots and violin plots. These results prove the repeatability and reliability of the proposed algorithm.

5. A physical prototype robot was designed, and the proposed algorithm managed to find the optimal path practically.

The remainder of this paper is organised as follows: Section 2 covers the state-of-the-art methods used for solving the TSP problem. Section 3 explains the problem statement and mathematical formulation of the OPR problem. Section4 discusses the methodology and proposed solution. Section 5 explores the model assumptions and algorithm parameters. Section 6 discusses the simulation results and statistical analysis of the proposed DDCS algorithm. In Section 8, the paper is concluded, and future work is explored. For the convenience of readers, the abbreviations used are listed in Table 1.

Table 1: List of Abbreviations

Symbol	Acronym / Abbreviation
CSA	Cuckoo Search Algorithm
TSP	Travelling Salesman Problem
OPR	Order Picking Routing
DDCS	Discrete Damped Cuckoo Search
ACO	Ant Colony Optimisation
GA	Genetic Algorithm
TS	Tabu Search
SA	Simulated Annealing
PSO	Particle Swarm Optimization
RR	Ratliff and Rosenthal
LKH	Lin–Kernighan–Helsgaun
PSO-NIC-SA	particle swarm optimization with simulated annealing and neighborhood information
GSAACS-PSOT	genetic simulated annealing ant colony system with particle swarm optimisation techniques
LFRW	Lévy Flight Random Walk
BSRW	biased/selective random walk.
KDE	kernel density estimation
GUI	Graphical user interface

2 Literature Review

Recently, extensive research has been conducted to solve NP-hard problems. TSP is one of the most well-known NP-hard problems. In this problem, it is necessary for the salesman to visit several cities once during the whole trip and then return to the starting point. The main objective is to determine a route that has the minimum travelling distance [2]. This problem has various

applications, such as determining the path for a driller to drill a printed circuit board, X-ray crystallography, vehicle routing, scheduling, and OPR inside warehouses [34], [49].

The main focus of our study is on the OPR problem inside warehouses as a TSP problem. Before exploring the algorithms that have been reported in the literature, the characteristics and complexity of the designed warehouse should be determined. In this study, we consider a conventional warehouse with multiple blocks. Conventional warehouses have a rectangular shape with parallel picking aisles perpendicular to a certain number of straight cross aisles. Warehouses with more than two cross aisles are often referred to as multi-block warehouses [38]. The number of blocks in this study will be up to 2500 blocks (huge warehouse).

The dimensions of the TSP depend on the number of items in the customer order. The worst-case scenario occurs when the order size equals the number of blocks, where each block holds only one item. It is necessary to obtain the final path within a few minutes to make the robot follow the resulting path and pick the order for the customer as quickly as possible. The final path is not necessarily the shortest (a good solution is not necessarily the best solution). Several techniques used to solve this problem are discussed in detail in the following sections.

2.1 Exact Algorithms

De Koster and van der Poort (1998) used a general form of the RR algorithm for decentralised deposition in traditional warehouses. It can deal with a situation where the order picker can deposit the retrieved items at the respective front ends of each picking aisle without returning to the depot [14].

Roodbergen and de Koster (2001) studied a conventional warehouse with a middle cross-aisle that divided the warehouse into upper and lower blocks. The authors applied the concept of RR to iteratively construct a minimum-length tour subgraph by expanding the subgraphs based on the specific steps proposed in [50]

Matusiak et al. (2014) solved the OPR and order batching problem in a multi-block conventional warehouse. They used the (exact) A* algorithm, which is based on dynamic programming. The authors assumed that multiple depots were located at the back cross-aisle [39].

Çelik and Süral (2016) proposed another extension of the RR algorithm by considering turn penalties in addition to the regular travel time of the order picker. They solved different types of problems (single-objective turn minimisation, bi-objective travel time minimisation, and tri-objective problems with U-turn minimisation) in polynomial time [5].

2.2 Heuristics Algorithms

Goetschalckx and Ratliff (1988) introduced a simple heuristic approach for wide-aisle warehouses, called the Z-pick heuristic. The heuristic approach determines a route where the robot, which navigates the warehouse, travels in a zigzag shape through a wide aisle to collect requested items from both sides of the aisle[26].

Hall (1993) proposed three simple heuristics for the OPR problem. A comparative study was conducted among the three methods. The OPR problem is assumed to exist in a single-block warehouse with narrow aisles and a single depot. The three methods are referred to as the traversal, midpoint, and largest gap heuristics [28].

Theys et al. (2010) applied the Lin–Kernighan–Helsgaun (LKH) TSP heuristic to solve the OPR problem in a conventional warehouse with two blocks. These heuristics, which were used in their algorithm, improved the initial solutions generated without increasing the run time.[57].

Chabot et al. (2017) [6] modified the heuristics proposed by Hall (1993) [28]. This modified heuristic is the same as the original procedure, with the additional condition that a requested item is retrieved only if it respects all constraints of the problem. Otherwise, it is skipped and picked on the next tour.

Chen et al. (2019) proposed heuristics for warehouses with ultra-narrow aisles. In this configuration, the picker cannot enter sub-aisles using a picking device. However, the device must remain at the aisle entrance. Moreover, the picker can enter and leave from only one side of the aisle because the other side is blocked by a warehouse wall [8].

2.3 Metaheuristic Optimization Techniques

As previously discussed, exact algorithms require a long execution time to find an optimal solution, whereas the heuristic algorithms are problem-specific. Therefore, the need to find good (not necessarily the best) solutions to this problem has led to an increase in the use of metaheuristic optimisation algorithms. Metaheuristic optimisation algorithms have advantages over the exact traditional algorithms. These algorithms are easy to implement and can solve the most complex optimisation problems [23],[24]. Additionally, metaheuristic optimisation algorithms are very flexible; they can deal with discrete, continuous, or even mixed optimisation problems with any number of dimensions [25],[68].

Various metaheuristic algorithms have attempted to solve the TSP, such as ACO [16], GA [33], TS [36], SA [4], and discrete PSO (DPSO) [53]. Moreover, some hybrid algorithms are also used to solve this problem, such as fuzzy particle swarm optimization with simulated annealing and neighborhood information (PSO-NIC-SA) [1] and genetic simulated annealing ant colony system with particle swarm optimisation techniques (GSAACS- PSOT) [10]. The fol-

lowing sections discuss in detail some of these metaheuristic algorithms for solving the TSP.

2.3.1 Genetic Algorithm

Grefenstette et al. proposed a version of the GA to solve the TSP[27]. The algorithm begins with a random population consisting of candidate solutions (tours). Next, a percentage of this population is selected for the crossover operation to produce new offspring (children). Then, a small percentage of the new generation is mutated to improve individuals in the new population.

Crossover and mutation operators are the most important operators in GA. The former includes various methods, such as cyclic crossover, order crossover, and partially-mapped crossover. The latter uses different methods, such as swap mutation and scramble mutation. The selection of the crossover and mutation methods depends greatly on the application and encoding type [42].

Mutation is one of the most effective operations in GA. This can improve the exploration of search space. Fogel [22] claimed that everything can be performed using the mutation operator [21], which is the most important operator in optimisation [19]. There are three types of mutation operators: inversion operator, displacement operator, and pairwise swap operator [20]. These operators have been explained in detail in [11].

2.3.2 Particle Swarm Optimization (PSO)

PSO generates an initial population consisting of particles that are considered candidate solutions. Each particle in the population moves with a variable velocity to reach the global best solution obtained thus far and the best solution of the previous generation. PSO has proved its efficiency in many applications and complex optimisation problems. However, it could not prove its efficiency in the TSP because of the difficulty in introducing velocity in the combinatorial space [42]. Shi et al. introduced the permutation concept into PSO and proposed a DPSO algorithm. They focused on interpreting the velocity via a mutation operator that is inspired by the swap operator developed by Wang et al. [62].

2.3.3 Ant Colony Optimization (ACO)

ACO emulates the behaviour of ants when searching for food. An ant starts by visiting a random place and then moves from one place to another, leaving a mark in each city visited. Other ants passing these cities can follow the marks of the previous ants or choose another path randomly. A basic version of the ACO algorithm was designed to solve the TSP [16].

A hybrid algorithm, named GSAACS- PSOT, was developed by Chen and Chien to solve the TSP. This algorithm combines the ACO, GA, PSO, and SA algorithms. This algorithm generates an initial population using the ACO algorithm. Next, the population is improved by applying the GA and SA

mutation operators. After a specific number of cycles, PSO is used to exchange information between subgroups of the population [10].

2.4 Cuckoo Search Algorithm (CSA)

The (CSA is a meta-heuristic optimisation algorithm that is also used to solve the TSP. This separate section discusses it in detail because the proposed algorithm is based on it. The CSA algorithm was developed by Yang and Deb in 2009 to imitate the brood parasitism behaviour of cuckoos. This bird lays its eggs in another nest, called the host nest. Cuckoo birds imitate the pattern and colour of the eggs owned by the host to prevent them from being discovered by the host. If the host recognises the cuckoo eggs, it can throw them out of its nest or leave its nest and build a new one.

The CSA has three main components [67], [63]. The first one is Exploitation using Lévy Flight Random Walk (LFRW), which is used to generate new solutions near to the best-obtained solution so far. The LFRW is expressed in Eq. (1), where $X_{i,G}$ is the current solution, $X_{i,G+1}$ is the newly generated solution, and X_{best} is the best obtained solution. Furthermore, α_0 is the scaling factor that is equal to 0.01 [69], β is a constant that is equal to 1.5 [66], and μ and ν are randomly generated numbers. The parameter ϕ is calculated as in (2), where Γ is the gamma function.

$$X_{i,G+1} = X_{i,G} + \alpha_0 \frac{\phi \times \mu}{|\upsilon|^{\frac{1}{\beta}}}(X_{i,G} - X_{best}), \tag{1}$$

$$\phi = \left(\frac{\Gamma(1+\beta) \times sin(\frac{\pi \times \beta}{2})}{\Gamma\left(\frac{1+\beta}{2} \times \beta \times 2^{\frac{\beta-1}{2}}\right)} \right)^{\frac{1}{\beta}}. \tag{2}$$

The second component is exploration using biased/selective random walk (BSRW), which is used to generate new solutions in locations far from the best-obtained solution, thereby preventing the search from being trapped in a local optimum. Moreover, it provides an acceptable diversity because it explores various areas of the entire search space. The third component is represented by the elitism scheme. In this operation, the CSA selects the best solutions that have the best fitness values and passes them directly to the next generation. This prevents the best solutions from being lost. A good compromise between the above three components can ensure an efficient algorithm. A balance among these components exists in the CSA; therefore, it is considered an effective optimisation algorithm.

2.4.1 Adaptive Variants of CSA

One of the most important parameters in the CSA is Lévy Flight step size. Various modifications were made to this parameter to make it adaptive to the number of generations. Making it adaptive improves the performance and

convergence rate of this algorithm [47]. In 2019, Reda et al. [46] proposed a damped CSA. This algorithm was based on making the step size of Lévy Flight adaptive using the concept of damped oscillations as shown in Eq. (3), where α_{min} and α_{max} denote the lower and upper bounds of α_0, respectively. Furthermore, τ and ω are constants, g is the current generation, and g_{max} is the maximum number of generations. This variant outperformed all the adaptive variants of CSA. Therefore, this method is used in the proposed DDCS algorithm to make the step size of Lévy Flight adaptive, but this time for a discrete version of the CSA.

$$\alpha_0 = \alpha_{min} + (\alpha_{max} - \alpha_{min}) e^{-\left(\frac{\tau\ g}{gmax}\right)} \left[cos\left(\frac{\omega\ g}{gmax}\right) + \alpha_{min} \sin\left(\frac{\omega\ g}{gmax}\right) \right] . \tag{3}$$

2.4.2 Discrete Variants of CSA

Ouaarab et al. [41],[42] proposed a new discrete version of the CSA. The main objective of this discrete version is to solve the TSP. This algorithm proved to be effective on a set of benchmark TSPs.

The main idea of this algorithm is to represent the small steps of Lévy Flight using 2-opt moves [12]. The 2-opt moves are shown in Fig.1. Edges (a,b) and (c,d) are replaced by edges (a,c) and (b,d). This move is used for small step sizes in the Lévy Flight random walk. Large Lévy Flight step sizes are represented by the Double-bridge move [37].

This algorithm proved its efficiency in solving the TSP. However, it did not consider the discovery probability P_a, which is an important parameter. It also replaced the entire LFRW equation 1 with 2-opt and double bridge moves. This is acceptable for converting an algorithm that solves continuous optimisation problems into algorithms that solve permutation problems, such as TSP. Therefore, our motivation in this study is to maintain the main characteristics of the original CSA and adapt it to solve the TSP.

Ouaarab et al. proposed another discrete variant of the CSA in 2015[43]. This variant is based on a simplified random-key encoding scheme that converts the population from a continuous representation (real numbers) into a combinatorial space. Displacement of a solution in both spaces use Lévy flights. Random key encoding was used to solve the TSP in [9] and [54]. In this variant, a random key encoding scheme was used with CSA. The random key encoding is explained in detail in [3].

This random key cuckoo search variant overcomes the problems faced by the DCSA proposed in 2014 [41]. It can preserve some of the continuous characteristics of the standard CSA. Furthermore, it uses the Lévy Flight to cause perturbation to the real indices, then reorders the solution. However, it used static values for Lévy Flight step size. In the proposed algorithm, we use this scheme to represent the TSP solution. Therefore, the main continuous characteristics of the standard CSA can be preserved. In addition, the step size

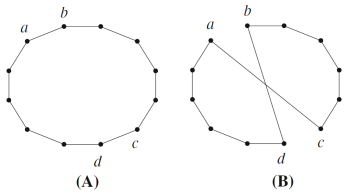


Fig. 1: The 2-opt move [41]

Table 2: The Advantages and disadvantages of the aforementioned algorithms

Category	Advantages	Disadvantages
Exact algorithms	They can always find the optimal solution	They take too long execution time. They are not suitable for real-time applications.
Heuristic algorithms	They are faster than the exact algorithms	They are problem-specific
Meta-heuristic optimization algorithms	They are problem independent They are easy to implement They have fast execution time	These algorithms can find near-optimal solutions, not necessarily the optimal one.

of Lévy Flight is adaptive to the generation number. This adaptive scheme provides a balance between exploration and exploitation[46].

Table 2 summarises the advantages and disadvantages of the three main categories of algorithms. The first is the exact algorithms. They can always find the optimal solution. However, they are based on exhaustive search; therefore, they are not suitable for real-time applications because they are too time-consuming. The second type is heuristic algorithms, which are faster than exact algorithms but are problem-specific. The third type comprises meta-heuristic optimisation algorithms, such as GA, PSO, ACO, and CSA. These algorithms can find near-optimal solutions, but not necessarily the optimal one. They are problem-independent, easy to implement, and have fast execution times. The most promising meta-heuristic algorithm is the CSA. The advantage of this approach over the others is that it has only one tuning parameter. In addition, it has a good balance between exploration and exploitation and has not been applied before in a warehouse OPR problem. Some CSA variants attempted to solve the TSP, but their main disadvantage was that they could not preserve the main characteristics that distinguished the CSA.

3 Problem Statement: Order Picking Routing (OPR) Problem vs TSP

As previously stated, the most time-consuming process inside a warehouse system is the OPR problem. The OPR problem can be formulated as the traditional TSP. In TSP, the salesman wants to visit multiple cities with the shortest distance. Similarly, in the OPR problem, the robot wants to pick items from the shelves with shortest possible distance. Therefore, the salesman is mapped to the robot, and the cities are mapped to the items to be picked, as shown in Fig. 2.

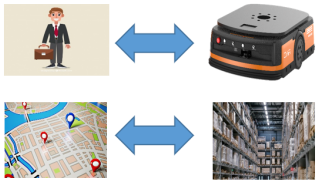


Fig. 2: Mapping Concept between TSP and OPR problem.

The difficulty in this problem is the large number of possible permutations required to pick the orders [59]. This is an NP-hard problem, where the number of possible solutions increases exponentially with the problem dimensions. In this section, the relationship between the OPR problem and TSP is discussed in detail. First, the warehouse configuration is explained. Second, some assumptions are made to reduce the dimensions of the OPR problem to improve the performance. Third, a mathematical representation of the fitness function for the OPR problem is presented.

3.1 Warehouse Configurations and Assumptions

The model of the warehouse that we plan to apply to the proposed DDCS algorithm is shown in Fig. 3. This warehouse consists of four blocks: A, B, C, and D. Each block consists of eight sections; for example, block A has eight sections from A1 to A8. Each section can store a specific type of product/item. The aisles consist of vertical and horizontal lines that intersect at specific points. The intersection points are shown in Fig. 3 as red circles and each are named P1, P2, ..., P9. Each aisle contains a set of points called item points. Each item on the shelf of a specific section can be obtained from the item points. For example, the items on section A1 can be reached when the robot is located at point PA1. Similarly, the items on section B4 can be obtained when the robot is located at point PB4.

3.2 The concept of using Intersection points of the aisles instead of Item points

The total number of item positions in the warehouse shown in Fig. 3 is 32. Therefore, the worst-case scenario is when there is an order containing 32 different items. This is a complex TSP. To simplify this, we added the concept of dealing with corner points (intersection points) instead of item points. This reduces the worst case of TSP from 32 locations to nine locations.

For example, to reach the item A1 at point PA1, the robot must first go to intersection point P1, as shown in Fig.4. Therefore, the main focus should be on the intersection points of aisles and not on the item points themselves. This

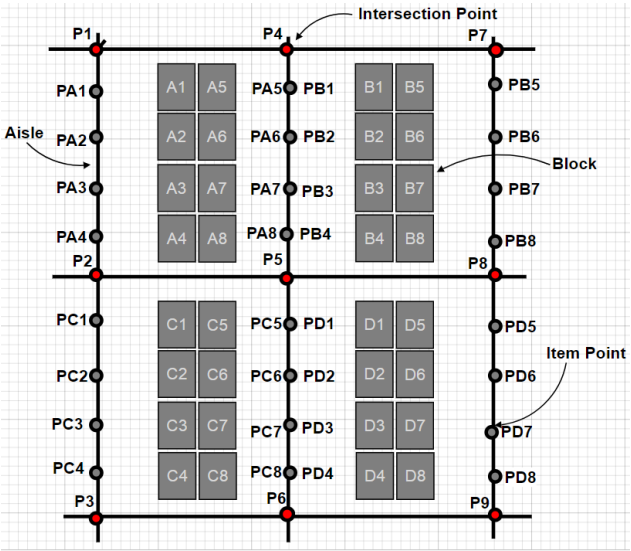


Fig. 3: Warehouse configuration map (4 blocks).

enables us to determine the exact shortest path. Many items can be obtained from the same intersection point, as shown in Fig.4.

Algorithm 1 shows the use of intersection points instead of item points. In the first step, Lines 1 and 2 are used to obtain the coordinates of the required items. Next, in Line 3, we obtain the intersection points that correspond to the item points; there may be repeated intersection points because one intersection point can have multiple item points. Therefore, the repeated points in C are deleted in Line 4. As a traditional TSP, we find the best permutation of the intersection points to obtain the shortest path between them. From Lines 9 to 14, the item points must be added after the corresponding intersection points to construct the final path.

However, this method also has several limitations. We assumed that an item can be reached from only one end, whereas it can be reached from two intersection points, but we choose the intersection point nearest to it. For example, point PB5 can be reached from intersection points P7 or P8, as shown in Fig.3. However, we chose P7 because it was nearest to it. Thus, this appears to be the best method. Nevertheless, this can cause a slightly longer path, as in the case explained in Fig. 5. In this case, it is necessary to move from point PA5 to item point PD7. The shortest path is indicated by blue arrows. However, we assumed that point PD7 could be reached through intersection point P9. Therefore, the algorithm provides a green path, which is slightly longer. This disadvantage seems to be unsuitable for small-scale problems; however, it has a significant effect on the performance and computation time in large-scale problems because the number of possible solutions exponentially decreases.

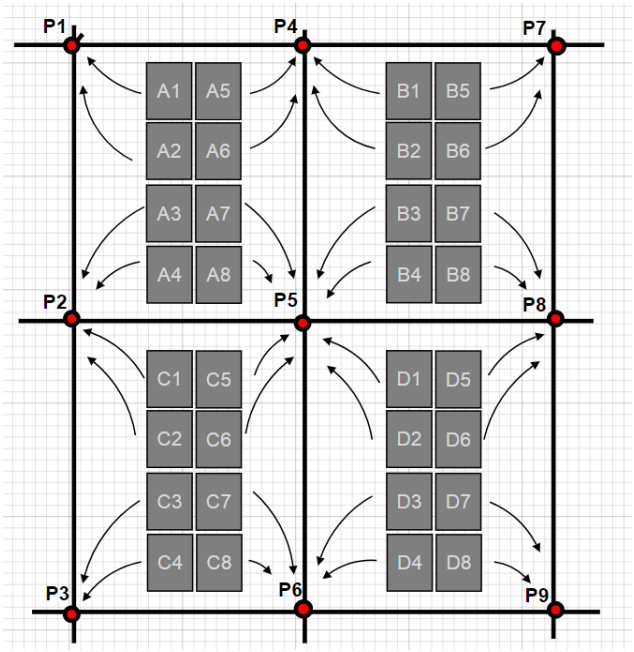


Fig. 4: Reducing TSP points via dealing with intersection points.

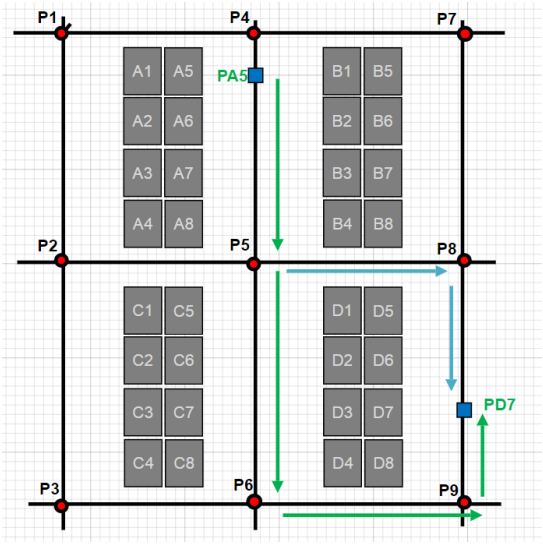


Fig. 5: Algorithm limitations that could cause a longer path.

Algorithm 1: Warehouse path construction using the concept of intersection points

```
Initialise matrix M that contains the item points and their corresponding
  intersection points, as shown in Fig.4;
Initialise vector O that contains a set of items (item points);
Construct vector C that contains the intersection points corresponding to the item
  points from M;
Remove duplications from vector C.
Find the best permutation in vector C as a traditional TSP;
The best solution found in the vector P is stored;
Vector P contains the best order for the intersection points;
Initialise S that stores the final path;
for each point p in P do
    Add point p to the vector S;
    Find all item points in O that are related to point p, and add these points in
      vector K;
    Note: All points in the vector K have the same x-coordinate; Sort the vector K
      based on the Y-coordinate ;
    Add the sorted K to vector S
end
Visualise the final solution stored in S;
```

3.3 Vertical and Horizontal displacement constraints inside warehouse

As seen in Fig. 3, the aisles inside the warehouse are a set of horizontal lines and vertical lines that intersect at a set of points. Therefore, the robot is not allowed to move along a diagonal path between two points. This constraint should be considered when implementing the movement of the robot inside the warehouse. Therefore, we have to develop a movement algorithm that allows the robot to move from one point to another in vertical and horizontal paths.

As explained previously, the results from Algorithm 1 represent only the order of the points that should be visited. To move from one point to another, we must find the vertical and horizontal paths to move from one point to another. Fig. 6 shows the effect of the diagonal movement constraint in the OPR problem compared to the traditional TSP. The black TSP path moves diagonally from points P1 to P5, leading to a collision with the block. On the other hand, the OPR path, the red one, moves only in horizontal and vertical directions to avoid collisions.

For every two successive points in the path obtained from Algorithm 1, the displacement difference is calculated on the X- and Y-axes by subtracting the coordinates of the starting point from the endpoint. This calculation is expressed in Eq. (4), where (X_s, Y_s) are the coordinates of the starting point/current position, and (X_e, Y_e) are the coordinates of the target point (next point).

There are eight possible solutions for this problem. These cases are illustrated in Fig. 7. In the first four cases, the link between the starting point and the next point is diagonal. Therefore, an intermediate point (X_c, Y_c), which is marked in red in Fig. 7, should be calculated to make the displacement from

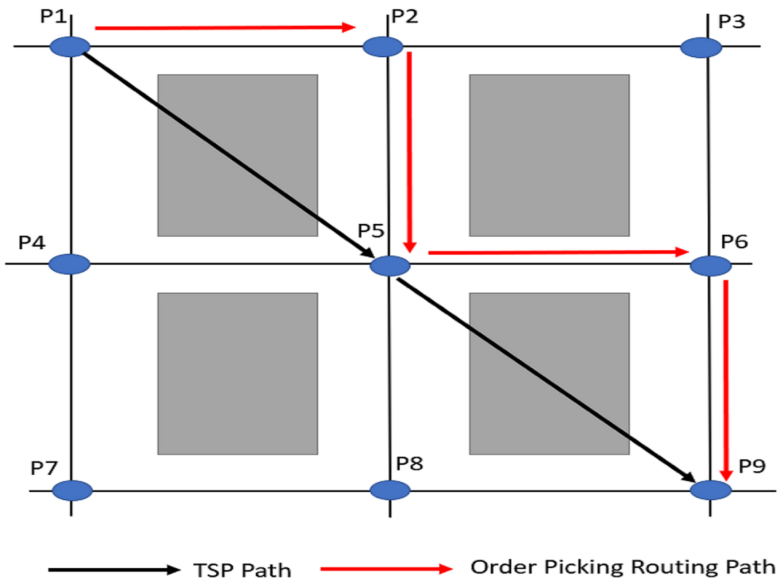


Fig. 6: Difference between TSP and OPR paths

the starting point and the next point in vertical and horizontal displacements. As seen in Fig.7, the intermediate point has the same X-coordinate as that of the starting point, while the Y-coordinate of the intermediate point can be obtained by adding the displacement dy to the Y-coordinate of the starting point. Hence, the intermediate point in the first four cases can be evaluated using Eq. (5). In the last four cases, there will be no intermediate points because the starting point and endpoint both lie on the same vertical axis as in cases 5 and 6, or on the same horizontal line as in cases 7 and 8.

$$\begin{aligned} dx &= X_e - X_s \\ dy &= Y_e - Y_s \end{aligned} \tag{4}$$

$$\begin{aligned} X_c &= X_s \\ Y_c &= Y_s + dy \end{aligned} \tag{5}$$

3.4 Cost Function for the OPR problem vs TSP

In general, the cost of a path consisting of n cities for the TSP is calculated by measuring the distance $dist_{i,i+1}$ between every two successive points $i, i+1$ in the tour, as in Eq. (6).

As previously explained, the robot inside the warehouse cannot move along a diagonal path between two points. Therefore, movement inside the warehouse should be vertical and horizontal, as shown in Fig.7. Hence, the actual distance

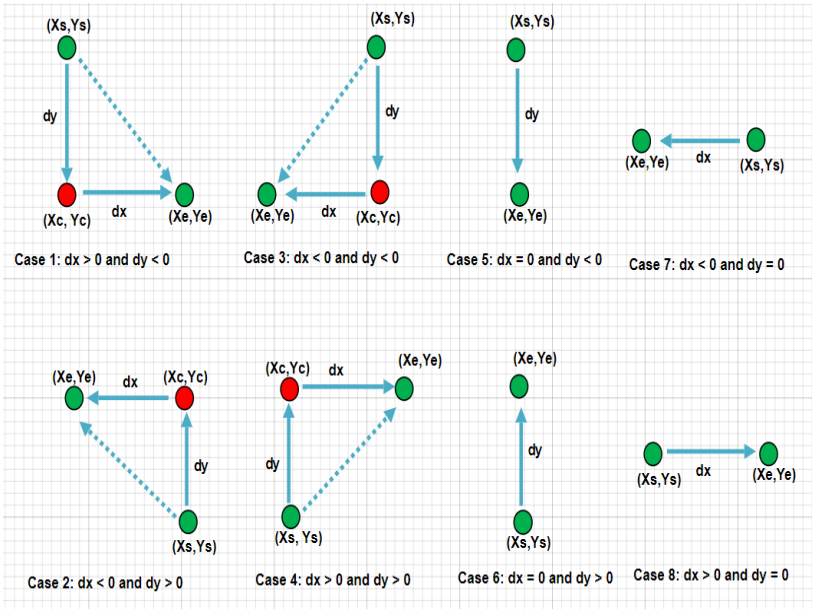


Fig. 7: The eight cases for avoiding diagonal displacement.

between any two points can be obtained by adding the absolute values of the vertical and horizontal components,dy and dx, as in Eq. (7). These equations are shown in Fig. 8 to distinguish between the cost functions in TSP and OPR problem. The total cost function for the entire path is calculated using Eq.(8), where n is the number of points on the tour.

$$dist_{i,i+1} = \sqrt{dx_{i,i+1}^2 + dy_{i,i+1}^2} \tag{6}$$

$$dist_{i,i+1} = \mid dx_{i,i+1} \mid + \mid dy_{i,i+1} \mid \tag{7}$$

$$Cost = \sum_{i=1}^{n-1} dist_{i,i+1} \tag{8}$$

This section can be summarised as follows:

1. Determine the location of the points for required items in the warehouse.
2. Replace the item point with its nearest intersection point.
3. Find the best permutation of the points as a traditional TSP.
4. Evaluate the corner points between the two successive intersection points to avoid diagonal movement. The only ways to move inside the warehouse are vertically and horizontally along the aisles.

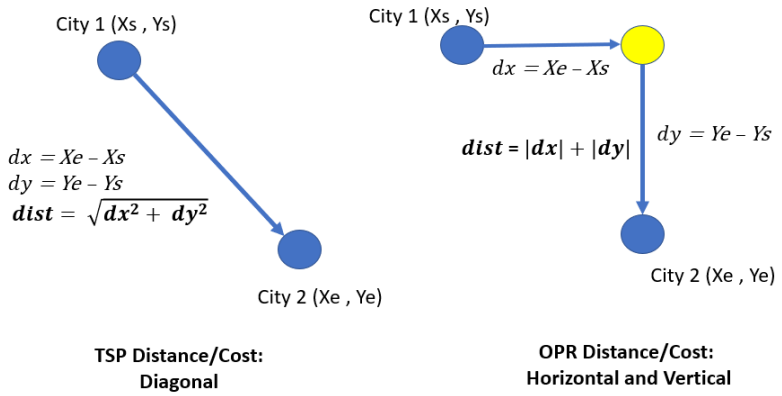


Fig. 8: Cost (Fitness Function) for the TSP and OPR problem.

The decisive step in determining the best path inside the warehouse is step number 3. In this step, we aim to find the best permutation of the intersection points. This description can be represented as a traditional TSP. Furthermore, the next section discusses the proposed algorithm for determining the shortest tour for the designated OPR problem.

4 Proposed Algorithm: Discrete damped cuckoo search algorithm DDCS

The main contribution of this study is to solve the OPR problem using the new proposed variant of the CSA, called the discrete damped cuckoo search (DDCS) algorithm. The proposed algorithm can be used to solve traditional TSPs. Additionally, it can solve the OPR problem, which is formulated as TSP in Section 3.

In the proposed DDCS algorithm, several modifications have been made to adapt the algorithm to the TSP and warehouse OPR problems. These modifications can be summarised as follows:

- Using the 2-opt moves for the small Lévy flight step size.
- Representing the TSP solutions via real random key indices as a real-valued representation.
- Adapting the damped Lévy flight random walk for discrete problems and using it for large steps.
- Applying the concept of dividing the population into sub-groups to increase the diversity of solutions.
- Applying different mutation operators for the sub-groups in the case of the solutions being discovered by a probability P_a.

4.1 Modification 1: The 2-opt Move

At the beginning of every iteration of the proposed DDCS algorithm, the population is sorted based on the fitness value of each candidate solution. The best solution is at the top of the population, whereas the worst solution is at the bottom.

The proposed DDCS algorithm depends on the 2-opt moves in the Lévy flight step size for the top 50% of the population. This is because the 2-opt move has a small step size (Fig.1). Therefore, it is used to slightly perturb the top best solutions without significantly affecting the quality of the solution.

On moving from top to bottom for the first half of the population, more 2-opt-move steps are applied to the solution. The number of 2-opt-move steps applied is according to Table 3.

Table 3: The 2-opt Move steps for the first half of the population.

Percentage of Population %	No. of 2-opt-move steps
Top 5 %	Elitism (Keep the solutions with no change
From 5% to 15%	Apply one step by 2-opt move
From 15% to 25%	Apply two steps by 2-opt move
From 25% to 35%	Apply three steps by 2-opt move
From 35% to 40%	Apply four steps by 2-opt move
From 40% to 45%	Apply five steps by 2-opt move
From 45% to 50%	Apply six steps by 2-opt move

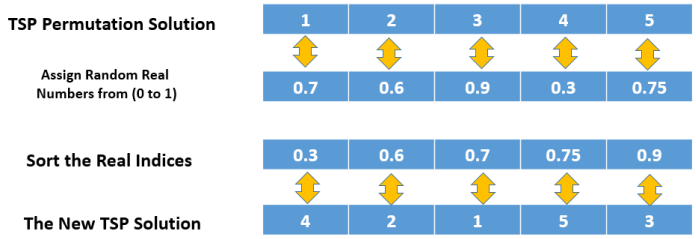


Fig. 9: Random Key Encoding Process

The other half of the population had lower fitness than the first half. Therefore, the 2-opt moves will not change the solution significantly, and it is required to perform a larger move. These large moves are represented in the proposed DDCS algorithm using the damped Lévy flight step size via random keys.

4.2 Modifications 2 to 4: Adaptive Lévy flight random walk with random key

As previously explained, the first half of the population is updated via the 2-opt move steps. However, the second half of the population contains worse solutions, which require large step modifications. Therefore, the Lévy flight random walk is a suitable choice because of the tremendous size of the step, which enables the algorithm to explore more search spaces and enhance exploration.

The problem with Lévy flight random walk is that it is applied on continuous optimisation problems. However, TSP and OPR problems are permutation problems. Therefore, a new method for encoding solutions inside the population is required. Random index encoding is the best choice in this case.

4.2.1 Random Key Encoding

The random index encoding process is illustrated in Fig.9. Assume that the TSP solution contains five cities in the following order (1, 2, 3, 4, and 5). Random real numbers will be generated and assigned to these cities, for example (0.7, 0.6, 0.9, 0.3, and 0.75). These real random numbers will be sorted, and consequently, the new order of the cities will be (4, 2, 1, 5, 3), and the corresponding real indices will be (0.3, 0.6, 0.7, 0.75, 0.9).

4.2.2 Update the random key using Lévy flight Random walk

As previously explained, LFRW can be applied only on real solutions, not the permutation solutions, such as TSP and OPR solutions. This problem is solved using random key encoding, where each city has a corresponding real

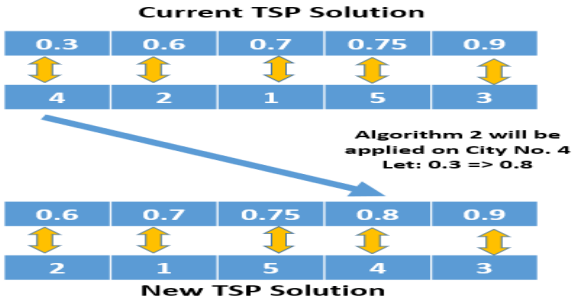


Fig. 10: Application of Algorithm 2 on Random Keys

random number that varies from 0 to 1. To update the permutation solution, the LFRW is applied to the real random index that corresponds to the city. This will lead to a new key that will change the order of the city, as shown in Fig. 10.

As seen in Fig. 10, if LFRW is applied to city number 4, the real index of city number 4 will be modified (i.e., from 0.3 to 0.8). The real indices are sorted as (0.6, 0.7. 0.75, 0.8, 0.9). Hence, the new TSP solution is (2, 1, 5, 4, 3). We refer to this process as the damped Lévy flight on a random key. This process is explained in detail in Algorithm 2.

As seen in Algorithm 2, the process starts at Line 4 by selecting random m cities from the solution x_i. The value of m will be high (90% of the cities) because this process is performed on the worst solutions. Therefore, we aimed to explore different solutions, and many cities should be exchanged. From Lines 7 to 14, it is necessary to generate a new order of cities based on changing the real index using Eq. (9). Then, x_i^{new} will be the new solution obtained from Damped LFRW with random key representation.

$$c_{new}^{real} = c^{real} + \alpha_0 \frac{\phi \times \mu}{|v|^{\frac{1}{\beta}}}(c^{real} - c_{best}^{real}) \tag{9}$$

4.2.3 Adaptive Lévy flight Step Size

The parameter α_0 in Eq.(9) is called Lévy flight step size. The adaptive step size of the Lévy flight is more effective than the fixed value. The best adaptive Lévy flight step size is defined by Eq. (3). Therefore, it is used in the proposed algorithm to make the step size adaptive to the generation number[46].

The effect of adaptive step size can be explained as follows. Earlier, the solutions were not sufficiently good, so we needed a large step size. On moving to the last generation, the quality of the solutions is increased, so the step size will be smaller. Therefore, using an adaptive step size is expected to improve the performance of the proposed algorithm. The adaptive step size together with the random-key encoding will be used in the large step size of Lévy flight as shown in Algorithm2.

Algorithm 2: Damped Lévy flight on a Random Key algorithm.

Let the best solution be x_{best} ;
Define permutation solution x_i that contains the order of d cities;
Define the random-indexed solution x_i^{real} as having a real number $\in$ [0,1] corresponding to each city;
Set $m = ceil(0.9 * d)$, where m is the number of selected cities;
Select m cities randomly from x_i and store them in L;
Get the corresponding real indices of the selected cities from x_i^{real} and store them in L^{real};
for $k \leftarrow$ *1 to* m **do**
    Get the next city, $c = L[k]$;
    Get the corresponding real index for the city from x_i^{real} and store it in c^{real} ;
    Get the corresponding real index for the city from x_{best} and store it in c_{best}^{real} ;
    Update c^{real} via the adapted Lévy flight by using Eq.(9);
    Ensure that c^{real} is within the bounds [0,1] ;
end
Sort the updated c^{real} ;
Find the corresponding order of the cities based on the new order of c^{real} and store it in x_i^{new};
x_i^{new} is the new solution obtained after applying the damped LFRW;

4.3 Modification 5: Concept of Dividing Population and Mutation Operators

In the original continuous CSA, there are two major steps: the first one is to update the solutions based on the Lévy flight random walk; the second one is to mutate the solutions that are discovered with a probability P_a. The method of mutation in the continuous CSA algorithm is called biased selection random walk (BSRW). In the proposed DDCS, BSRW is not applicable to permutation solutions. Therefore, a new modification is required to enable the proposed solution to mutate the discovered solutions.

Three mutation operators are applied to the discovered solution. This means that the discovered solutions are divided into subgroups, and each subgroup contains four solutions. For each sub-group, the solutions are sorted based on the fitness value. The best solution is maintained without mutation. The other three solutions are mutated by three different mutation operators: inversion mutation, pairwise swap mutation, and displacement mutation [51],[11].

The mutation operator is the most important operator in optimisation [22], [21], [19], [20]. Therefore, using three types of mutation operators adds more diversity to the population. These steps are illustrated in Algorithm 3.

The flowchart in Fig. 11 shows the complete steps of the proposed DDCS. First, the initial population is generated and the initial fitness of the population is calculated, which represents the total distance of the solution. Second, random key encoding is applied to the population to generate random keys for each solution in the population.

The iteration of the algorithms begin by updating the adaptive Lévy Flight step size using Eq. (3). The population is then sorted based on its fitness values

Algorithm 3: Mutation operators on sub-groups of the population.

```
Divide the population into k sub-groups, and each group G_k contains four nests;
foreach sub-group G_k do
    Sort the sub-group so that the best nest is at the top;
    foreach nest x_j in the sub-group G_k do
        if j == 1 then
            keep the best solution x_j
        else if j == 2 then
            apply inversion mutation on x_j
        else if j == 3 then
            apply pairwise swap mutation on x_j
        else
            apply displacement mutation on x_j
        end
    end
end
```

from best to the worst. Subsequently, the population is divided into two groups. The first half uses the 2-opt moves of the Lévy flight small step size, while the second half will be updated using Lévy Flight discussed in Algorithm 2.

The next phase checks whether the solutions are discovered with the probability of discovery P_a. If a solution is discovered, the population is divided into subgroups. Each sup-group encounters a different mutation operator, as discussed in Algorithm 3. The last step in the iteration is to update the best-obtained solution, that is, the global minimum. The results are then visualised.

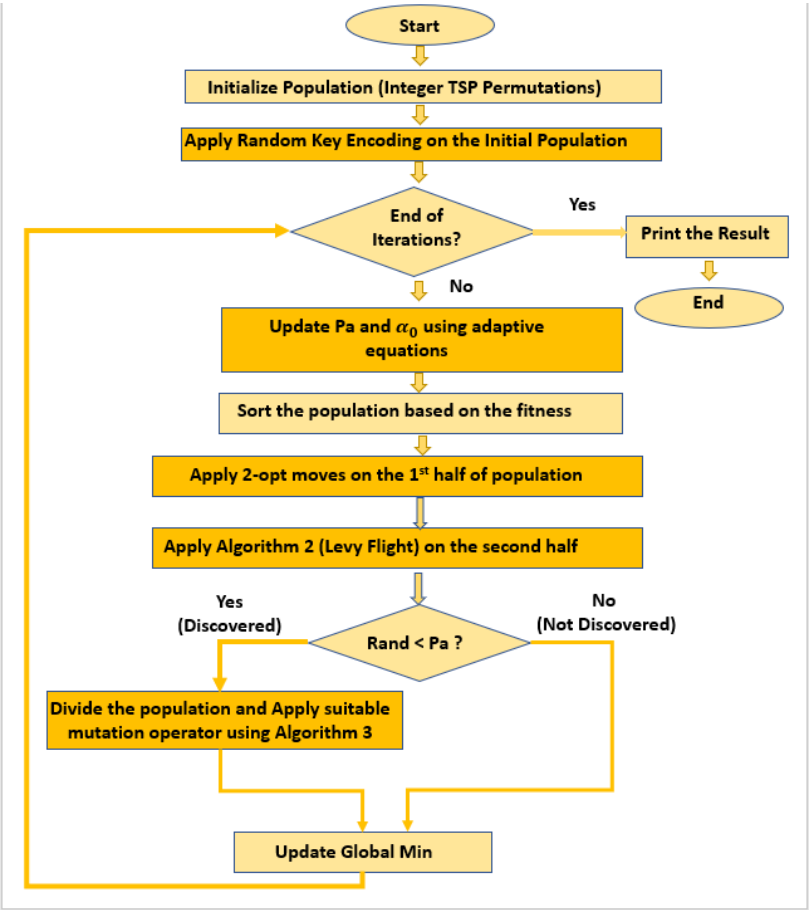


Fig. 11: Complete Steps of the Proposed DDCS Algorithm

5 Model Assumptions and Parameter Settings

The proposed DDCS algorithm is compared to ACO, PSO, and GA. These algorithms are tested on two types of problems. The first one is the traditional TSP benchmark used to test the performance and validity of the algorithm. The second problem is the customised warehouse OPR problem. In this section, the parameter settings of the algorithms and problems are discussed.

5.1 Parameter settings of the Algorithms

The proposed DDCS algorithm has been applied to some of the most common benchmark TSPs[48]. In addition, the algorithm is applied to large-scale warehouse order picking problems. The performance of the algorithm is compared

to that of the ACO, PSO, and GA. The population size is set to 30 for both the algorithms. The algorithms are applied to each problem for 20 independent runs to overcome randomness and statistically verify the results. In each run, the algorithms are terminated after 2000 iterations.

The parameter settings for the algorithms were set to the values recommended by the corresponding authors. The ACO algorithm was implemented to solve TSP in [31]. Similarly, the PSO algorithm was implemented to solve TSP in [52]. Furthermore, the GA implementation code used in the TSP test is described in [32]. The parameters of the GA were set based on De Jong's guidelines [13]. The parameters of the proposed DDCS algorithms: α_{min}, α_{max}, t, w, and P_a were set to 0.1, 0.5, 5, 3, and 0.05, respectively, as recommended in [46]. All the parameter settings are summarised in Table 4.

Table 4: Parameter Settings of the algorithms (ACO, PSO, GA, DDCS).

Algorithm Name	Parameter Name	Parameter Symbol	Value
Shared Parameters	Population size	nPop	30
	Max. No. of iterations	maxIter	2000
	No. of independent runs	nRuns	20
ACO	Evaporation coefficient	e	0.15
	Effect of ants' sight	α	1
	Trace's effect	beta	4
	Primary tracing	t	0.0001
	Common cost elimination	el	0.97
PSO	Inertia Weight	w	0.9
	Damped Inertia	w_{damp}	0.95
	Personal Learning Coefficient	α	0.85
	Global Learning Coefficient	β	0.85
GA	Crossover Probability	Pc	0.9
	Mutation Probability	Pm	0.01
DDCS	Min. Levy step size	α_{min}	0.1
	Max. Levy step size	α_{max}	0.5
	Damping parameter	τ	5
	Damping frequency	ω	3
	Probability of Discovery	Pa	0.05

5.2 TSP benchmark problems and Warehouse Test Cases

Before applying the proposed DDCS algorithm, it was tested on a few common benchmark TSPs. All of these problems were defined in [48]. Ten problems with different dimensions and real data were selected. These functions are summarised in Table5 and are labelled F1 to F10. The X-Y coordinates and distances between each city in each problem represent a real dataset.

The performance of the proposed algorithm will also be verified on a large set of combinations of large-scale OPR problems inside the warehouse and for finding the optimal path. The test cases are generated by constructing different

warehouses with different number of blocks and applying the algorithm to pick orders with different number of items.

We consider conventional warehouses with multiple blocks. The number of blocks of the assumed warehouses was 100, 400, 900, and 2500 for large warehouses. The problem dimension depends on the number of items in the customer order. The test cases included up to 600 items within one order in the 2500-blocks warehouse. The test cases for different warehouse configurations are summarised in Table 5, and are labelled from F11 to F28.

Table 5: TSP benchmark functions and warehouse order picking Test cases

Benchmark Fn. Name	No. of Cities (nd)	Fn. Label
ATT48	48	F1
DANTZIG42	42	F2
HA30	30	F3
KN57	57	F4
LAU15	15	F5
SGB128	128	F6
SP11	11	F7
UK12	12	F8
WG22	22	F9
WG59	59	F10
No. of Blocks in Warehouse	No. of Items	Fn. Label
100	10	F11
	30	F12
	50	F13
	100	F14
400	10	F15
	30	F16
	50	F17
	100	F18
900	10	F19
	30	F20
	50	F21
	100	F22
2500	100	F23
	200	F24
	300	F25
	400	F26
	500	F27
	600	F28

6 Simulation Results

In this section, we discuss the simulation results both statistically and graphically. The convergence and performance of the proposed algorithm was compared with those of ACO, PSO, and GA. These algorithms were tested on complex TSPs and high-dimensional warehouse OPR problems.

6.1 Simulation results (Best, Worst, Mean, Median, and STD)

The proposed DDCS algorithm was applied to all the test cases explained in the previous section. The performance of the proposed algorithm was compared with that of ACO, PSO, and GA. To overcome randomness, each algorithm was executed for 20 independent runs for each function. The best solution for each run, which had the minimum total path distance, was recorded to obtain the 20 best solutions for 20 independent runs. The best, worst, median, mean, and standard deviation (STD) of the distance for all these solutions were then calculated. Table6 shows the results for all test cases from F1 to F28. The best values among the four algorithms are indicated in bold font. These results need to be meaningful, and hence, the results of statistical analysis will be shown in the next section.

6.2 Statistical Analysis of the results

Statistical analysis was performed to determine the best of the four algorithms, namely, ACO, PSO, GA, and the proposed DDCS. Three different statistical analysis methods were used in this study. Each method was applied independently to all the five metrics listed in Table 6. Initially, the Friedman test was used to find the best algorithm among the four algorithms. Subsequently, the Friedman test was followed by the sign test and Wilcoxon signed-rank test to hold paired comparison between the proposed DDCS and each of the other algorithms.

6.2.1 The Friedman Test

The most decisive test in this statistical analysis is the Friedman test, which compares all the algorithms at the same time. The Friedman test not only determines the best algorithm after comparison but also considers all the other algorithms and their rank in each comparison. For each function, the algorithm with the best mean value has a rank of 1. The worst algorithm has a rank of 4. The summation and average of the ranks are then calculated for each algorithm, as shown in Table 7.

The threshold decision value is obtained from chi-square distribution using a degree of freedom equal to three, and a level of significance α of 0.05 to obtain a threshold equal to 11.34. The null hypothesis is assumed such that all four algorithms have similar performance. If the null hypothesis is rejected, this means that the algorithm with the least average rank in the Friedman test performs significantly better than all other algorithms.

Eq. (10) is applied for the worst solution metric, and the F_r value is 42.45, which is more than 11.34. Therefore, the null hypothesis is rejected, and the proposed DDCS algorithm has the best dominant performance over the other algorithms because it has the minimum average rank far from the other algorithms.

Table 6: Results of the ACO, PSO, GA, and DDCS algorithms on functions (F1-F28).

Fn.	Alg.	Best	Worst	Median	Mean	STD
F1	ACO	4.39E+04	4.92E+04	4.69E+04	4.69E+04	1.50E+03
	PSO	1.05E+05	1.14E+05	1.11E+05	1.11E+05	2.53E+03
	GA	3.39E+04	**3.54E+04**	3.47E+04	3.46E+04	**4.09E+02**
	DDCS	**3.37E+04**	3.57E+04	**3.43E+04**	**3.44E+04**	5.36E+02
F2	ACO	8.33E+02	1.01E+03	9.44E+02	9.36E+02	4.43E+01
	PSO	2.03E+03	2.27E+03	2.18E+03	2.17E+03	7.01E+01
	GA	7.05E+02	**7.53E+02**	**7.18E+02**	7.25E+02	**1.47E+01**
	DDCS	**6.99E+02**	7.72E+02	7.31E+02	**7.32E+02**	1.98E+01
F3	ACO	5.42E+02	6.03E+02	5.66E+02	5.67E+02	1.59E+01
	PSO	1.03E+03	1.21E+03	1.15E+03	1.13E+03	5.22E+01
	GA	**5.03E+02**	**5.30E+02**	**5.05E+02**	**5.09E+02**	**8.28E+00**
	DDCS	**5.03E+02**	5.54E+02	**5.05E+02**	5.18E+02	1.82E+01
F4	ACO	1.74E+04	2.03E+04	1.89E+04	1.89E+04	7.19E+02
	PSO	4.27E+04	4.69E+04	4.50E+04	4.48E+04	1.08E+03
	GA	1.33E+04	1.45E+04	1.39E+04	1.39E+04	2.99E+02
	DDCS	**1.31E+04**	**1.40E+04**	**1.35E+04**	**1.35E+04**	**2.56E+02**
F5	ACO	**2.91E+02**	3.19E+02	2.99E+02	3.00E+02	9.23E+00
	PSO	3.63E+02	4.27E+02	3.96E+02	3.93E+02	1.81E+01
	GA	**2.91E+02**	**2.91E+02**	**2.91E+02**	**2.91E+02**	**0.00E+00**
	DDCS	**2.91E+02**	3.07E+02	**2.91E+02**	2.94E+02	6.57E+00
F6	ACO	3.17E+04	3.74E+04	3.50E+04	3.49E+04	1.50E+03
	PSO	1.30E+05	1.38E+05	1.34E+05	1.34E+05	2.44E+03
	GA	3.64E+04	4.15E+04	3.95E+04	3.95E+04	1.40E+03
	DDCS	**2.96E+04**	**3.24E+04**	**3.08E+04**	**3.08E+04**	**7.35E+02**
F7	ACO	1.33E+02	1.45E+02	1.38E+02	1.38E+02	3.19E+00
	PSO	1.33E+02	1.38E+02	1.34E+02	1.34E+02	1.45E+00
	GA	1.33E+02	**1.33E+02**	**1.33E+02**	**1.33E+02**	**0.00E+00**
	DDCS	1.33E+02	**1.33E+02**	**1.33E+02**	**1.33E+02**	**0.00E+00**
F8	ACO	1.73E+03	2.08E+03	**1.73E+03**	1.79E+03	1.06E+02
	PSO	1.73E+03	1.84E+03	1.80E+03	1.79E+03	3.45E+01
	GA	1.73E+03	**1.80E+03**	**1.73E+03**	**1.74E+03**	**1.50E+01**
	DDCS	1.73E+03	1.84E+03	**1.73E+03**	1.75E+03	3.48E+01
F9	ACO	7.86E+02	9.07E+02	8.47E+02	8.45E+02	3.25E+01
	PSO	1.23E+03	1.44E+03	1.37E+03	1.36E+03	6.08E+01
	GA	**7.81E+02**	**7.98E+02**	**7.81E+02**	**7.84E+02**	**6.23E+00**
	DDCS	**7.81E+02**	8.34E+02	**7.81E+02**	7.90E+02	1.33E+01
F10	ACO	1.37E+03	1.62E+03	1.53E+03	1.52E+03	6.00E+01
	PSO	3.50E+03	3.84E+03	3.75E+03	3.72E+03	9.34E+01
	GA	1.03E+03	1.14E+03	1.08E+03	1.08E+03	3.63E+01
	DDCS	**1.01E+03**	**1.09E+03**	**1.06E+03**	**1.05E+03**	**2.42E+01**
F11	ACO	3.60E+02	4.20E+02	3.60E+02	3.64E+02	1.39E+01
	PSO	**3.40E+02**	**3.60E+02**	**3.40E+02**	3.41E+02	4.47E+00
	GA	3.60E+02	**3.60E+02**	3.60E+02	**3.60E+02**	**0.00E+00**
	DDCS	4.00E+02	4.00E+02	4.00E+02	4.00E+02	**0.00E+00**
F12	ACO	6.80E+02	1.22E+03	8.10E+02	8.09E+02	1.10E+02
	PSO	1.20E+03	1.30E+03	1.26E+03	1.26E+03	3.18E+01
	GA	**6.00E+02**	**6.60E+02**	**6.00E+02**	**6.16E+02**	2.11E+01
	DDCS	6.60E+02	7.20E+02	6.80E+02	6.81E+02	**1.52E+01**
F13	ACO	1.06E+03	1.22E+03	1.16E+03	1.15E+03	3.68E+01
	PSO	2.60E+03	2.80E+03	2.71E+03	2.71E+03	4.67E+01
	GA	8.40E+02	9.40E+02	8.60E+02	8.64E+02	2.72E+01
	DDCS	**8.00E+02**	**8.80E+02**	**8.40E+02**	**8.29E+02**	**2.00E+01**
F14	ACO	1.86E+03	2.20E+03	2.08E+03	2.06E+03	9.22E+01
	PSO	5.60E+03	5.92E+03	5.78E+03	5.77E+03	8.47E+01
	GA	1.44E+03	1.60E+03	1.54E+03	1.53E+03	4.66E+01
	DDCS	**1.22E+03**	**1.40E+03**	**1.28E+03**	**1.29E+03**	**4.22E+01**

$$F_r = \frac{12}{nk(k+1)} \sum_{j=1}^{k} R_j^2 - 3n(k+1). \tag{10}$$

For the best solution metric, the F_r value is 50.96, which is greater than 11.34. Therefore, the null hypothesis is rejected, and the algorithm with the minimum average rank is deemed the best. As shown in Table 7, the proposed

Table 6: Best, worst, median, mean. and STD results of the ACO, PSO, GA, and DDCS algorithms on functions (F1-F28) (continued).

Fn.	Alg.	Best	Worst	Median	Mean	STD
F15	ACO	8.40E+02	9.80E+02	8.60E+02	8.74E+02	3.79E+01
	PSO	**7.20E+02**	**7.20E+02**	**7.20E+02**	**7.20E+02**	0.00E+00
	GA	7.80E+02	7.80E+02	7.80E+02	7.80E+02	0.00E+00
	DDCS	**7.20E+02**	**7.20E+02**	**7.20E+02**	**7.20E+02**	0.00E+00
F16	ACO	1.30E+03	1.52E+03	1.42E+03	1.42E+03	5.27E+01
	PSO	2.42E+03	2.68E+03	2.60E+03	2.59E+03	7.03E+01
	GA	1.38E+03	1.48E+03	1.42E+03	1.42E+03	3.20E+01
	DDCS	**1.14E+03**	**1.18E+03**	**1.14E+03**	**1.14E+03**	**9.79E+00**
F17	ACO	1.90E+03	2.22E+03	2.06E+03	2.05E+03	8.91E+01
	PSO	5.22E+03	5.50E+03	5.32E+03	5.34E+03	8.04E+01
	GA	1.54E+03	1.78E+03	1.62E+03	1.62E+03	6.34E+01
	DDCS	**1.48E+03**	**1.64E+03**	**1.54E+03**	**1.54E+03**	**4.60E+01**
F18	ACO	3.32E+03	3.80E+03	3.54E+03	3.57E+03	1.14E+02
	PSO	1.05E+04	1.11E+04	1.09E+04	1.08E+04	1.71E+02
	GA	2.64E+03	3.06E+03	2.91E+03	2.90E+03	1.12E+02
	DDCS	**2.22E+03**	**2.44E+03**	**2.31E+03**	**2.32E+03**	**6.30E+01**
F19	ACO	1.18E+03	1.18E+03	1.18E+03	1.18E+03	0.00E+00
	PSO	1.38E+03	1.38E+03	1.38E+03	1.38E+03	0.00E+00
	GA	7.80E+02	7.80E+02	7.80E+02	7.80E+02	0.00E+00
	DDCS	**7.40E+02**	**7.40E+02**	**7.40E+02**	**7.40E+02**	0.00E+00
F20	ACO	1.94E+03	2.22E+03	2.01E+03	2.04E+03	8.91E+01
	PSO	4.08E+03	4.54E+03	4.36E+03	4.34E+03	1.37E+02
	GA	**1.74E+03**	**1.86E+03**	**1.77E+03**	**1.78E+03**	3.61E+01
	DDCS	1.86E+03	2.00E+03	1.94E+03	1.93E+03	4.28E+01
F21	ACO	2.48E+03	3.14E+03	2.93E+03	2.87E+03	1.76E+02
	PSO	7.42E+03	8.34E+03	8.02E+03	7.99E+03	2.20E+02
	GA	**2.20E+03**	**2.48E+03**	**2.32E+03**	**2.34E+03**	7.12E+01
	DDCS	2.30E+03	2.54E+03	2.38E+03	2.38E+03	**6.17E+01**
F22	ACO	5.04E+03	5.88E+03	5.59E+03	5.58E+03	2.19E+02
	PSO	1.66E+04	1.74E+04	1.70E+04	1.70E+04	1.91E+02
	GA	3.70E+03	4.50E+03	4.28E+03	4.22E+03	1.85E+02
	DDCS	**3.20E+03**	**3.74E+03**	**3.47E+03**	**3.47E+03**	**1.25E+02**
F23	ACO	8.26E+03	9.54E+03	8.99E+03	8.99E+03	3.49E+02
	PSO	2.63E+04	2.76E+04	2.71E+04	2.70E+04	3.45E+02
	GA	6.46E+03	7.30E+03	6.69E+03	6.78E+03	2.41E+02
	DDCS	**5.18E+03**	**6.02E+03**	**5.48E+03**	**5.52E+03**	**2.10E+02**
F24	ACO	1.51E+04	1.65E+04	1.58E+04	1.58E+04	4.16E+02
	PSO	5.60E+04	5.75E+04	5.69E+04	5.68E+04	4.50E+02
	GA	1.45E+04	1.64E+04	1.53E+04	1.54E+04	4.48E+02
	DDCS	**9.92E+03**	**1.12E+04**	**1.04E+04**	**1.05E+04**	**3.77E+02**
F25	ACO	1.89E+04	2.26E+04	2.11E+04	2.10E+04	8.56E+02
	PSO	8.70E+04	8.95E+04	8.83E+04	8.83E+04	7.48E+02
	GA	2.43E+04	2.67E+04	2.55E+04	2.55E+04	6.55E+02
	DDCS	**1.75E+04**	**1.95E+04**	**1.84E+04**	**1.84E+04**	**6.53E+02**
F26	ACO	**2.49E+04**	**2.72E+04**	**2.61E+04**	**2.61E+04**	6.36E+02
	PSO	1.20E+05	1.24E+05	1.23E+05	1.23E+05	1.12E+03
	GA	3.72E+04	4.01E+04	3.90E+04	3.87E+04	9.94E+02
	DDCS	2.66E+04	2.87E+04	2.80E+04	2.79E+04	**5.89E+02**
F27	ACO	**2.67E+04**	**3.24E+04**	**3.01E+04**	**3.00E+04**	1.16E+03
	PSO	1.53E+05	1.55E+05	1.54E+05	1.54E+05	**6.97E+02**
	GA	5.03E+04	5.38E+04	5.25E+04	5.23E+04	1.06E+03
	DDCS	3.78E+04	4.10E+04	3.95E+04	3.95E+04	7.54E+02
F28	ACO	**3.18E+04**	**3.58E+04**	**3.38E+04**	**3.39E+04**	1.08E+03
	PSO	1.85E+05	1.88E+05	1.87E+05	1.87E+05	9.98E+02
	GA	6.42E+04	6.86E+04	6.66E+04	6.67E+04	9.55E+02
	DDCS	5.01E+04	5.36E+04	5.17E+04	5.17E+04	**8.67E+02**

DDCS has the lowest average rank over all other algorithms; therefore, it is the best algorithm considering the worst solution metric. Concerning the median metric, the F_r value is 48.80, which is greater than 11.34. Therefore, the null hypothesis is rejected, and the proposed DDCS is considered to be the best algorithm because it has the minimum average rank.

Similarly, for the mean and STD result metrics, the F_r values are 48.83 and 48.45, respectively. Both F_r values are greater than 11.34. Therefore, the null

hypothesis is rejected, and the algorithm with the minimum average rank is the best. Hence, the proposed DDCS algorithm outperforms all other algorithms with significant performance for all the metrics: worst, best, median, mean, and STD.

6.2.2 Paired Comparisons using Sign test and Wilcoxon Signed-rank test

It is advisable to perform paired comparisons between the proposed DDCS algorithm and each of the other algorithms separately to determine the significance level of the results. The sign test and Wilcoxon signed-rank test are the methods used to hold paired comparisons.

In the sign test, the "+" sign indicates that the proposed DDCS has won in the paired comparison, the "-" sign means that the DDCS has been defeated, and the "=" sign indicates a draw between the two algorithms. In the Wilcoxon test, R- represents the negative rank summation, which indicates the advantage of DDCS over other algorithms. On the other hand, R+ is the positive rank summation that shows the advantage of the other algorithms over the proposed DDCS algorithm. The P values in the last column, obtained from the SPSS program, show the significance level of the results. The significance level was set at p ¡ 0.05. If the p value obtained from the test is less than 0.05, the results are considered significant.

As shown in Table 7, the proposed DDCS algorithm has the largest number of wins ("+" signs) in every paired comparison in the sign test. Moreover, R- was always greater than R+ in all paired comparisons in the Wilcoxon test. Furthermore, the P values for all the metrics for all paired comparisons were less than 0.05 level of significance. Hence, it is statistically proved that the proposed DDCS algorithm has a significant advantage over all the other algorithms for all the problems.

6.3 Graphic analysis of the results

The graphical analysis is illustrated using three types of plots. The box and violin plots show the distribution of the results, including the mean and median. The convergence graphs show the speed at which the algorithm can reach an optimal solution through iterations.

6.3.1 Box Plots

A box plot is a diagram that shows the variance and distribution of results for each algorithm over independent runs. The horizontal axis represents the algorithm names and the vertical axis represents the fitness value on logarithmic scale. Fitness value represents the shortest path distance. The length of the box represents variance. The distribution of the results has a small standard deviation when the box is short and compact. When the box is in a low position, the results are close to the optimal fitness value. The median value is

Table 7: Statistical Analysis for all the algorithms. The reference of paired comparisons is the proposed DDCS algorithm for all the metrics (Best, Worst, Median, Mean, and STD). In Friedman Test: the lower the average rank, the better the algorithm. The best average rank is marked as bold. In Sign Test: '+' means the proposed DDCS is better. In Wilcoxon Test: R+ < R- means the proposed DDCS is the better. Bold P values refer to significant results in the paired comparison.

Metric	Algorithm	Friedman Test		Sign Test	Wilcoxon Test		
		Sum Rank	Mean Rank	+/=/-	R+	R-	P Val.
Best	ACO	73.5	2.6300	21/3/4	70	255	**0.0130**
	PSO	103.5	3.7000	24/3/1	1	324	**0.0000**
	GA	60.5	2.1600	19/5/4	30	246	**0.0010**
	DDCS	42.5	**1.5200**	NA	NA	NA	NA
Worst	ACO	80	2.8600	25/0/3	73	333	**0.0030**
	PSO	105	3.7500	26/1/1	3	375	**0.0000**
	GA	52	1.8600	17/1/10	73	305	**0.0050**
	DDCS	43	**1.5400**	NA	NA	NA	NA
Median	ACO	76	2.71	23/1/4	73	305	**0.005**
	PSO	105.5	3.77	26/1/1	2	376	**0.000**
	GA	57	2.04	18/5/5	30.5	245.5	**0.001**
	DDCS	41.5	**1.48**	NA	NA	NA	NA
Mean	ACO	77	2.7500	24/0/4	76	330	**0.0040**
	PSO	105.5	3.7700	26/1/1	3	375	**0.0000**
	GA	54.5	1.9500	18/1/9	59.5	318.5	**0.0020**
	DDCS	43	**1.5400**	NA	NA	NA	NA
STD	ACO	92.5	3.3000	26/1/1	1	377	**0.0000**
	PSO	94.5	3.3800	24/2/2	13	338	**0.0000**
	GA	51.5	1.84	17/4/7	65	235	**0.0150**
	DDCS	41.5	**1.4800**	NA	NA	NA	NA

represented by the red line inside the box, which matches the results in Table 6. Therefore, the best algorithm among all others is the one with the most compact and lower box [40].

Box plots for selected high-dimensional TSPs are shown in Fig. 12. The proposed DDCS algorithm has better distribution than ACO, PSO, and GA, where it shows a very consistent and compact distribution for most TSP benchmarks and warehouse path planning problems, especially high-dimensional problems.

6.3.2 Violin Plots

A violin plot represents and visualises numeric data. Violin plots are similar to box plots, as they represent the median, mean, and inter-quarter ranges. However, the violin plot can also display the probability density of the data at different values using kernel density estimation (KDE).

Wider sections of the violin plot represent a higher probability that members of the population will take a given value; the skinnier sections represent a lower probability [58]. Violin plots were plotted to show the entire distribution of the results, including mean and median [30]. Fig. 13 shows the violin plots

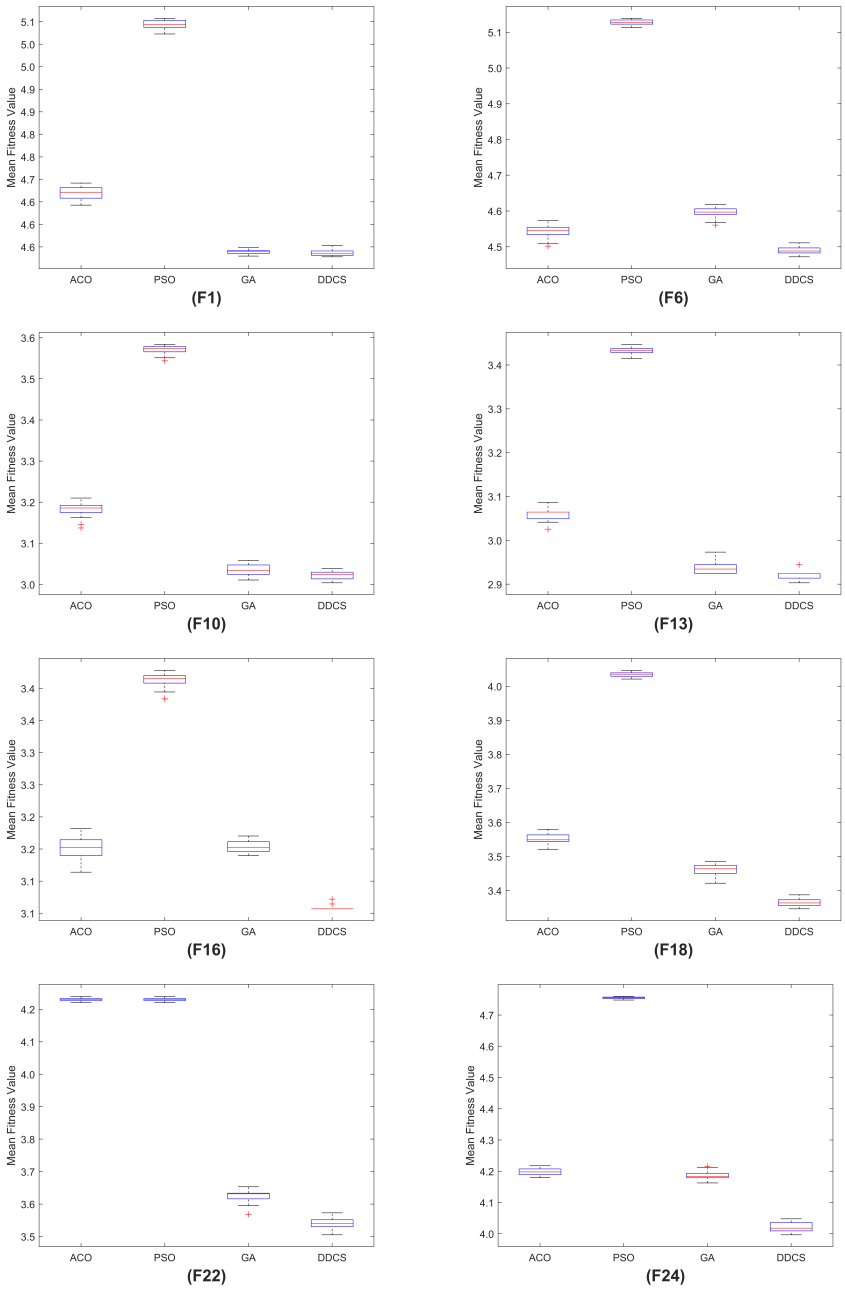


Fig. 12: Box plots for the best total distance in 20 independent runs for ACO, PSO, GA, and DDCS.

for a few high-dimensional TSP benchmark functions. The violin plots for the proposed DDCS algorithm showed the most compact distribution among the other algorithms.

6.3.3 Convergence Graphs

The speed of convergence is one of the most important performance metrics for judging the effectiveness of an algorithm. As explained previously, each algorithm was executed for 20 independent iterations. In each run, the best fitness values obtained were stored after reaching certain percentages of the maximum number of iterations (maxIter). The best solution in each run for each algorithm was recorded after reaching specific steps (0.01 maxIter, 0.02 maxIter,. . . , 0.1 maxIter, 0.2 maxIter, . . . , maxIter). Then, the mean value was calculated for each percentage of the maxIter over 20 runs. These average values were used to construct the average convergence graphs for each algorithm.

In each run, each algorithm was executed for 2000 iterations, and the minimum distance of the path " fitness value " was improved throughout the iterations. The minimum distance was obtained in the final iteration. The convergence graphs show how rapidly these distances decrease with time(iterations). For all algorithms, the convergence graphs display the average fitness value (minimum distance) in 20 runs against the iteration number.

Fig.14 shows the convergence graphs for a set of TSPs, which are selected for high-dimensional problems. It is shown that PSO, depicted by the green curve, has the worst convergence among all the algorithms. PSO has a very slow and poor convergence compared with the other three algorithms. The ACO algorithm, depicted by the red curve, has the best initial fitness value among all the algorithms. However, it has a very slow convergence compared with the other algorithms.

Graphically, it is proved that in almost all TSPs, the speed of convergence for the proposed DDCS algorithm is faster than that of the other algorithms, especially in high-dimensional problems. It starts with a high fitness value, but the cost of the path is reduced to reach the minimum among all other algorithms after performing 2000 iterations.

6.4 Final Path inside Warehouse Plots

The final path obtained from the proposed DDCS algorithm is shown in Fig. 15. The figure shows the final path for the benchmark TSP and is labelled from F1 to F10. The final path for the function F6 is not clear because of the complexity of the TSP for F6 compared with other standard TSPs under the same termination condition (2000 iterations). The TSP for F6 is a high-dimensional problem (128 dimensions) compared with the other TSPs (varying from 11 to 59 dimensions). Therefore, the connections are not clear compared with the nearby figures, as shown in Fig. 15. Additionally, the distribution of

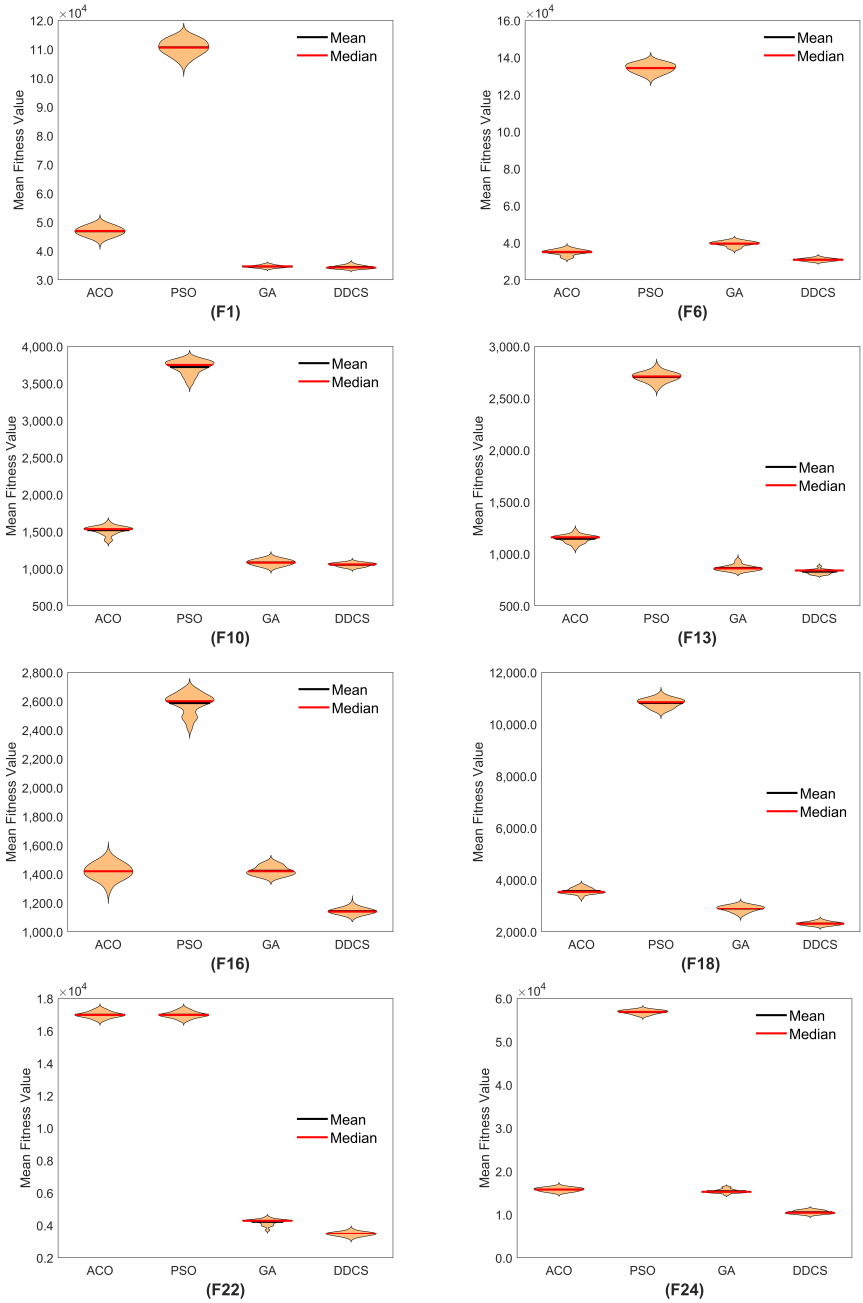


Fig. 13: Violin plots for the best total distance in 20 independent runs for ACO, PSO, GA, and DDCS.

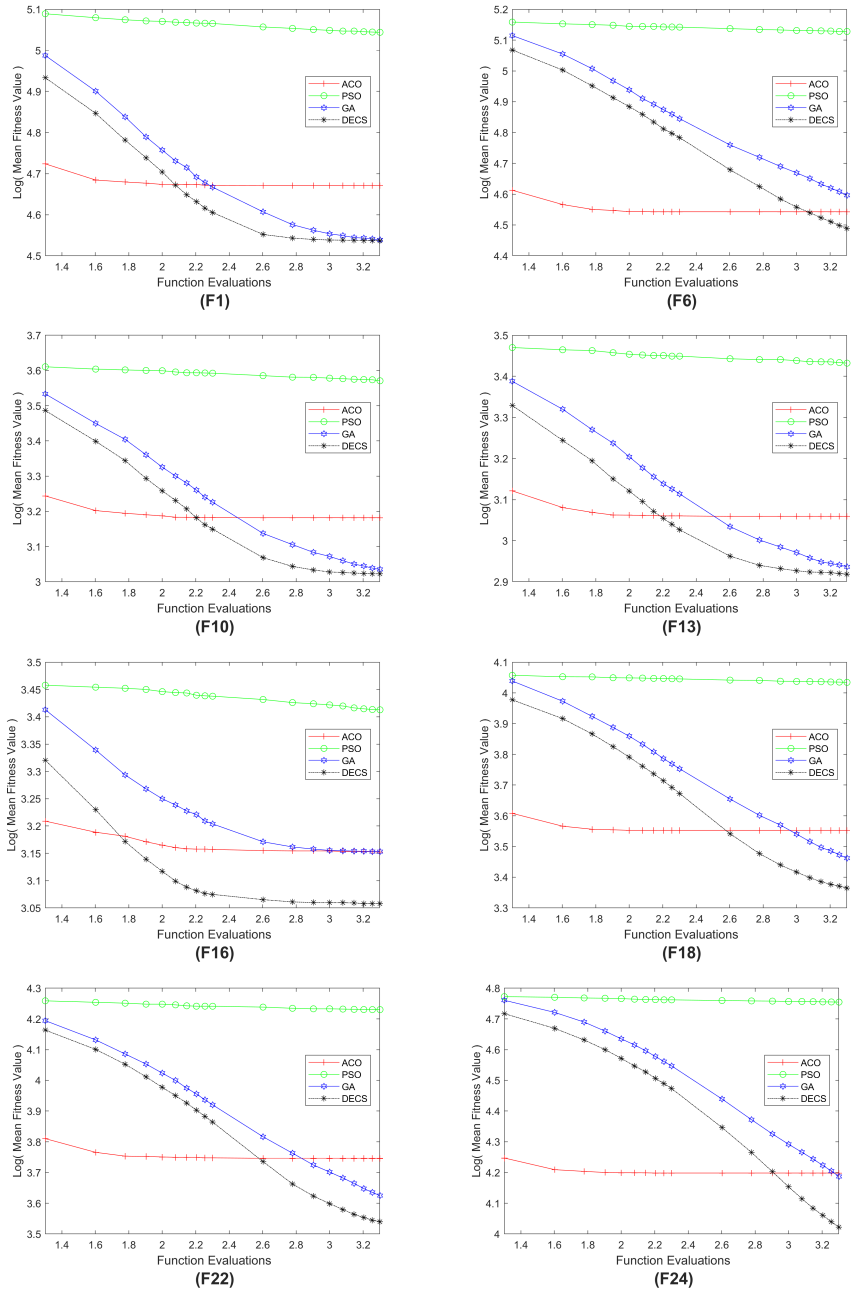


Fig. 14: Convergence graphs for the mean path distance over 20 runs.

the 128 cities of F6 is scattered in an extremely narrow area, so finding the optimal route for this problem in just 2000 iterations is not easy. In contrast, for F9, which has a simpler distribution of cities, finding the optimal route for this problem in 2000 iterations is easier.

Regarding path planning inside a warehouse, there are different warehouse configurations based on the number of blocks inside the warehouse. In each warehouse, it is required to execute a batch of orders. In Fig. 16, the blue star marks the starting point, while the red square marks the endpoint. Fig.16a shows the final path for a 100-block warehouse with different number of items for each order. F13 refers to an OPR problem for a warehouse with 100 blocks, where the number of items to be picked is 50. Whereas, F14 is an OPR problem for a warehouse with 100 blocks, where the number of items to be picked is 100.

Fig.16b shows the final path for a 400-block warehouse with different number of items for each order. The left figure is for F17, which refers to an OPR problem for a warehouse with 400 blocks, and the number of items to be picked is 50. The right figure is for F18, which refers to an OPR problem for a warehouse with 400 blocks, and the number of items to be picked is 100.

The final path for a warehouse with 900 blocks is shown in Fig.16c. The left figure shows F21, which refers to an OPR problem for a warehouse with 900 blocks, and the number of items to be picked is 50. The right figure shows F22, which refers to an OPR problem for a warehouse with 900 blocks, and the number of items to be picked is 100.

Fig. 16d shows the final path inside the largest warehouse with 2500 blocks. The left figure illustrates F24, which indicates an OPR problem for a warehouse with 2500 blocks, and the number of items to be picked is 200; while F26 is an OPR problem for a warehouse with 2500 blocks, and the number of items to be picked is 400.

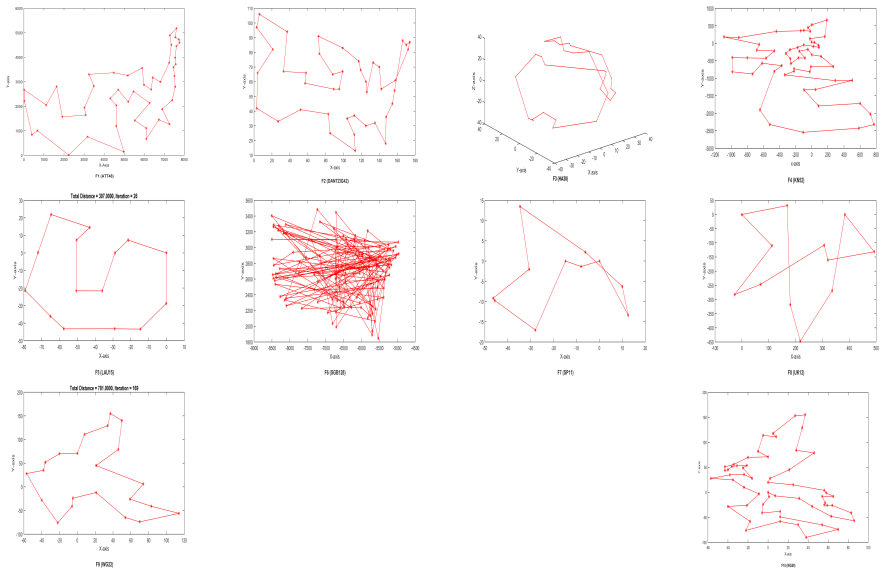
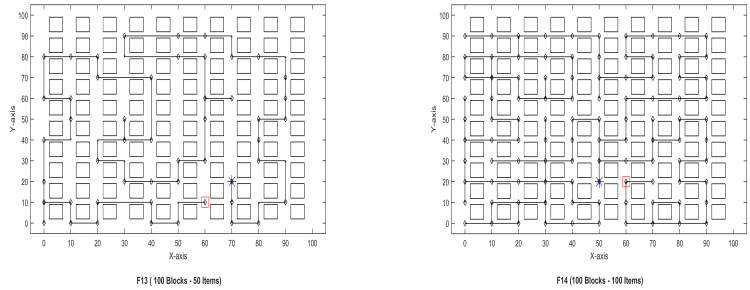
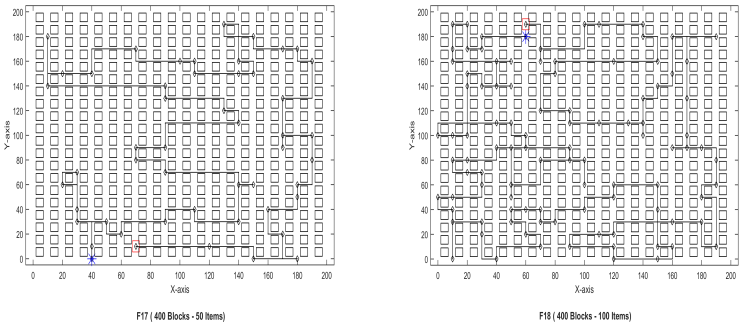


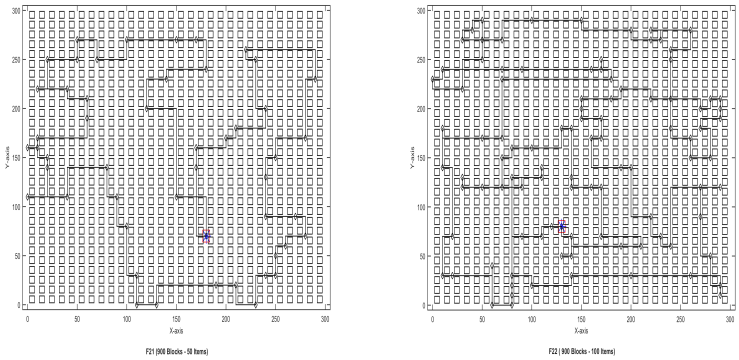
Fig. 15: Final Path for TSPs (F1–F10).



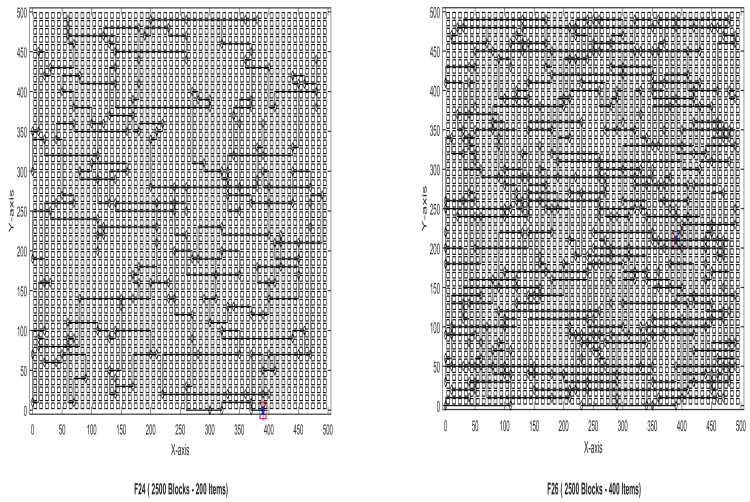
(a) Final Path for 100-block Warehouse (F13–F14).



(b) Final Path for 400-block Warehouse (F17–F18).



(c) Final Path for 900-block Warehouse (F21–F22).



(d) Final Path for 2500-block Warehouse (F24–F26).

Fig. 16: Final Path Graphs inside Different Warehouse Configurations. Blue star marks the starting point, while the red square marks the end point

7 Warehouse Application

This section presents the software and hardware for prototype implementation. The first part discusses the warehouse layout design, while the second part discusses a simple prototype hardware implementation and system structure. The actual path of the prototype is illustrated at the end of this section.

7.1 Warehouse Layout Design

The implemented warehouse layout consists of nine blocks and a set of aisles that intersect, as shown in Fig.17. The black lines represent the aisles on which the trolley will move. The squares inside these black lines contain blocks. However, the main focus here is path planning; therefore, we do not represent the blocks. The intersection points can be marked using QR and/or RFID tags to identify the current position of the trolley by adding information about the point itself and its neighbours [56][65]. This will be suitable for large warehouses such as an Amazon warehouse. In the proposed implementation, we do not need to add the neighbours' information for each point because the robot knows exactly all the points that should pass from the beginning to the endpoint. Therefore, it is necessary to know only the current position and not the neighbour information.

There are some limitations and assumptions regarding this prototype. The order is entered statically using a simple graphical user interface (GUI). It is assumed that there is only one trolley in the prototype warehouse. Therefore, collaboration between different trolleys is beyond the objective of this study. The physical warehouse is assumed to have nine blocks for design simplicity. The procedure applied to the nine blocks is the same as that to a larger warehouse. A trolley can take one order at a time. Furthermore, batch organisation is beyond the scope of this study. The primary focus is on path planning.

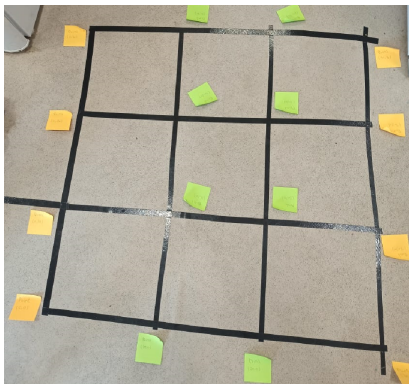


Fig. 17: The Physical layout of a 9-block warehouse.

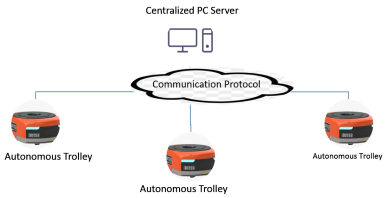


Fig. 18: The three main components of a warehouse.

7.2 Warehouse System Structure

The warehouse is organised into three main components: a centralised server, an autonomous trolley, and the communication protocol between the centralised server and autonomous trolley. The relationship between them is shown in Fig.18.

The Centralized Server

The first component is a centralised server. This is simply a computer system where all the processing is performed. The proposed DDCS algorithm is executed on this server because it is iterative and requires a strong processor. It receives the items ordered by the customer via a GUI, as shown in Fig.20. Next, the proposed algorithm is executed on the required items to determine the shortest path between all the items. Finally, it sends points of the path to the trolley.

The Autonomous Trolley

The second component is an autonomous trolley, as shown in Fig.19. The trolley is a mobile robot with a single job. The job is to follow the path received from the centralised server. It then picks the items from the warehouse shelves. The physical trolley consists of a set of main components: physical body, sensors, actuators, and controller. The actuators used in this trolley are DC motors. These motors control the movement in all directions. The sensors used in this trolley are IR sensors that are used in line following the system. IR sensors operate by transmitting infrared radiation and measuring reflected radiation. The black line does not reflect radiation but absorbs it, whereas the white line reflects the radiation. This is the key concept of line-following. Five IR sensors are used in the trolley. Based on the alignment of these sensors on the black line, the deviation of the robot can be determined. Hence, the trolley can decide in which direction it should move to keep itself on the line.

This robot has a simple algorithm to follow a line and detect only the current location. Therefore, a small inexpensive microcontroller, such as Arduino Uno, can be used. Complex processing is centralised on the server. The sensors and motors are connected to the Arduino pins. The code that manages

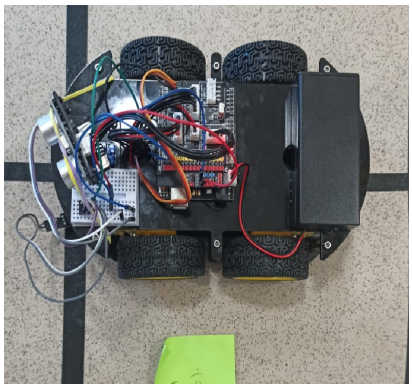


Fig. 19: The physical prototype of the trolley.

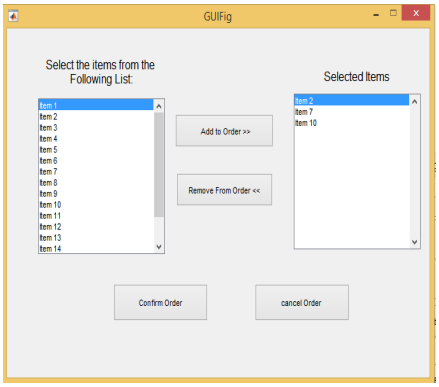


Fig. 20: Placing the order using the GUI

the line-following algorithm is written in C language. After gathering all these components, the final prototype of the trolley is shown in Fig.19.

The Communication Protocol

The third component is the communication protocol. The protocol used in this prototype is serial communication via a Bluetooth module. This method is simple and suitable for use in small warehouses. In the case of large warehouses, a WIFI module can be used instead. This protocol transfers the path points from the server to the chosen robot.

Advantages of the Proposed Structure

The structure of an autonomous warehouse has several advantages. The most important advantage is the centralised processing on the server. This is because the proposed algorithm requires a powerful processor that cannot easily

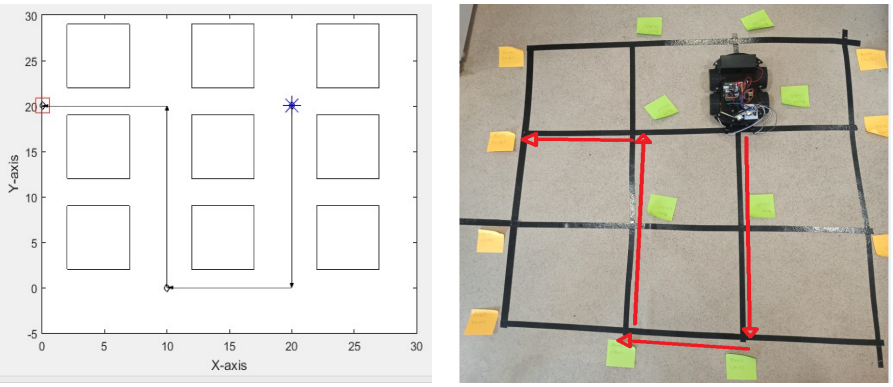
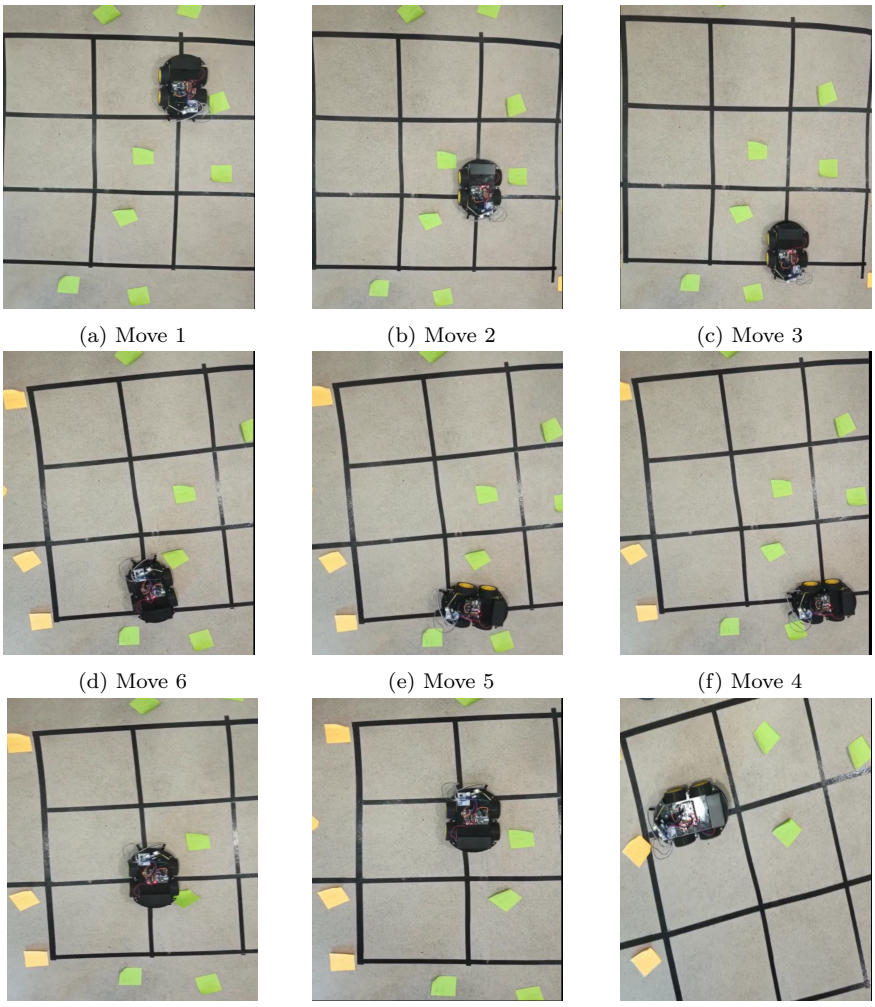


Fig. 21: The expected path from the proposed algorithm that should be followed by the autonomous trolley.

be executed on any microcontroller. Therefore, fast execution of the algorithm on the server saves considerable time. In addition, the trolley has a simple code on its controller, which includes line-following code and detection of its location. This simplifies the manufacturing process of this robot, and inexpensive microcontrollers can be used for these robots. This will lead to ease of extension because the order distribution process is centralised and does not depend on robots. Therefore, many trolleys can be added to the system, all of which can be connected to the same central unit. The maintenance of the system is easier because all the robots have the same code on their controllers, and the main code responsible for executing the proposed algorithm is centralised on the server.

7.3 Physical warehouse results

An order of three items is placed. This order contains items 2, 7, and 10. The order is placed using GUI on the server, as shown in Fig.20. The proposed algorithm is executed, and it generates the expected path, as shown in Fig.21. The trolley receives this path via serial communication, and it begins to move on this path on the actual warehouse prototype. After the order is received by the autonomous trolley, it follows the expected path. The steps of the trolley are shown in Fig.22.



(a) Move 1 (b) Move 2 (c) Move 3

(d) Move 6 (e) Move 5 (f) Move 4

(g) Move 7 (h) Move 8 (i) Move 9

Fig. 22: The actual movement of the trolley inside the warehouse prototype.

8 Conclusion

This study aimed to solve the OPR problem for an autonomous trolley inside a warehouse. This problem is defined as a TSP. A new algorithm, named DDCS, is proposed to solve the OPR problem for autonomous trolleys inside a warehouse. Various modifications were made to the standard CSA. Random key encoding was used to represent TSP solutions. The 2-opt moves and adaptive LFRW were used to update the solutions. Moreover, different mutation techniques were used for different population subgroups. The proposed DDCS algorithm was customised to solve the OPR problem with diagonal movement

constraints, which was solved using the concepts of intersection and corner points.

The proposed DDCS was compared with the GA, ACO, and PSO algorithms on a set of common benchmark TSPs, in addition to different warehouse OPR problems. The simulation and practical results show that the proposed DDCS algorithm outperforms all other algorithms, especially in large dimensions. A physical centralised architecture was assumed, which saves time and cost because bulky processing is performed on a centralised server. Furthermore, trolleys can use inexpensive controllers because they do not have a high processing load. Therefore, this system is suitable for large warehouses, such as Amazon and royal mail warehouses.

In the future, a parametric study can be conducted on the parameters of the DDCS algorithm, such as discovery probability, to test their impact on performance. The proposed algorithm can be extended to more real-time applications, such as PCB drilling process and other permutation applications. This algorithm can be hybridised with other algorithms, and its performance can be tested on various types of applications.

Conflict of interest

The authors declare that they have no conflicts of interest.

References

1. Abdel-Kader RF (2011) Fuzzy particle swarm optimization with simulated annealing and neighborhood information communication for solving tsp. International Journal of Advanced Computer Science and Applications 2
2. Arora S (1998) Polynomial time approximation schemes for euclidean traveling salesman and other geometric problems. Journal of the ACM (JACM) 45(5):753–782
3. Bean JC (1994) Genetic algorithms and random keys for sequencing and optimization. ORSA journal on computing 6(2):154–160
4. Bonomi E, Lutton JL (1984) The n-city travelling salesman problem: Statistical mechanics and the metropolis algorithm. SIAM review 26(4):551–568
5. Çelik M, Süral H (2016) Order picking in a parallel-aisle warehouse with turn penalties. International Journal of Production Research 54(14):4340–4355
6. Chabot T, Lahyani R, Coelho LC, Renaud J (2017) Order picking problems under weight, fragility and category constraints. International Journal of Production Research 55(21):6361–6379
7. Chen F, Wang H, Qi C, Xie Y (2013) An ant colony optimization routing algorithm for two order pickers with congestion consideration. Computers & Industrial Engineering 66(1):77–85

8. Chen F, Xu G, Wei Y (2019) Heuristic routing methods in multiple-block warehouses with ultra-narrow aisles and access restriction. International Journal of Production Research 57(1):228–249
9. Chen H, Li S, Tang Z (2011) Hybrid gravitational search algorithm with random-key encoding scheme combined with simulated annealing. International Journal of Computer Science and Network Security 11(6):208–217
10. Chen SM, Chien CY (2011) Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. Expert Systems with Applications 38(12):14439–14450
11. Chieng HH, Wahid N (2014) A performance comparison of genetic algorithm's mutation operators in n-cities open loop travelling salesman problem. In: Recent Advances on Soft Computing and Data Mining, Springer, pp 89–97
12. Croes GA (1958) A method for solving traveling-salesman problems. Operations research 6(6):791–812
13. De Jong KA, Spears WM, et al. (1989) Using genetic algorithms to solve np-complete problems. In: ICGA, pp 124–132
14. De Koster R, Van Der Poort E (1998) Routing orderpickers in a warehouse: a comparison between optimal and heuristic solutions. IIE transactions 30(5):469–480
15. De Koster R, Le-Duc T, Roodbergen KJ (2007) Design and control of warehouse order picking: A literature review. European journal of operational research 182(2):481–501
16. Dorigo M, Gambardella LM (1997) Ant colonies for the travelling salesman problem. biosystems 43(2):73–81
17. Dwivedi V, Chauhan T, Saxena S, Agrawal P (2012) Travelling salesman problem using genetic algorithm. IJCA Proceedings on Development of Reliable Information Systems, Techniques and Related Issues (DRISTI 2012) 1:25
18. Enright JJ, Wurman PR (2011) Optimization and coordinated autonomy in mobile fulfillment systems. In: Workshops at the twenty-fifth AAAI conference on artificial intelligence
19. Fogel DB (1993) Empirical estimation of the computation required to discover approximate solutions to the traveling salesman problem using evolutionary programming. In: Proceedings of the Second Annual Conference on Evolutionary Programming, Evolutionary Programming Society, La Jolla, CA, pp 56–61
20. Fogel DB (2006) Evolutionary computation: toward a new philosophy of machine intelligence, vol 1. John Wiley & Sons
21. Fogel DB, Atmar JW (1990) Comparing genetic operators with gaussian mutations in simulated evolutionary processes using linear systems. Biological Cybernetics 63(2):111–114
22. Fogel LJ, Owens AJ, Walsh MJ (1966) Artificial intelligence through simulated evolution. Wiley

23. Gandomi AH, Yang XS, Alavi AH (2011) Mixed variable structural optimization using firefly algorithm. Computers & Structures 89(23-24):2325–2336
24. Gandomi AH, Talatahari S, Yang XS, Deb S (2013) Design optimization of truss structures using cuckoo search algorithm. The Structural Design of Tall and Special Buildings 22(17):1330–1349
25. Gandomi AH, Yang XS, Alavi AH (2013) Cuckoo search algorithm: a metaheuristic approach to solve structural optimization problems. Engineering with computers 29(1):17–35
26. Goetschalckx M, Donald Ratliff H (1988) Order picking in an aisle. IIE transactions 20(1):53–62
27. Grefenstette J, Gopal R, Rosmaita B, Van Gucht D (1985) Genetic algorithms for the traveling salesman problem. In: Proceedings of the first International Conference on Genetic Algorithms and their Applications, Lawrence Erlbaum, vol 160, pp 160–168
28. Hall RW (1993) Distance approximations for routing manual pickers in a warehouse. IIE transactions 25(4):76–87
29. Henn S, Koch S, Wäscher G (2012) Order batching in order picking warehouses: a survey of solution approaches. In: Warehousing in the global supply chain, Springer, pp 105–137
30. Hoffmann H, et al. (2015) violin. m-simple violin plot using matlab default kernel density estimation. Katzenburgweg, Germany: INRES (University of Bonn)
31. Ibrahim S (2021) Ant colony optimization (aco) to solve traveling salesman problem (tsp). https://www.mathworks.com/matlabcentral/fileexchange/51113-ant-colony-optimization-aco-to-solve-traveling-salesman-problem-tsp, retrieved: Oct. 2, 2021
32. Kirk J (2020) Traveling salesman problem - genetic algorithm. https://www.mathworks.com/matlabcentral/fileexchange/13680-traveling-salesman-problem-genetic-algorithm, retrieved: May 16, 2020
33. Larranaga P, Kuijpers CMH, Murga RH, Inza I, Dizdarevic S (1999) Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review 13(2):129–170
34. Lenstra JK, Kan AR (1975) Some simple applications of the travelling salesman problem. Journal of the Operational Research Society 26(4):717–733
35. Lin CC, Kang JR, Hou CC, Cheng CY (2016) Joint order batching and picker manhattan routing problem. Computers & Industrial Engineering 95:164–174
36. Malek M, Guruswamy M, Pandya M, Owens H (1989) Serial and parallel simulated annealing and tabu search algorithms for the traveling salesman problem. Annals of Operations Research 21(1):59–84
37. Martin O, Otto SW, Felten EW (1991) Large-step Markov chains for the traveling salesman problem. Citeseer

38. Masae M, Glock CH, Grosse EH (2020) Order picker routing in warehouses: A systematic literature review. International Journal of Production Economics 224:107564
39. Matusiak M, de Koster R, Kroon L, Saarinen J (2014) A fast simulated annealing method for batching precedence-constrained customer orders in a warehouse. European Journal of Operational Research 236(3):968–977
40. McGill R, Tukey JW, Larsen WA (1978) Variations of box plots. The American Statistician 32(1):12–16
41. Ouaarab A, Ahiod B, Yang XS (2014) Discrete cuckoo search algorithm for the travelling salesman problem. Neural Computing and Applications 24(7-8):1659–1669
42. Ouaarab A, Ahiod B, Yang XS (2014) Improved and discrete cuckoo search for solving the travelling salesman problem. In: Cuckoo Search and Firefly Algorithm, Springer, pp 63–84
43. Ouaarab A, Ahiod B, Yang XS (2015) Random-key cuckoo search for the travelling salesman problem. Soft Computing 19(4):1099–1106
44. Parker LE (2007) Distributed intelligence: Overview of the field and its application in multi-robot systems. In: AAAI Fall Symposium: Regarding the Intelligence in Distributed Intelligent Systems, pp 1–6
45. Poudel DB (2013) Coordinating hundreds of cooperative, autonomous robots in a warehouse. Jan 27:1–13
46. Reda M, Haikal AY, Elhosseini MA, Badawy M (2019) An innovative damped cuckoo search algorithm with a comparative study against other adaptive variants. IEEE Access 7(1):119272–119293
47. Reda M, Elhosseini M, Haikal A, Badawy M (2021) A novel cuckoo search algorithm with adaptive discovery probability based on double mersenne numbers. Neural Computing and Applications pp 1–26
48. Reinelt G (1991) Tsplib—a traveling salesman problem library. ORSA journal on computing 3(4):376–384
49. Reinelt G (1994) The traveling salesman: computational solutions for TSP applications. Springer-Verlag
50. Roodbergen KJ, De Koster R (2001) Routing order pickers in a warehouse with a middle aisle. European Journal of Operational Research 133(1):32–43
51. Sallabi OM, Elhaddad Y (2009) An improved genetic algorithm to solve the traveling salesman problem. University of Benghazi
52. Sekar A (2021) Simple tsp using pso. https://www.mathworks.com/matlabcentral/fileexchange/71589-simple-tsp-using-pso, retrieved: Oct. 15, 2021
53. Shi XH, Liang YC, Lee HP, Lu C, Wang Q (2007) Particle swarm optimization-based algorithms for tsp and generalized tsp. Information processing letters 103(5):169–176
54. Snyder LV, Daskin MS (2006) A random-key genetic algorithm for the generalized traveling salesman problem. European journal of operational research 174(1):38–53

55. Sörensen K (2015) Metaheuristics—the metaphor exposed. International Transactions in Operational Research 22(1):3–18
56. Teja PR, Kumaar AN (2018) Qr code based path planning for warehouse management robot. In: 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), IEEE, pp 1239–1244
57. Theys C, Bräysy O, Dullaert W, Raa B (2010) Using a tsp heuristic for routing order pickers in warehouses. European Journal of Operational Research 200(3):755–763
58. Thrun MC, Gehlert T, Ultsch A (2020) Analyzing the fine structure of distributions. PloS one 15(10):e0238835
59. Tompkins JA, White JA, Bozer YA, Tanchoco JMA (2010) Facilities planning. John Wiley & Sons
60. Tsai CY, Liou JJ, Huang TM (2008) Using a multiple-ga method to solve the batch picking problem: considering travel distance and order due time. International Journal of Production Research 46(22):6533–6555
61. Van Gils T, Ramaekers K, Caris A, de Koster RB (2018) Designing efficient order picking systems by combining planning problems: State-of-the-art classification and review. European Journal of Operational Research 267(1):1–15
62. Wang KP, Huang L, Zhou CG, Pang W (2003) Particle swarm optimization for traveling salesman problem. In: Proceedings of the 2003 international conference on machine learning and cybernetics (IEEE cat. no. 03ex693), IEEE, vol 3, pp 1583–1585
63. Wang L, Yin Y, Zhong Y (2015) Cuckoo search with varied scaling factor. Frontiers of Computer Science 9(4):623–635
64. Wurman PR, D'Andrea R, Mountz M (2008) Coordinating hundreds of cooperative, autonomous vehicles in warehouses. AI magazine 29(1):9–9
65. Xiao-Long W, Chun-Fu W, Guo-Dong L, Qing-Xie C (2017) A robot navigation method based on rfid and qr code in the warehouse. In: 2017 Chinese Automation Congress (CAC), IEEE, pp 7837–7840
66. Yang XS, Deb S (2009) Cuckoo search via lévy flights. In: 2009 World Congress on Nature & Biologically Inspired Computing (NaBIC), IEEE, pp 210–214
67. Yang XS, Deb S (2010) Engineering optimisation by cuckoo search. arXiv preprint arXiv:10052908
68. Yang XS, Gandomi AH (2012) Bat algorithm: a novel approach for global engineering optimization. Engineering computations
69. Yang XS, Press L (2010) Nature-Inspired Metaheuristic Algorithms Second Edition. Luniver Press